UNIVERSITY OF CALIFORNIA,
IRVINE

Security and Privacy Challenges in Content-Centric Networks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Christopher A. Wood

Dissertation Committee:
Professor Gene Tsudik, Chair
Professor Marco Levorato
Doctor Ersin Uzun

2017

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1321846

DEDICATION

To Kaitlin.
What a plunge.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

This dissertation is the culmination of many hours of research, design, development, and writing. And after four and a half years, condensing the set of people directly involved in this effort seems most challenging. Starting this list is easy, though. None of this would have been possible without advisement from Gene Tsudik. Thanks to his expertise, experience, and patience, I have grown and am not the same researcher or individual I was four years ago. My sincere gratitude also go to Ersin Uzun, my research mentor and advisor during my tenure at PARC. His support brought me to and kept me at PARC for several years, and is an experience I will always treasure. Lastly, I would like to thank Marco Levorato for his thoughtful lectures and advisement in the early and late years of my time at UCI.

Many other people helped me in one way or another. I would like to thank my colleagues in the SPROUT lab, old and new, who offered critical assessment and thoughtful discussion of many problems while at UCI. I am grateful for our arguments, recess, and evenings out. I would like to especially thank Cesar Ghali, Ivan De Oliveira Nunes, and Tyler Kaczmarek. There was also no shortage of constructive criticism or technical leadership during my time at PARC. I'm indebted to Glenn Scott, Ignacio Solis, Alan Walendowski, Marc Mosko, and Laura Hill for their endless stream of challenges, technical advisement, and (infrequent) light-hearted harassment. I am also thankful to them for launching me into the worlds of the IETF and IRTF. Participation in the ICNRG led to joint work and standardization efforts in part driven by and alongside Dave Oran, Dirk Kutscher, Christian Tschudin, and Börje Ohlman. It was at the IETF that I also became more acquainted with TLS and modern transport security protocols. This interest eventually manifested in the start of a career at Apple as a software engineer while finishing my PhD. This joint venture would not have been possible without support from Lucia Ballard and Ivan Krstić. I am also grateful to my friends and colleagues Tommy Pauly, David Schinazi, Eric Kinnear, Bailey Basile, Frederic Jacobs, Yannick Sierra, and Sean Devlin for constantly reminding me of how little I know and how much I have to learn.

Work-life balance was essential to finishing this dissertation. Thus, naturally, I am thankful to friends outside of my academic and professional circles, especially Sam Skalicky, Greg Knox, and Khrystin Knox, as sources of companionship, relief, and healthy distractions. Similarly, I am thankful for my family, old and new. Although separated by an entire country, my parents still find ways to be supportive and encouraging from afar. I'm also eternally grateful to my brother for his support (and couch) during the early years of my PhD. I am also thankful to the Corbin clan for welcoming me into their home and hearts. Although not official, I consider them family. Finally, and most importantly, I want to thank my fiancé Kaitlin for her endless, unwavering, and unparalleled support and encouragement. She saw me through some of the best and worst times over these past years. She and our dogs, Eris and Olik, made our many Californian apartments feel like home away from the East Coast. I consider myself lucky to come home to them every day.

CURRICULUM VITAE

Christopher A. Wood

EDUCATION

Ph.D. in Computer Science **2017**
University of California, Irvine *Irvine, California*

M.S. in Computer Science **2013**
Rochester Institute of Technology *Rochester, New York*

B.S. in Software Engineering and Computer Science **2013**
Rochester Institute of Technology *Rochester, New York*

RESEARCH EXPERIENCE

Research Scientist **2014–2016**
Palo Alto Research Center, Irvine *Palo Alto, California*

Graduate Research Assistant **2013-2014**
University of California, Irvine *Irvine, California*

Graduate Research Assistant **2010–2013**
Rochester Institute of Technology *Rochester, New York*

PROFESSIONAL EXPERIENCE

Secure Transports Engineer **Fall 2016–present**
Apple, Inc. *Cupertino, California*

Network Software Engineer **Fall 2014 – Fall 2016**
PARC, Computer Science Laboratory *Palo Alto, California*

Security and Privacy Research Intern **Summer 2013, 2014**
PARC, Computer Science Laboratory *Palo Alto, California*

REFEREED JOURNAL PUBLICATIONS

Privacy-Aware Caching in Information-Centric Networking **2017**
IEEE Transactions on Dependable and Secure Computing

Can We Make a Cake and Eat It Too? A Discussion of ICN Security and Privacy **2017**
ACM SIGCOMM Computer Communication Review

Characterization of Small Trees Based on their L(2,1)-Span **2015**
AKCE International Journal of Graphs and Combinatorics

REFEREED CONFERENCE PUBLICATIONS

Namespace Tunnels in Content-Centric Networks **2017**
42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)

Mitigating On-Path Adversaries in Content-Centric Networks **2017**
42nd Annual IEEE Conference on Local Computer Networks (LCN 2017)

When Encryption is Not Enough: Privacy Attacks in Content- Centric Networking **2017**
4th ACM Conference on Information-Centric Networking (ICN 2017)

Closing the Floodgate with Stateless Content-Centric Networking **2017**
26th International Conference on Computer Communication and Networks (ICCCN 2017)

Protecting the Long Tail: Transparent Packet Security in Content-Centric Networks **2017**
IFIP Networking 2017

Mobile Sessions in Content-Centric Networks **2017**
IFIP Networking 2017

Secure Off-Path Replication in Content-Centric Networks **2017**
IEEE ICC 2017 Next Generation Networking and Internet Symposium (NGNI 2017)

(The Futility of) Data Privacy in Content-Centric Networking **2016**
2016 Workshop on Privacy in the Electronic Society (WPES 2016)

Network Names in Content-Centric Networking **2016**
3rd ACM Conference on Information-Centric Networking (ICN 2016)

Trust in Information-Centric Networking: From Theory to Practice **2016**
25th International Conference on Computer Communication and Networks (ICCCN 2016)

BEAD: Best Effort Autonomous Deletion in Content-Centric Networking **2016**
IFIP Networking 2016

Practical Accounting in Content-Centric Networking **2016**
2016 IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)

AC3N: An API and Service for Anonymous Communication in Content-Centric Networking **2016**
2016 IEEE Consumer Communications and Networking Conference (CCNC 2016)

Constructing Large S-boxes with Area Minimized Implementations **2015**
2015 IEEE Military Communications Conference (MILCOM 2015)

Secure Fragmentation for Content-Centric Networking **2015**
IEEE MASS 2015 Workshop on Content-Centric Networking (CCN 2015)

Interest-Based Access Control for Information Centric Networks **2015**
2nd ACM Conference on Information Centric Networking (ICN 2015)

Secure Fragmentation for Content- Centric Networks **2015**
14th IEEE International Symposium on Network Computing and Applications (NCA 2015)

An Encryption-Based Access Control Framework for Content-Centric Networking **2015**
IFIP Networking 2015

Mission Control: A Performance Metric and Analysis of Control Logic for Pipelined Architectures on FPGAs **2014**
2014 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2014)

Flexible End-to-End Content Security in CCN **2014**
2014 IEEE Consumer Communications and Networking Conference (CCNC 2014) Special Session: Information Centric Networking

High Level Synthesis: Where Are We? A Case Study on Matrix Multiplication **2013**
2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2013)

ABSTRACT

Security and Privacy Challenges in Content-Centric Networks

By

Christopher A. Wood

Doctor of Philosophy in Computer Science

University of California, Irvine, 2017

Professor Gene Tsudik, Chair

Today's Internet is aging. Connections are point-to-point and increasingly protected by end-to-end encryption. This reduces security to data transport instead of data itself. Content-Centric Networking (CCN) is a paradigm shift away from this host- and channel-based design. CCN is an architecture for naming, securing, and transferring named data from producers to consumers upon request. Consumers issue interests for named content. Routers forward interests towards producers capable of providing authentic content with cryptographic name-to-data bindings. Once found, routers forward content, in reverse, towards consumers. Routers may also choose to cache content to serve duplicate future interests. Object security, native authenticity, pull-based data transfer, flow symmetry, and in-network services are among the notable characteristics of CCN. In this dissertation, we study security and privacy issues that stem from these architectural properties. Specifically, we study variations and facets of access control, privacy risks and remedies, and network-layer availability attacks and architectural mitigations. For each issue, we describe the problem in detail and explain several countermeasures. We also present detailed analyses and experimental assessments for each approach. We find that sound engineering can mitigate several issues, while others remain insurmountable challenges exacerbated by fundamental security and performance tradeoffs made by CCN.

Chapter 1

Introduction

This dissertation centers on security and privacy in Content-Centric Networking (CCN) We begin by describing the current landscape of computer networks and the public Internet. We then survey next generation networks and Information-Centric Networking architectures, of which CCN is a particular instance. Specific details of the CCN architecture follow. We then identify unresolved security and privacy problems in CCN. The remainder of this dissertation presents a variety of approaches to mitigate these issues. The main contributions are:

- We present two distinct approaches to access control that vary in ease of use, network participation, and application visibility. One based on content encryption – CBAC – and another based on interest encryption – IBAC. CBAC and IBAC are complementary techniques that may be used in isolation or together for enhanced security and privacy. We also describe a network-level content deletion mechanism to aid revocation, a particularly challenging problem in asynchronous and distributed access control designs. My contribution involved designing, implementing, and evaluating the access control designs and content deletion protocol.

- We show that pivotal architectural features make privacy difficult to attain in current CCN. Attacks exploiting statically encrypted content suggest CCN is vulnerable to a new set of attacks not applicable in modern IP-based networks. We describe essential requirements for varying degrees of privacy in CCN. Strong variants of privacy reduce CCN traffic to end-to-end encryption similar to TLS in IP-based networks. We then present a number of protocols and system designs that can be used to improve privacy of all entities. Approaches include Tor-like tunnels, network-layer tunnels, and application-layer key exchange and end-to-end encryption. Each targets different adversaries and operate at different layers in the network stack. My role in this segment involved formalizing notions of privacy in CCN and studying the extent to which privacy is feasible with static content. Building on these results, I designed and evaluated systems and protocols to enable or improve privacy.

- We present a variety of availability or denial-of-service attacks on the CCN architecture and constituent applications. Examples include standard resource exhaustion in the data plane and prefix hijacking (content reachability) attacks. Some attacks can be mitigated with network-layer modifications. We argue that others, such as the classic Interest Flooding attack, mandate more substantial architectural changes. To that end, we propose a CCN modification that aims to mitigate this class of attacks in standard deployments. We also show how packet formats can be slightly modified to reduce router-specific computational DoS vectors. Lastly, we propose a technique for lightweight and efficient network-layer integrity checks that help detect prefix hijacking, or on-path, attackers. My involvement in this area included exploring and evaluating incremental modifications to the CCN network layer, as well as a complete re-design of the data plane.

1.1 The Internet and Modern Network Stacks

Today's Internet is built on the Internet Protocol (IP) [256]. IP is conceptually simple. Every packet, or datagram, has an IP source and destination address, along with some additional control information. Every network entity, e.g., host or router, is identified by an IP address. IP addresses have two parts: (1) network prefix and (2) host identifier. This allows IP addressing to scale with the number of network hosts. Routers, or packet switches, use addresses to move packets from source to destination. A Forwarding Information Base (FIB) maps IP address prefixes to interfaces. When processing a packet, a router performs a longest-prefix-match (LPM) query on the FIB and, if a match is found, forwards the packet to the corresponding interface. If the packet is larger than the destination link MTU, it can be fragmented and subsequently reassembled by the receiver.[1] If a packet cannot be forwarded, it is dropped and an error message is generated according to the Internet Control Message Protocol (ICMP) [257]. Routing protocols such as BGP [296], iBGP [166], and RIP [167] populate router FIBs. These enable packets to be forwarded within and between networks in the Internet.

IPv4 was the first version of IP. IPv6 is the next incarnation [100]. In addition to extending the address space, IPv6 enhances IPv4 in several ways:

1. Fragmentation is not allowed. Endhosts must fragment packets based on the path minimum MTU (PMTU). PMTU discovery is possible using techniques such as those described in RFC 1981 [218].

[1]Not all fragments can be fragmented. Those which requirement fragmentation yet carry a "do not fragment" flag are dropped instead.

2. Packets can carry optional extension headers (EHs) to modify packet processing behavior. Examples of headers include AH and ESP protocol headers (see below). This obviates the need for framing to encode these protocol messages.[2]

Neither IPv4 nor IPv6 provide origin authenticity, data integrity, or confidentiality. Any part of an IP packet can be modified or forged. IPsec is an extension to IP that provides some security features to mitigate these attacks. The Encapsulating Security Payload (ESP) [185] protocol encrypts and authenticates IP packets. The less-used Authentication Header (AH) [184] protocol only provides origin authentication and integrity. Both ESP and AH can be run in either tunnel and transport mode. Tunnel mode is used between two gateways or routers that share keying material. Endpoints encrypt and encapsulate entire packets over this tunnel. In contrast, transport mode offers end-to-end protection of (only) IP packet payloads. The Internet Key Exchange (IKE) [182] protocol is used to establish ESP and AH protocol keys. As an extension, IPsec is not used by default – it must be explicitly enabled by the endpoints. The typical use of IPsec is to establish Virtual Private Networks (VPNs) in IP-based networks.

Transport protocols such as TCP [258] are built on top of IP. TCP in particular provides a reliable, in-order, byte stream interface. TCP also provides flow and congestion control. Applications may send and receive arbitrary-length application data over TCP. The protocol is responsible for segmenting data into segments, which are encapsulated into individual IP packets that are then sent across the network. End-hosts can support multiple TCP streams (or flows) simultaneously. Each one is identified by a unique port number. In contrast to TCP, User Datagram Protocol (UDP) [255] is a much simpler transport protocol that supports lightweight connectionless delivery of arbitrary-size datagrams. As with TCP, each UDP process is identified by a unique port number.

[2]IPv6 and IPv4 headers may still precede AH and ESP headers. Extensions are not needed for these protocols.

End-to-end transport security protocols are built on top of transport-layer protocols. For example, Transport Layer Security (TLS) [107] and the more recent Quick UDP Internet Connections (QUIC) [20] protocol are built on top of TCP and UDP, respectively. Other transport security protocols include tcpcrypt [60], CurveCP [54], and MinimalT [252]. These all enable end-to-end (authenticated) encryption where possible, and may subsume or duplicate similar functionality at the network layer, e.g., ESP.

Secure transport protocols such as TLS are slowly becoming the thin waist of the Internet and bedrock of modern network stacks. For example, many ubiquitous application protocols, such as HTTP and DNS, map to TLS. This is done to limit information leaked to (untrusted) networks and ensure origin authenticity. However, end-to-end encryption raises challenges for existing networks and technologies. Standard techniques such as deep-packet inspection and differential treatment become infeasible since middleboxes cannot inspect encrypted packet payloads [223]. These are often useful for detecting specific types of attacks, e.g., mail abuse and SPAM, phishing, botnet activity, etc. Other standard practices such as measurement, congestion management, load balancing, content compression, and use of performance-enhancing proxies (PEPs) are all affected by the proliferation of end-to-end encryption [238, 223], leading to adaptive solutions [284, 239].

To cope with these challenges, network operators often deploy TLS and other connection terminators close to clients in geographically localized points-of-presence (POPs). Typical traffic inspection is possible after secure connection termination. This is a common design for modern content-distribution networks (CDNs). Such CDNs are composed of pre-positioned caches that terminate TLS connections and typically serve cached data so as to avoid expensive client-to-server round trips. Through mechanisms such as anycast, client connections are redirected to geographically close CDN nodes.

After TLS connection termination, application data flows through CDN POP pipelines. Typically, part of this pipeline involves checking for cached resources and, if absent, forwarding

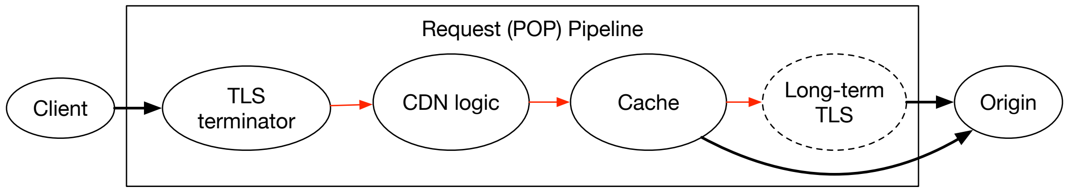


Figure 1.1: Example CDN architecture

requests to origin servers. Response are then sent back, cached, and served to clients. This type of flow is shown in Figure 1.1.[3] Application data remains protected, albeit visible, within the CDN network, and is (usually) encrypted in transit between the CDN and origin servers.

In conclusion, modern network stacks begin with TLS or similar secure transport protocols. End-to-end encryption forces end-to-end communication semantics and encourages overlay systems to combat scalability problems and improve network performance. Thus, deviating from these architectural patterns may require a change at the network layer. It is precisely this idea and endeavor that we focus on in the remainder of this thesis.

[3]CDNs such as Fastly, Akamai, and Cloudflare have similar architectures.

Chapter 2

Next-Generation Networks

IP is aging. Over time, its limitations have been addressed piecemeal by protocols such as IPsec and TLS, and massive deployments of technologies such as CDNs. Many alternative network-layer architectures were designed, developed, and even deployed in the past decade in an attempt to move beyond the IP-based Internet. The National Science Foundation (NSF) Future Internet Architecture (FIA) program facilitated many such efforts by funding the following projects: eXpressive Internet Architecture (XIA) [165], MobilityFirst [280], Nebula [35], and Named-Data Networking (NDN) [342]. NDN, unlike the others, is a type of Information-Centric Networking (ICN) architecture, along with CCN. Other prominent ICN architectures include the Data-Oriented Network Architecture (DONA) [192], Network of Information (NetInf) [112, 96], ICEMAN [332], and PURSUIT [124]. In this section, we overview these architectures, except for CCN and NDN, which are discussed in Chapter 3.

2.1 Future Internet Architecture Candidates

2.1.1 eXpressive Internet Architecture

eXpressive Internet Architecture (XIA) [165] is a *principal-centric* network architecture. There are four primary principal types: network, host, service, and content. Network and host-centric networking supports end-to-end communication between two peers, e.g., video communication and file sharing. Service-centric networking enables access to services, e.g., databases and file servers. Content-centric networking supports direct access to data and enables, among other features, content distribution. Each principal has an intrinsic identifier, denoted XID, where X is the type of principal. Thus, HID, SID, and CID denote a host, service, and content identifier, respectively.

XIA is amenable to new principals, provided each principal can authenticate itself to others. Typically, this is done by cryptographically binding a principal's name, or identifier, to one or more security properties or characteristics. For example, a host or service may use the cryptographic hash of its public key as its name. For hosts, this ensures that each host has a unique name across the entire network. Content may be authenticated by using the hash of the content as its name.

One additional requirement is that each network entity must be capable of expressing its intent, e.g., support for specific services or availability of content. This permits routers to perform principal-specific behavior when processing or forwarding packets, as shown in Figure 2.1.

XIA uses the eXpressive Internet Protocol (XIP) to forward packets based on XIDs. XIP defines the packet format, addressing scheme(s), and router processing behavior. XIP "addresses" are XID trees, i.e., XID directed acyclic graphs (DAGs), as illustrated in Figure 2.2. Each DAG may have multiple leaf nodes (sinks) and only one root (source) node. An edge

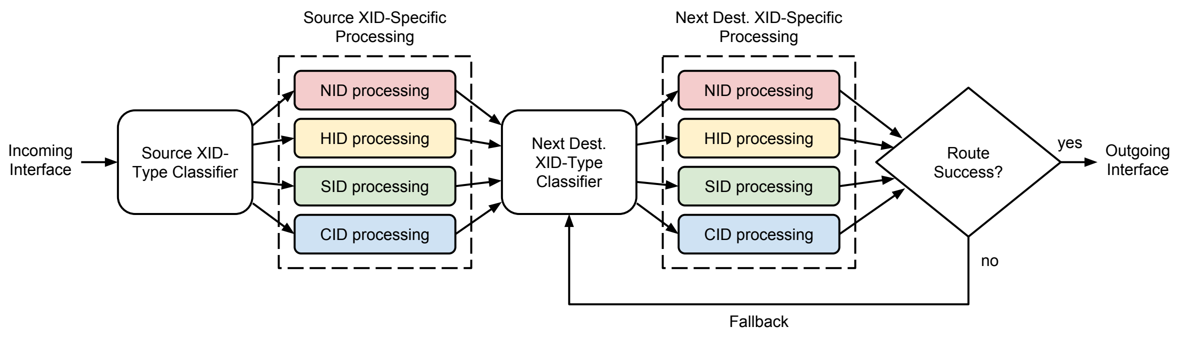
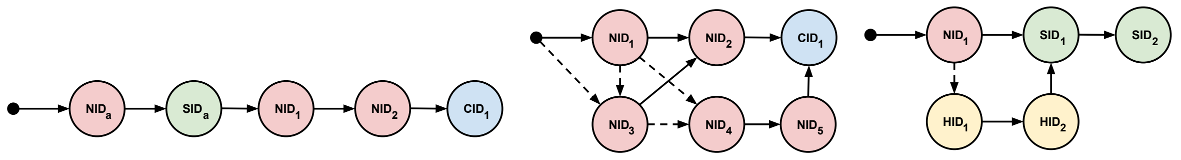


Figure 2.1: XIA router [165]

represents a link between two principals. XIP addresses may specify any DAG, provided they are single connected components [165].



(a) Source routing (b) Multiple paths (c) Infrastructure evolution

Figure 2.2: XIP addressing styles [165]

XID trees enable in-network processing and reliable path selection while permitting principal type expansion. They also permit several interesting forms or styles of addresses, as shown in Figure 2.2. Common variations include:

- Source routing (Figure 2.2a): A source chooses the principals through which the packet will flow.
- Multipath routing (Figure 2.2b): Packets carry multiple paths.
- Path evolution (Figure 2.2c): Host-based path alternatives are offered as services are rolled out.

XIA routers process XIP packets as shown in Figure 2.1. After identifying the source and destination types, principal-specific processing steps are performed. Source-specific processing is not necessary, though it may be used for CID requests to inspect or populate a router content cache. Destination-specific processing is done to determine the appropriate forwarding behavior based on the type of the next hop principal, as indicated by the XIP address. A router attempts to forward a packet to all possible destination XIDs in the order they are constructed in the address, i.e., each outgoing XID address edge is tried in sequence. If none succeed, the packet is dropped and an unreachable error is generated. Thus, users are responsible for providing fallback paths – typically composed of specific HIDs – to ensure correct packet forwarding. Moreover, all routers must implement HID and NID (network ID) functionality to ensure that "standard" host-based addresses are available.

Depending on the principal type, addresses may be formatted differently. For example, to address host HID inside network NID, derived from the public key of the corresponding host and network, an address of the form NID:HID is used. (NID and HID authentication mitigates address spoofing, DoS, and cache poisoning attacks.) Similarly, the address of a service with SID, offered by host HID in network NID, is NID:HID:SID.[1] Content addresses, i.e., those with a CID, can be extensions of a service address or, depending on the application, unbound from services entirely.

2.1.2 MobilityFirst

As the name suggests, MobilityFirst is a network architecture focused on pervasive node mobility [8, 280]. Every principal, including devices, content, interfaces, services, and even human end-users, is a uniquely identifiable and addressable network entity. MobilityFirst separates the locator, or dynamic network location, of a principal from its constant and unique identity. This permits mobility via network locator updates, provided by the distributed Global Name Service (GNS), and facilitates service implementation and location-agnostic deployment [240].

[1]This is analogous to the IP address and port combination that is ubiquitous in today's Internet.

MobilityFirst network layer is responsible for routing packets using locators and identifiers. (Technically, however, both are a form of identifier.) A principal identifier, or *Global Unique Identifier* (GUID), is cryptographically bound to a unique value associated with the corresponding principal. For example, a GUID can be the hash of a service's public key, or the hash of a specific content. GUIDs may be associated with human-readable names through another external service called the Name Certification Service (NCS). GNS resolves these names to GUIDs. (We describe the GNS in more detail later.) GUIDs can also be grouped into sets, called multicast GUIDs. Unlike IP multicast addresses, these are used to support multihoming, anycast, and multicast among *principals*.

A network locator, or *Network Address* (NA), is a flat identifier bound to a specific network. NAs are akin to AS-es in today's IP-based Internet, as they collate hosts, routers, and services under common address spaces. Multihomed hosts, e.g., those with a WiFi connection to an enterprise network and cellular connection to a carrier network, may be associated with multiple NAs.

(NA, GUID) tuples serve as fully-formed principal addresses. Packets directed to principals are transported in Packet Data Units (PDUs), which carry the destination address, payload, and additional metadata and control information. NAs are used to route PDUs between networks using an inter-domain routing protocol. Within a network, routers forward PDUs to specific principals using GUIDs. Routers may also optionally cache GUID packets to reduce traffic. Moreover, if the network fails to deliver a PDU to the destination GUID, then it can store the PDU, in one or more routers, and periodically query GNS for a new (NA, GUID) binding to re-route it accordingly. Endpoints must know or be able to derive the GUID with which they want to communicate. For example, for a unique address content generated by a specific producer, GUID may be derived from content GUID (CID) and producer GUID (PID). This is precisely how content distribution networks are built in MobilityFirst.

Since principals are assumed to be mobile, endpoints must query GNS with GUIDs to obtain fresh principal NAs. Internally, GNS is composed of two services: (1) NCS and (2) *Global Name Resolution Service* (GNRS). NCS is analogous to today's Certificate Authority (CA), which is tasked with creating trustworthy bindings between standard identifiers (GUIDs) and human-readable representations. MobilityFirst does not mandate a common root of trust across NCSs; each principal is free to choose its own NCS(s) to trust. GNRS is a resolution service backed by a distributed database, similar to DNS[2], that stores mappings from GUIDs to NAs [231, 207, 313]. Clients can query GNRS with a GUID to obtain an NA mapping or update GUID to NA mappings. GNRS offers a secure insert and update variant [207]. Clients sign their insert and update queries, which are subsequently verified and validated by border routers. (Validation is done by, e.g., querying the DHCP server to validate client NAs.) When a GUID to NA mapping is updated, the border router forwards it to the appropriate GNRS replica to be inserted into the distributed database.

2.1.3 Nebula

Nebula [35, 36, 37] is a future Internet architecture that integrates confidentiality, integrity, and availability into the design. It is composed of three tightly coupled parts:

- NEBULA Data Plane (NDP): Responsible for securely moving packets along mutually agreed-upon paths.
- NEBULA Virtual and Extensible networking Techniques (NVENT): Responsible for establishing trustworthy routes between nodes based on policy routing [43] and service naming [245].

[2]GNRS differs from DNS in several respects. First, GNRS does not restrict entries to structured, hierarchical names. Second, GNRS does not use TTL-based caching for tracking the liveness of mapping entries. Third, GNRS does not statically delegate authority for names to specific servers, thereby allowing replication responsibilities to be actively distributed.

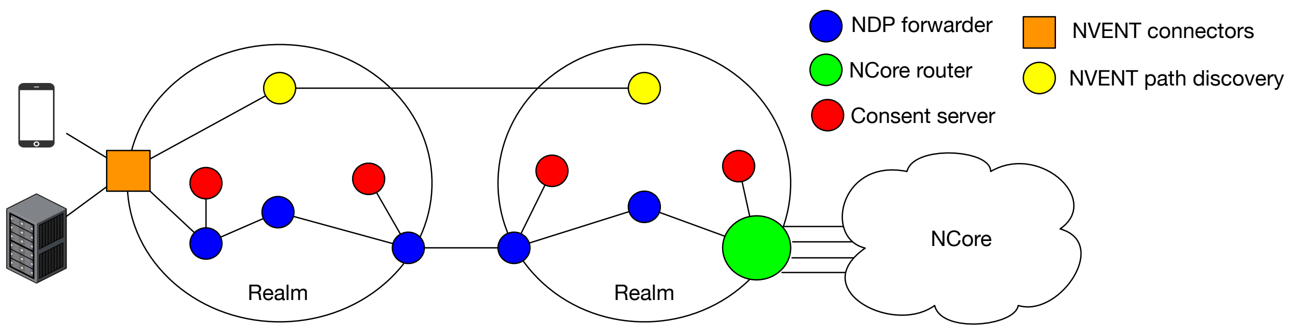


Figure 2.3: Nebula architecture

- Network core (NCore): Large, interconnected collection of enterprise data centers. Data is replicated across data centers to ensure reliability and availability. NCore is based on specially-designed, high-performance core routers [205].

These components are shown in Figure 2.3.

NDP is based on a *Path Verification Mechanism* (PVM), which defines packet forwarding protocol and mechanics. PVMs provide path consent, which is when that every entity on the path has agreed to participate, and path compliance, which is when every hop on the agreed-upon path can verify packet provenance. ICING [235] is one PVM that provides these properties; others include TorIP [206], Transit-as-a-Service [251], and plain IP. ICING can run at the network layer, composed of ICING nodes, or as an overlay, composed of ICING endpoints. Each ICING hop can verify a packet's path compliance cheaply before dedicating any additional (upstream) resources to forwarding the packet.

Informally, a packet is sent as follows. First, a client requests a path P to a service through NVENT, the control plane manager. (We describe service discovery later.) Paths target services and terminate at NCore, the data center interconnect. A path is essentially a sequence of ICING node IDs, each of which is a public key. Paths can be established using any policy mechanism [235]. Once acquired, NVENT obtains permission to use the path from each ICING hop. Permission is granted in the form of a Proof of Compliance (PoC). This is a cryptographic token generated by a per-domain consent server trusted by each forwarding node. Each PoC asserts that the entire path is permissible and supported by the transit policy. Once NVENT returns all PoCs needed for the path to the client, the latter creates the initial packet header by computing the following fields.

- Proof of Provenance (PoP) token, for each node, computed using key $k_{i,j}$ shared between nodes i and j ($i < j$). (To begin, $i = 0$ since the client is the first hop in the path.) A PoP is a message authentication code (MAC) over the packet. Moreover, PoP keys are not pre-distributed or exchanged symmetric keys; they are derived using non-interactive Diffie-Hellman with (a) the sender's private key and (b) receiver's public key (ID).

- Authenticator A_j for each node j using PoC_j, P, and M.

For a path of length l, the i-th PoP token is XOR'd with A_i to form a verifier V_i, where $0 < i \leq l$. Verifiers are checked as a packet moves throughout the network to ensure path compliance and provenance. Assume, for example, that NDP router R_p ($1 \leq p < l$) receives a packet with l verifiers, $V_1, \ldots, V_l$, constructed as follows:

$$V_i = A_i \oplus_{j=0}^{\min\{p-1,i\}} \text{PoP}_{j,i}$$

To verify V_i, for $i \leq p$, R_p first recomputes PoC_i, A_i, and $\text{PoP}_{j,i}$ for each downstream hop R_j. R_p then uses these values to recompute V_i. If the verifiers match, then each verifier V_j, $j = p+1, \ldots, l$, is modified as follows:

$$V_j = V_j \oplus \text{PoP}_{p,j}$$

ICING packet headers are modified in place at each hop until the packet reaches the destination or an NCore data center, where it is forwarded to the correct service provider.

Nebula paths are service-centric, not host-centric. (Transition from host names to services aids client mobility and distributed service generation.) Clients resolve service names, or IDs, to paths using a service resolution service in NVENT. Service providers, which emanate from NCore, use NVENT to request services with certain properties, such as high availability or reliability [35]. NVENT handles route creation to satisfy desired properties. For example, in the case of a high availability service, NVENT might use multi-path inter-domain routing when building a path.

Serval is the de facto service resolution mechanism used by end-hosts in the Nebula architecture [245].[3] Beyond adopting service IDs in lieu of host names, Serval keeps end-hosts unaware of Nebula via a transparent Service Access Layer (SAL). This layer also replaces IP and port connections with interface-agnostic flow IDs. A sender and receiver use flow IDs to identify connections. This permits client mobility in the presence of network address changes such as NAT re-bindings.

2.2 Information-Centric Networking

Content is the primary focus of many emerging applications. Yet, existing networks are focused on hosts, not content. Moreover, in-network processing becomes increasingly problematic with the proliferation of protocols such as TLS.[4] Technologies such as CDNs are content-aware and can store and forward content closer to consumers. According to [168], this trend will only increase, leading to increased reliance on end-to-end encryption and more widespread deployment of content-aware overlay networks.

Information-Centric Networking (ICN) is a network paradigm that moves away from today's host-based design. Content, rather than endpoints, are the primary addressable units in the

[3]Serval does not support service discovery – only resolution.
[4]As of 2017, HTTPS adoption is quickly growing on all modern platforms, including Android, Windows, Mac, and Chrome OS [116].

network. Producers publish content (or ways to access content), and consumers request, or subscribe to, this content. Traffic consists predominantly of content requests (subscriptions) and responses (publications).

Content is addressed by a specific *name* that is securely bound to data. This allows consumers to verify the name-to-data binding. Moreover, by naming data, network nodes, or routers, can opportunistically cache content in hopes of satisfying future requests for the same content.

Request and response forwarding mechanics vary across ICN incarnations. Some ICN architectures buffer requests for content that cannot be satisfied locally via a cache or remotely by forwarding to another hop, whereas others may drop the requests immediately. Mechanics aside, named content is requested by consumers, generated securely and published by authoritative producers, and cached unilaterally in the network [152].

In this section, we describe several prominent ICN architectures, including: Data-Oriented (and Beyond) Network Architecture, Network of Information, and ICEMAN. NDN and CCN are deferred to the following chapter.

2.2.1 Data-Oriented (and Beyond) Network Architecture

Data-Oriented (and Beyond) Network Architecture (DONA) is a network architecture built on top of IP [192]. As the name suggests, DONA focuses on data and the principals (publishers) of data. Each principal is associated with a public and private key pair. An address for a specific piece of data is composed of a publisher's public key digest (P) and label (L) assigned by the publisher, denoted P:L. L can be set to the cryptographic data hash to ensure uniqueness. A data packet with the name P:L is digitally signed using the private

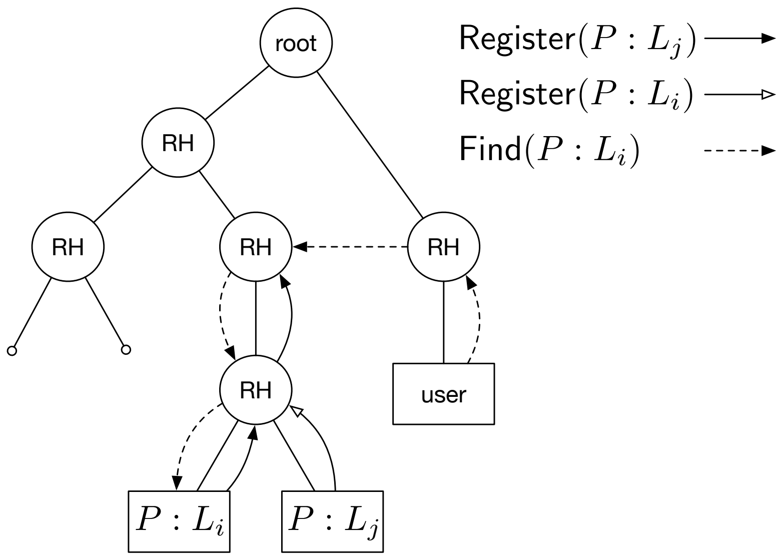


Figure 2.4: Example DONA RH tree

key associated with P. (Trust management, i.e., determining which P's are trustworthy, is deferred to the application.)

DONA principals can delegate to other hosts to serve data on their behalf. In such cases, P:L tuples remain unchanged, thereby permitting host-based or service-based mobility and redundancy.

Two fundamental name-resolution messages are supported: FIND and REGISTER. Each message carries a name, e.g., FIND(P:L). So-called Resolution Handler (RH) nodes route these messages. RH nodes are arranged in a hierarchical, DNS-like fashion to support routing scalability, as shown in Figure 2.4.[5] DONA requires every domain have at least one logical RH, though it may be implemented by a series of physically interconnected RH nodes. Also, the root RH has visibility of the entire network. RHs are assumed to have shared public keys so that they can authenticate all received packets.

Each RH contains a *registration table* (RT) that maps names to adjacent nodes, i.e., parents, peers, or children, and the distance to the name. It may also contain an optional cache

[5]Domains can vary in size from a small collection of nodes to an entire AS.

that is used to store previously forwarded data packets. RHs update their RTs, if necessary, upon receipt of a REGISTER(P:L) packet. RHs may also forward registration packets to their parents or peers depending on RT contents and local policy, similar to BGP [166]. Examples of policies include refusing to serve a peer absent some financial agreement. Also, registration effects are temporary since name entries have a TTL; principals must send fresh registration messages at or before TTL expiration.

An RH attempts to forward a FIND(P:L) packet its closest destination using RT contents. If an RT entry for P:L exists, the packet is forwarded downstream to the corresponding hop. If there are no viable RT entries, the packet is forwarded to the parent RH. This repeats recursively until the packet reaches the root, at which point it will be forwarded down the tree towards the correct destination, or dropped.

Unlike other architectures, the DONA overlay exists to route data requests. Once found, transfer of this data replica from the storing host to the FIND requestor is initiated and deferred to IP (and transport) layer(s). The FIND requestor may be a user or RH, depending on the IP source address in the packet. An RH can set itself as the originator of a FIND message to receive and cache data replicas before transmitting to the original requestor. Thus, DONA does not require modifications to the IP infrastructure and can exploit existing transport mechanisms and protocols for bulk data transfer.

Despite its simplicity, DONA is not free of security issues. Denial of Service (DoS) is a major concern. RHs can refuse to propagate REGISTER or FIND messages. Also, malicious consumers can flood the RH overlay with nonsensical names to overlay nodes near the RH tree root. DONA assumes rate limiting mechanism(s) at the IP layer and requires application-imposed restrictions for the number of FIND and REGISTER messages permitted per second (or any other unit of time). If trust is not subverted, only DoS attacks by malicious entities are possible. Lastly, DONA defers principal key revocation to applications.

2.2.2 Network of Information

NetInf is an ICN architecture for transferring Named Data Objects (NDOs). The architecture is designed to support creating, locating, exchanging, and storing NDOs. To ensure consistency, each NDO is associated with at least one name, or Network Information (NI), derived from information associated with the NDO [28, 112, 96]. In the case of static content, this is done with the content hash (and hash algorithm) in the NI, e.g., `ni:///sha-256;UyaQV-Ev4...` [113].[6] The hash may also be computed over the public key of the NDO producer, or Information Owner (IO). This is referred to as a dynamic NDO. NIs may also contain human-readable segments, e.g., `ni://example.com/sha-256;f4OxZX`. Consequently, NDO names are typically flat identifiers that simultaneously serve as locators and identifiers. This permits mobility across network domains.

NetInf data is moved via a request-response protocol. There are three message types [112]:

- GET: a request for a specific NDO, identified by the NDO's name and, optionally, a network locator. NetInf nodes may reply to GET requests with locators where the desired NDO may be found, i.e., another network or a specific host.
- PUBLISH: an announcement for a specific NDO name. PUBLISH messages may optionally carry a network locator or the actual NDO itself alongside the announcement.
- SEARCH: a query for a set of NDOs satisfying one or more keywords. SEARCH messages are often issued to translate human-readable keywords to specific NDOs.

Nodes cannot authenticated NetInf messages. Requests are inherently insecure and cannot be trusted. In contrast, since each NDO is bound to some security context, e.g., the data hash or publisher's public key, any entity may verify the integrity of an NDO. Thus, integrity is an intrinsic property of NetInf. As suggested by Dannewitz et al. [95], it is advantageous

[6]The NI grammar and encoding details is described at length in [113].

to verify both NDO contents and publisher provenance. This is done by (a) including the IO key in the dynamic name and (b) signing NDO content hash with the IO key. Verifying an NDO then requires one to verify the signature, rather than simply compare hashes for equality.

One consequence of dynamic NDOs is that identities are included in names, leaking information about the publisher. Pseudonymity is possible by using rotating identifiers for each NDO.

NetInf message routing uses Name-Based Routing (NBR) and a Name Resolution Service (NRS). Routers use NBR to forward `GET` messages based on NDO names and local routing table. NRS is a supplemental service designed to map NDO names to network locators. A locator identifies the network where the desired NDO is stored. As mentioned above, they can be sent with `GET` messages to help aid routing decisions. (In this way, a locator is similar to an IP address.) Typically, clients use NRS to obtain NDO locators. Routers may also use NRS if they receive a `GET` message they cannot forward further. In this case, a router may (repeatedly) query NRS to obtain a locator before forwarding the message.

In contrast to requests, there is no mandated forwarding mechanism for responses. For example, routers may store per-packet state used to forward `GET` responses in the reverse direction, i.e., to the interface on which they were received. Alternatively, a router may store this state in the packets via labels. Routers may also choose to cache forwarded NDOs to satisfy future `GET` messages. Off-path caches are permitted. Routers may forward `GET` messages to nearby off-path caches that have the desired content.

Clients may also cache NDOs and advertise their own locators. This permits peer-to-peer content sharing at the cost of client privacy.

2.2.3 ICEMAN

ICEMAN (Information-CEntric Mobile Ad-hoc Networking) [332] is an event-based ICN architecture designed for tactical environments where fresh information needs to be distributed quickly and reliably. It supports three types of information dissemination: (1) localized flooding and replication within a single connected network component (originating from the producer), (2) interest-driven routing, and (3) mobility-driven routing. Flooding quickly pushes information to the network and proactively caches content for quick access. Similarly, interest- or request-driven routing proactively fetches information that might be needed. Unlike other ICN architectures, per-request state is not removed once a request has been satisfied. This is done so that future updates to the same content can be routed to intended recipients. Mobility-based routing is built on PRoPHET [204], a probabilistic routing protocol for networks where connectivity is intermittent, yet predictable.[7]

Each content C carries a payload $P(C)$, associated metadata, $M(C)$, a timestamp of creation, and identity (hash) of the payload data. Content metadata consists of key-value attribute pairs for the data and is used as the basis for matching. There are two types of content: application data and control information, such as ICEMAN node descriptors and routing updates. The latter is used to configure and maintain paths, as well as to make forwarding decisions. For example, nodes share descriptors of their cache contents with neighbors in the form of bloom filters (BFs). Content is only forwarded to a specific neighboring node if there is no match in the neighbor's BF. Each content also has an associated *scope* $S(C)$ that defines the nodes, including consumers, which are eligible receivers of the content. Content is not forwarded to ineligible nodes.

Interests for content originating from node S are expressed as predicates $I(S)$, which are weighted key-value (or attribute-value) pairs. That is, each pair is a tuple of some key

[7]PRoPHET was originally designed for use in Delay Tolerant Networks (DTNS).

(attribute), value, and weight, denoted (a, v, w). An interest predicate matches a content if the sum of weights of overlapping attributes between the predicate and content metadata exceed some threshold. Content with the largest match weight and freshest timestamp is returned in response to an interest.

At each hop, content is cached according to some utility function [311, 77]. Content that does not meet a minimum utility is immediately discarded. Utility functions may be content and context-sensitive, i.e., vary over time. Content is evicted when space is needed or some pre-determined amount of time has elapsed. ICEMAN users can specify tags (or classes) of content to purge based on content attributes, as well as some maximum freshness threshold that will trigger eviction. At the time of eviction, fresh data is preferred over stale data, following the caching strategy outlined by Kim et al. in [189]. Each content's utility is also a factor in the eviction strategy, such that the goal is to reduce the number of items in the cache while simultaneously maximizing utility.

As an ICN architecture, ICEMAN emphasizes content-based instead of channel-based security. However, in contrast to other architectures, both authenticity and confidentiality are core services of the network. Multi-authority attribute-based encryption [57, 199] is used to encrypt content, where attributes are assigned to content metadata. Content is signed with keys certified by one or more authority. Thus, ICEMAN depends on a PKI where trust is anchored in one or more authorities.

2.2.4 PURSUIT

PURSUIT [124] is a publish-subscribe ICN architecture that emerged from the FP7 EU project [17] and its predecessor, Publish-Subscribe Internet Routing Paradigm (PRSIRP) [304, 122, 126]. Information in PURSUIT is identified by a name, composed of a Rendevouz ID (RID) and Scope ID (SID) tuple. RIDs identify a specific piece of information in the cor-

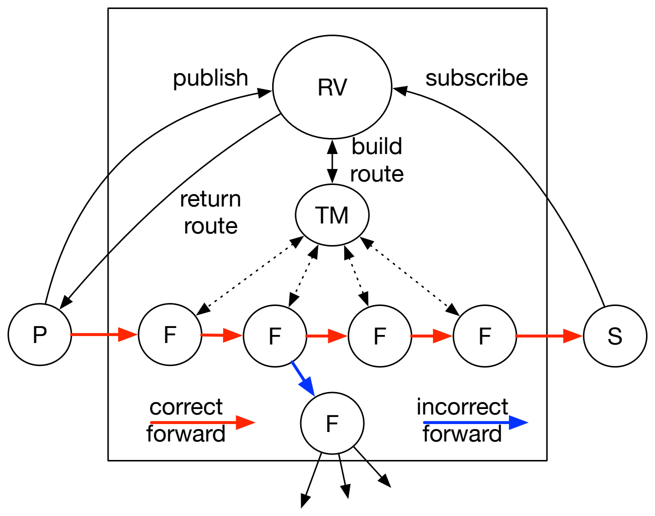


Figure 2.5: Example PURSUIT network

responding SID. RIDs can also belong to multiple SIDs. Thus, a single RID can be resolved via multiple names, each with a different scope. Scopes can be arranged hierarchically to control information access. Subscribers may specify a sequence of SIDs with a single RID, e.g., ($SID_1/SID_2/SID_3$, RID). Publishers typically use scopes to restrict access to a particular content.

PURSUIT networks consist of three primary entities: subscribers, publishers, and network brokers. Each network broker contains a Rendevouz Node (RN), Topology Manager (TM) and Forwarder Node (FN). RNs are responsible for matching subscriptions with publications with a specific scope. TMs monitor the network and construct pre-defined publisher-to-subscriber paths. These are delivered to publishers by RNs. TMs run a routing protocol between each other in order to move scoped publications or subscriptions to the correct broker. Lastly, FNs are responsible for moving packets from publishers to subscribers. This basic architecture is shown in Figure 2.5.

Links between entities are identified with Link IDs (LIDs). Each LID is a flat binary string (of some suitably large length). When constructing paths between publishers and subscribers, TMs construct a single routing label, called a zFilter, equal to the bitwise OR of each LID along the desired path. For example, if a packet must traverse LIDs l_1, l_2, and l_3, then the zFilter is $(l_1 \vee l_2 \vee l_3)$.

When a forwarder receives a packet, it performs a bitwise AND with each of its outgoing LIDs. The packet is forwarded to all links for which this result yields the same outgoing LID. Depending on LID construction, this may result in packets being forwarded to incorrect destinations.

Chapter 3

Content-Centric Networking

This chapter describes the Content-Centric Networking (CCN) architecture and core protocol. We begin with an overview of the architecture, design principles, and protocol messages between producers and consumers. We then describe how the network routes and forwards CCN packets. We finish with a brief survey of security and privacy problems unique to CCN and related ICN architectures.

3.1 Overview

CCN is a network architecture for transferring named data, or content, from producers to consumers upon request [226, 172]. (Producers generate or publish content under a given routable prefix.) Names are composed of one or more variable-length segments opaque to the network layer. Segment boundaries are explicitly delimited by "`/`" in the usual URI-like representation [53, 229]. For example, the name of a BBC's news homepage for May 20, 2017 might be: `/bbc/news/05-20-2017/index.html`.

To obtain content, consumers issue a request, or interest, carrying the data name. The network is responsible for forwarding requests towards authoritative producers capable of providing content responses. Once located, routers forward the content back to the consumer along the reverse path the interest traversed. Each router may choose to cache the content to satisfy future requests. This type of opportunistic caching serves three primary purposes: (1) to support error control or recovery, e.g., when packet loss occurs downstream and consumers retransmit interests, (2) to serve multiple consumers requesting identical content in (possibly rapid) succession, (3) to help smooth out link and path variation over lossy or heterogeneous networks.

As a consequence of opportunistic in-network caching, data may be served from entities (routers) other than the original producer. This means that consumers *must* be able to ascertain authenticity of each content object. This is especially important for caching routers; one core rule in CCN is that a router must never serve content from its cache that has not been verified. Otherwise, a malicious producer or router could inject and spread fake, or poisoned, content through the network.

The standard way to prove content authenticity is via a producer-generated digital signature. Signing content and its corresponding public verification key (or certificate) permits any entity to verify authenticity. In cases where the producer and verifying parties share a secret, such as a MAC key, the producer may compute and attach a MAC and shared key identifier to content in lieu of a signature and public verification key. Routers without the shared key cannot cache such content.

Taken as a whole, CCN has the following features:

1. Cryptographic name-to-data bindings.
2. Pull-based data transfer.

3. Symmetric traffic flow.

4. In-network services, e.g., opportunistic caching.

To the consumer, the network is a generic key-value service that maps names to content with cryptographic assurances. To the producer, the network is a service which provides data delivery to consumers. Routers merely facilitate the exchange of data for both services.

3.2 Forwarding and Matching Semantics

Each CCN node, i.e., consumer, producer, and router, is comprised of at least two components:

- Forwarding Information Base (FIB): a table that maps name prefixes to egress link identifiers using longest-prefix match (LPM). Interests are forwarded to links identified by the FIB. Prefix-matching is done on name segments, not name bytes (or bits). For example, $N_p =$`/foo` is a prefix of $N =$`/foo/bar`, but not a prefix of $N =$`/fo/obar`.

- Pending Interest Table (PIT): a high-speed cache-like structure that stores names, content identifiers, ingress link identifiers, and other miscellaneous information for pending, i.e., unsatisfied, interests.

CCN nodes may also contain an optional cache, or content store (CS).[1] The cache is indexed by the complete data name.

The dataplane of a router R, as shown in Figure 3.1, works as follows. When R receives an interest, it examines the name to determine if it has a copy of the content stored in its cache. If it does, R transmits the matching content object in reply to the interest on the arrival

[1] We use the terms CS and cache interchangeably in the remainder of this dissertation.

interface. (We describe the matching rules below.) Otherwise, R records some state derived from the interest, e.g., the arrival interface, in the PIT so as to provide a reverse path to the requester. If there is no matching PIT entry, the router transmits the interest to the next hop(s) specified in its FIB. If there is no matching FIB entry, an Interest Return, or network-layer NACK [86], is sent to the interest arrival interface. If there is a matching PIT entry, the new interest is *aggregated* with the existing pending interest and not forwarded further. When a content response is later returned to R, it is forwarded to all downstream interfaces listed in the matching PIT entry. R may cache the content in anticipation of a future interest with a matching name.

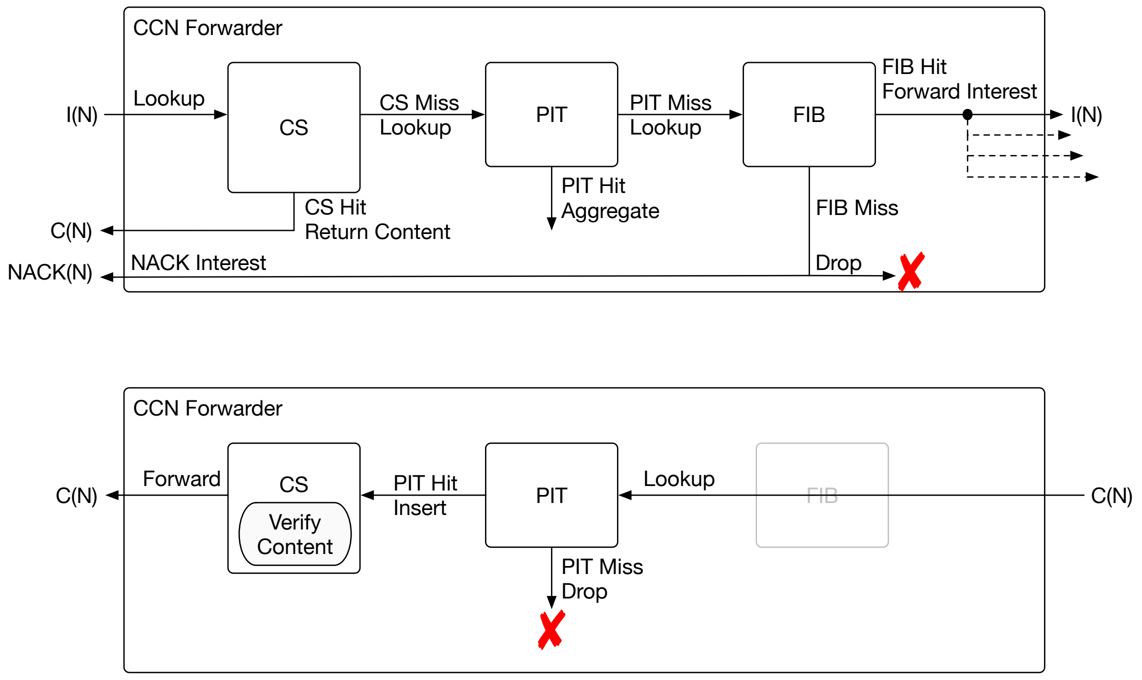


Figure 3.1: CCN interest and content forwarding logic

An interest matches a content object if their names match. Interests typically *refine* the possible matching content objects by carrying additional constraints, or restrictions. One restriction is `KeyIdRestriction`, or KeyId, which carries the hash digest of the content object public verification key. This restricts the content object to having been signed by the designated key. Another restriction is `ContentObjectHashRestriction`, or ContentId, which carries the hash of the desired content object. An interest with a ContentId can match at most one content object (with overwhelming probability), i.e., the one whose hash matches the ContentId. An interest with a ContentId is often said to have a *self-certifying name*, as first defined by Baugher et al. [48] and formalized by Ghali et al. [150], since the name and ContentId identify a unique content. Interests with restrictions match content if and only if the name and restrictions match. However, if an interest carries a ContentId it can match a content object *without a name*. We call such content objects *nameless*; their use is described below. These matching rules are expressed in the following predicate. Let N_i (N_o), K_i (K_o), and H_i (H_o) be the Name, KeyId, and ContentId in the subject interest (content).

$$\boxed{(\neg N_o \vee (N_i = N_o)) \wedge (\neg K_i \vee (K_i = K_o)) \wedge (\neg H_i \vee (H_i = H_o)) \wedge (\exists N_o \vee \exists H_i)}$$

3.3 Messages and Packet Formats

Structurally, names are sequence of typed *segments*.[2] There are a number of types such as T_NAMESEGMENT, T_VERSION, and T_CHUNK. A segment with type t and value v is written as t=v. For example, a segment with type T_NAMESEGMENT and value "chris" would be expressed as T_NAMESEGMENT=chris. A complete name for BCC's news homepage on May 20, 2017 might then be expressed as /bbc/news/05-20-2017/index.html. For clarity, we might also express this name, in wire format, as an S-expression as follows:

```
(T_NAME
        (T_NAMESEGMENT  3  "bbc")
        (T_NAMESEGMENT  3  "news")
        (T_NAMESEGMENT  4  "05 - 20 - 2017")
        (T_NAMESEGMENT 4  "index.html")
```

[2]CCN uses a Type, Length and Value (TLV) [259] format to encode all packet fields.

)

The default type of a segment is `T_NAMESEGMENT`. We omit this below for presentation clarity. Also, we refer to the i-th segment in name N as $N[i]$ or N_i, depending on context.

Each CCN packet begins with a fixed and optional header. The fixed header identifies the type of packet (message), the length of the fixed and optional header (in bytes), the total size of the packet, and some additional metadata. The optional header carries hop-by-hop fields, such as the message hash, interest TTL, or recommended cache time. See [225] for more details of these fields. The packet format reserves two bytes for the packet length, which places an upper bound of 64KB on the size of each packet. The body of the packet contains multiple top-level TLV containers. These top-level containers carry the message construct, e.g., the actual interest or content, as well as packet authentication information, e.g., the public verification key and digital signature.

Interests and content objects are the base for different types of messages. The interest message TLV contains a name TLV and optional payload TLV. The payload carries application data bound to the name. When an interest carries a payload, its name is modified by appending a unique "payload identifier" name segment, of type `T_PID`, to the end. This has the effect of diversifying the interest to prevent inadvertent cache hits. As previously discussed, interest messages may also carry a KeyId or ContentId. The content message TLV contains an optional name TLV and payload TLV. The name is optional if the corresponding interests carry a ContentId.[3] The payload carries the application data identified by the name. Content may also carry fields that indicate the type of payload and its expiration time, denoted `PayloadType` and `ExpiryTime`. Payload types include opaque application data, a cryptographic key, or a pointer (described below). The expiration time denotes the absolute time when the payload expires. Routers must not serve expired content.

[3]Producers will know whether or not this is the case and can omit the name as needed.

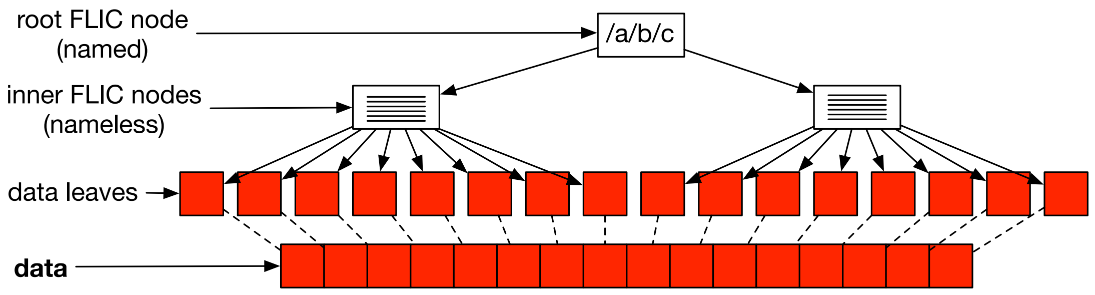


Figure 3.2: FLIC tree

Manifests are a special type of message used to convey ContentIds in bulk to consumers. The latter use these to create interests with a name and ContentId. FLICs are one type of Manifest structure [307]. A FLIC packet, or node, is part of a network-level collection of content objects. Each FLIC node in a collection contains a list of ContentIds that are used to construct interests for specific content objects. Each name and hash digest pair is called a *pointer*. (We often use the term *locator* to describe these names since the hash is the real identity of the content.) FLIC nodes may contain or encode pointers to normal content objects (data leaves) or other FLIC nodes, thereby creating a DAG structure with a single FLIC root and normal content objects as the leaves. Typically, all nodes except the root are nameless, as shown in Figure 3.2. Consumers resolve FLIC root nodes to data leaves by performing an in-order depth-first-search of the pointers.

3.4 Named Data Networking Differences

Although this dissertation focuses on CCN, it would not be complete without a discussion of Named Data Networking (NDN), which can be seen as CCN's academic dual. NDN started as a fork of the PARC CCNx project [10] in 2009. Since then, CCN and NDN have progressed in parallel towards similar goals. However, they differ in several key aspects, as described below.

Packet format: NDN interest and data packets are encoded using arbitrary-length TLVs. This inhibits fast-path packet processing.

Matching: NDN interest-to-data matching is not exact. An interest matches a data packet if the interest name is a prefix of the data name.

Discovery: As a consequence of LPM-based matching, NDN permits in-network data discovery. NDN consumers are forced to explicitly *exclude* names they do not wish to match in an interest. The network must not deliver data which matches one of these excluded names.

Data authenticity: Public key content digital signatures are mandatory in NDN.

Immutability: Name-to-data bindings in NDN are immutable, meaning that only one name can map to one data packet.

3.5 Security and Privacy Problems

CCN was designed in part to address several problems of IP-based networks. Content integrity, confidentiality, availability, access control, and privacy are all paramount to the success of CCN and other ICN architectures [121, 305]. While CCN addresses some of these fundamental problems, it comes with its own challenges. In this section, we survey a subset of open problems.

3.5.1 Content Poisoning

A content poisoning attack is when an attacker, such as a malicious router or producer, returns *fake* or *invalid* data in response to an interest. Fake or invalid content cannot be authenticated given the associated public key. Use of KeyId and ContentId restrictions is critical for protecting against content poisoning attacks. However, restrictions are only useful insofar as inputs to the verification process. If a router never caches content, e.g., if it does not have a cache and therefore no reason to verify content, or if its throughput is too high to perform public-key cryptographic operations at line speed, content may be forwarded without verification. This vulnerability allows so-called *on-path* attackers to hijack name prefixes and supply poisoned content in response. Identifying and avoiding on-path attackers remains an open problem.

3.5.2 Access Control

Unlike modern networks, CCN promotes data security instead of transport security. This means that sensitive or confidential data must be encrypted. Access control is the mechanism by which only authorized consumers are given access to certain content. This is an application-layer problem since it involves consumer authentication. However, the exact design may have many implications on the network, e.g., encrypting content with an ephemeral public key will render caches useless. To date, there have been many proposed techniques; see Chapter 4 for examples. Generally, each proposal differs in how private keys are distributed to consumers, how consumers obtain keying material necessary to decrypt per-content encryption keys, cryptographic algorithms used to protect keying material, and how revocation is handled (if at all). As of yet, there is no access control scheme that can accommodate all possible use cases.

3.5.3 Privacy

Privacy in CCN is simultaneously better and worse than privacy in IP-based networks. Since CCN interests do not carry a source address, consumers enjoy some measure of privacy or deniability. This improves consumer anonymity beyond that in IP-based networks [105, 309].

However, CCN forces names to be in the clear so that the routers can operate on them for forwarding and caching. This can be exploited to correlate requests across consumers [146]. If two consumers request the same content, they issue requests carrying the same name.

Named content can also be exploited to learn when nearby consumers ask for select content [23]. Specifically, an adversary can use caches as an oracle to determine if content is local. Given the RTT to the nearest cached copy of some suspect content as well as the RTT to the authoritative producer, the difference between these two values is close to zero if the content was not cached, and non-zero otherwise.

Content may also reveal information about its recipient consumer, through the name, or the producer, through the signature and associated verification information. In general, since CCN reveals more information to the network, privacy is difficult to define and achieve in comparison to IP.

3.5.4 Denial of Service

Interest aggregation and content caching provides limited protection against denial-of-service (DoS). However, this comes at a cost. Stateful forwarding via the PIT is a major DoS vector. It is trivial for an active and malicious adversary to saturate a PIT with fake or invalid interests. Since routers cannot distinguish real from fake interests, they must allocate space in their PIT to store pending interest information. Deterministically protecting CCN routers and network deployments against this type of attack is an open problem.

Another form of DoS is possible at the forwarding layer. Specifically, if routing updates are not authenticated, a malicious adversary may fake ownership of a namespace for which they do not control. Routers may also hijack a namespace and prevent interests from reaching a legitimate producer. Protecting against such attacks is an open problem in the absence of mandatory signature verification.

Chapter 4

Access Control

In this chapter we focus on access control and confidentiality in CCN. Fundamentally, access control is about restricting access to data to authorized consumers, or principals. (Where appropriate, we use the terms consumer and principal interchangeably in this chapter.) There are several dimensions of access control. One protects only sensitive data via encryption. We call this *content-based* access control (CBAC). CBAC schemes typically involve offline principal authentication and authorization, e.g., when obtaining private decryption keys.

Smetters et al. [287] defined the first group-centric CBAC scheme for CCN, based on legacy CCNx 0.8x [10]. This scheme grants principal groups, which may be composed hierarchically, access to parts of a content namespace tree. For example, a principal granted access to the `/foo` namespace would inherit access to the `/foo/bar` namespace. Hamdane et al. [164] proposed a similar scheme scheme based on hierarchical identity-based encryption (IBE) [65]. Content is encrypted under root namespace keys and becomes accessible to all nodes under that namespace. Write access is granted by encrypting prefix root keys for select principals. Zhang et al. [344] proposed another IBE-based access control scheme, wherein producers use consumer identities, rather than content names, to encrypt content. Misra et al. [219, 220] suggested another CBAC scheme based on broadcast encryption [66, 310, 119]. It uses Shamirs secret sharing scheme [281] to distribute content decryption keys. Each share is encrypted for a broadcast group to form an "enabling block." Consumers use their private keys to decrypt these enabling blocks and combine them to recover the content decryption key. Ion et al. [169] proposed using ciphertext-policy attribute-based encryption (CP-ABE) [57, 156] for content protection. Principals are associated with attributes, which are used to encrypt content. To locate authorized content, this scheme includes a routing technique based on principal attributes. This scheme, as others of the same type, assumes a private key generator for deriving and distributing principals' secret keys. Wang et al. [317] proposed a key-policy ABE (KP-ABE) scheme for access control. By definition, KP-ABE binds access control policies to keys, thus simplifying policy management as there are fewer keys to distribute. A hybrid attribute-based scheme was proposed by Li et al. [201, 200]. Content is encrypted under a random symmetric key, which is then encrypted under some attribute-based access policy. The encrypted symmetric key then serves as the name of the published content. A consumer obtains this name by requesting the name of the original data from the producer. Raykova et al. [261] described an access control system for ICEMAN built on multi-authority ABE [199]. It allows multiple, non-colluding authorities to issue private decryption keys. Their design uses standard signatures and ABE for content authenticity and confidentiality, respectively. Since ICEMAN is a publish-subscribe ICN, content is routed based on metadata tags associated with the data, selected by the publisher. These are matched against subscriber interests on a hop-by-hop basis. Attributes are hashed so as to not leak information about content. (However, this metadata is also a privacy leak, similar to a CCN name. We discuss this at length in Chapter 5.) At each hop, content is re-signed to ensure provenance. Thus, consumers must trust every node on the path to the publisher. In Section 4.1, we describe our variant of CBAC built on proxy-encryption (PRE).

Mangili et al. [211] proposed an access control scheme based on two layers of encryption. Content is first partitioned and then fragmented. Fragments are encrypted and then coupled with symmetric keys. Whole partitions are encrypted under a different key that is only made available to authorized principals. Principals obtain doubly encrypted fragments, decrypt and reassemble them into partitions, and then decrypt the whole partition to obtain the original content.

Designs discussed thus far differ in many respects, especially with respect to the underlying public-key encryption scheme. Mannes et al. [212] presented an experimental comparison of a subset of these designs. Despite performance differences, one commonality is the need to obtain decryption keys. They may be packaged alongside encrypted content or published separately as distinct content objects. For the latter case, consumers must discover the names of these keys. Yu et al. [338] proposed a design that uses name conventions and NDN longest-prefix-matching (LPM) rules for discovery, called name-based access control (NAC). Content are encrypted with (randomly generated and symmetric) content encryption keys, which are then encrypted under key-encrypt keys (KEKs). Key-decrypt keys (KDKs), used to invert KEK encryptions, are encrypted under consumer public keys and optionally bound, by name, to policy-specific time windows. Consumers interactively discover content encryption key and KDK names using NDN-specific LPM rules. Earlier, Kurihara et al. [195] proposed an alternate access control framework that uses a special type of CCN manifest to explicitly convey access control information, including key names, to principals. This generates more keys than NAC, though is arguably more flexible since producers can specify any type of access control scheme in these manifests. Moreover, it does not rely on in-network discovery of key names.

Centralized- or proxy-based access control designs represent another variety of content-based access control. Da Silva et al. [90] proposed using CP-ABE to encrypt content for authorized parties. Access control policies are encrypted and only visible to trusted proxies. Encrypted content objects are stored in routers. Proxies authenticate clients and decrypt policies on their behalf. Revocation is handled by the producer notifying the proxy of changes in access policies. Similarly, Hamdane et al. [163] used a distinct access control manager (ACM) that controls read and write access to the content namespace. Clients request encryption and decryption keys from the ACM to modify and read content, respectively. Content symmetric keys are encrypted with keys granted by the ACM. Access policies are delivered with content. Clients obtain decryption keys by authenticating themselves and presenting access policies to the ACM. Aiash et al. [29] constructed an identity-based based access control scheme for NetInf. This approach requires clients to register with publishers and for publishers to share their public keys with the NRS, which authenticates clients and provides "subscriber tokens" to be used when requesting content. Clients obtain content pointers, or locators, from the NRS. The publisher verifies subscriber tokens by validating them against the NRS before returning requested content to the subscriber.

Enforcing access control with online authorization checks can also be done alongside, or in conjunction with, content encryption. Chen et al. [80] proposed an access control mechanism with in-network assistance. Content is hybrid-encrypted for authorized consumers identified by their public key. Additionally, producers distribute Bloom Filters (BFs) encoding public keys of active consumers. Routers use BFs to filter requests from fake or inactive consumers before they reach the producer. As presented, the design is vulnerable to replay attacks and BF false positives. In Section 4.2, we propose a stronger interest-based access control (IBAC) design wherein interests can only be generated by authorized consumers. This makes the authorization check online and, with router participation, enforceable in the network.

Fotiou et. al. [123] proposed an access control mechanism similar to IBAC wherein access control enforcement is delegated to a separate, non-caching entity called an access control provider (ACP). ACPs maintain producer-supplied access control policies. Each content object has a pointer to a function that determines whether to serve the content to the requesting consumer. ACPs evaluate this function. Relaying parties store content objects and are oblivious to access control policies. Similarly, ACPs have no knowledge of consumers requesting content (for privacy purposes); they merely evaluate whether relaying parties should forward specific content.

More lightweight alternatives have been studied to complement existing content-based access control schemes using pubic-key encryption. Shang et al. [282] proposed an approach for access control in constrained environments. Symmetric-key cryptography is used for efficient device and command authentication. They present a key exchange and management protocol that derives shares keys. Access control policies are bound to shared keys and expressed in names. Zheng et al. [346] presented a session-based access control scheme for online social networks (OSNs) which revolves around separating publicly addressable names and secret names that only authorized consumers can request. The OSN maintains the mapping between these types of names and only provides secret (encrypted) names to authorized consumers. The mapping is changed at regular intervals to minimize correlation between data and secret names. Renault et al. [265, 266] presented a session-based transport protocol. Caching nodes require a “security controller” to authorize clients and perform key exchange (based on, e.g., Diffie-Hellman). The complete protocol includes challenge-response client authentication and authorization followed by encryption. Wang et al. [327] proposed a so-called access control system built on mutual authentication. Users register with a content hosting service, e.g., an OSN. Users interact with the system over sessions, each of which is associated with a unique key shared with the service. A session is started by logging in and authenticating oneself to the service. Uploading content requires consumers to encrypt content with their shared service key. The hosting service then decrypts content and, with a freshly generated key, re-encrypts it before storage. Consumers obtain content by requesting it over a session and receiving a unique name for encrypted content, decryption key, and some additional metadata, all of which are encrypted with a session key.

One critical facet of any access control system is revocation. By and large, existing designs do not offer a plausible way to deal with this problem since producers have no way to delete content from the network. The problem of *unsafe replicas* or stale content in CCN was first addressed by Angius et al. [38]. Analytical and experimental assessment confirmed that: *"...the more frequently content is requested the higher is the chance of one request ending up in between a revocation and the eviction [of the stale key]."* The proposed method relies on a monotonically decreasing cache lifetime enforced by cooperating routers. This does not allow a producer to change the lifetime after content is published; it only seeks to minimize the time window when stale or unsafe replicas can be accessed.

Mauri et al. [217] proposed a mechanism that implements content revocation without input from the consumer. The proposed approach uses the ccnx-sync protocol to perform OCSP-like synchronization of key data, i.e., determine content that has been revoked. This requires proactive behavior by each participating repository and is not distributed. Yu et al. [339] suggested using ChronoSync [348] to synchronize revoked key endorsements among group members. Revocation, however, is not the same as cache deletion. Revoked content, if still cached, can be inadvertently accessed by malicious or benign consumers.

Ishiyama et al. [170] discussed a new caching technique allowing routers to proactively share content with downstream peers which requested that content. The suggested multicast forwarding strategy increases the number of replicas in the network. However, unsolicited content objects can be seen as a form of attack similar to cache poisoning [150].

The concept of cost-aware caching in CCN was introduced in [27, 39, 316, 41, 298]. Various economic incentives for ISPs and ASes to cache content on behalf of producers have been explored. Cost-aware routers that cache based on popularity and economic incentives are studied in [40]. In general, the economic problem of supporting prioritized caching in the network is addressed without any attention to the inverse problem: how can content be removed from caches? In Section 4.3, we address this problem, which is also aimed at addressing access control revocation, with a scheme for best-effort autonomous content deletion in CCN.

4.1 Content-Based Access Control

In this section, we describe PRE-AC, a CBAC system for CCN based on proxy re-encryption (PRE) [62]. PRE-AC does not rely on trusted content replicas such as CDNs or traditional PKIs. It also uses an identity-based variant of PRE for intuitive derivation of public keys that are associated with users and their access rights. Our design has several benefits, including:

- Few keys to store for producers and consumers. Consumers need only store two private keys associated with their identity – one for identity-based encryption and one for proxy re-encryption).
- End-to-end content security without sacrificing network caches.
- Support for intermediary content re-encryption on behalf of the consumer.

In Section 4.1.1, we review two PRE schemes: one identity-based construction proposed by Green and Ateniese [157] and another based on a combination of ElGamal encryption and Schnorr's signature scheme proposed by Chow et al. [81]. PRE-AC is described at length in Section 4.1.2. Results in Section 4.1.3 indicate that the identity-based construction is significantly cheaper due to the complexity of modular arithmetic operations in the Chow et al. [81] variant. We describe our proof of concept implementation for CCNx 0.8 [10] in Section 4.1.4, the legacy open source implementation of CCN.

4.1.1 Proxy Re-Encryption Overview

Proxy Re-Encryption (PRE), first conceptualized by Blaze et al. [62], is a family of cryptographic schemes in which an untrusted proxy is allowed to transform a ciphertext encrypted under Alice's public key to one encrypted under Bob's public key, given an appropriate re-encryption key. PRE has many practical applications, including: message transfer in secure email systems, fine-grained access control in secure cloud storage, and, most relevant to this work, improved DRM technologies [157].

Formally, a PRE scheme is a tuple of six algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ defined below in general terms for *multi-hop* schemes. A *single-hop* scheme only permits a ciphertext to be re-encrypted at most once. We denote a ciphertext re-encrypted l times as c^{l+1}. Thus, $c^1 = c$, c^2 is the first re-encryption of c, and so on.

- $\mathsf{Setup}(1^k)$: This procedure takes a security parameter k, which determines the size of the underlying group in which all operations take place, and outputs the public set of parameters params. This procedure may also output a master secret key.
- $\mathsf{KeyGen}(\mathsf{params})$: Generate and output a public and private key pair. Identity-based schemes typically specify a particular identity and master key that are used in the creation of the private key.
- $\mathsf{Encrypt}(\mathsf{params}, pk, m)$: Encrypt plaintext $m \in \mathcal{M}$ using the input public key pk (or identifier) and output the resulting level 1 ciphertext c_i^1.
- $\mathsf{ReKeyGen}(\mathsf{params}, pk_i, pk_j)$: Generate and output a re-encryption key $rk_{i \to j}$ using the public parameters and public keys for users i and j.
- $\mathsf{ReEncrypt}(\mathsf{params}, rk_{i \to j}, c_i^n)$: Re-encrypt the level n ciphertext c_i^n, which is encrypted under the public key of user i, to a new level $n+1$ ciphertext c_j^{n+1} using the re-encryption key $rk_{i \to j}$ that may then be decrypted by the secret key of user j.
- $\mathsf{Decrypt}(\mathsf{params}, sk_j, c_j^n)$: Parse the level n ciphertext c_j^n to determine n, decrypt the ciphertext accordingly using the secret key sk_j, and output the original plaintext m.

Further distinctions between different PRE schemes can be made based on whether they are *unidirectional* or *bidirectional*. In a unidirectional scheme, a ciphertext originally produced by Alice and then transformed by the untrusted proxy for Bob cannot be transformed back to one for Alice. A bidirectional PRE scheme allows transformations to be performed in both directions. Non-interactivity of a PRE scheme is another important property. A PRE

Table 4.1: Comparison of Green et al. [157] and Chow et al. [81] PRE schemes.

PRE Property	*Identity-Based* [157]	*ElGamal-Schnorr* [81]
1. Unidirectional	✓	✓
2. Non-interactive	✓	✓
3. Multi-hop	✗	✗
4. Non-transitivity	✓	✓
5. Space-optimal	✓	✓

scheme is said to be non-interactive if re-encryption keys $rk_{i \to j}$ can be generated without input or collaboration with user j. Finally, a PRE scheme is space-optimal if it does not incur additional communication costs, e.g., ciphertext expansion upon re-encryption, in order to support re-encryption.

Following the breakthrough identity-based encryption scheme from Boneh and Franklin [66], Green and Ateniese [157] proposed a non-interactive, identity-based, single-hop PRE scheme. Built upon bilinear maps, the security of this particular scheme depends on the computational intractability of the Decisional Bilinear Diffie Hellman (DBDH) assumption. It is CCA-secure in the random oracle model. Many subsequent efforts built on this work, including, though not limited to: replicating the scheme without the use of pairings [81], proving security in the standard model [171], and adding conditions which limit when and how re-encryption keys may be used [203].

Among these options, we chose two PRE schemes based on their relative performance, flexibility, and security. We prioritized client-side re-encryption and decryption, as this is critically important for usable access control. Ultimately, we considered two different PRE constructions – one built on pairings and one built on ElGamal-type cryptosystems. The identity-based variant of PRE-AC uses the original PRE scheme from Green and Ateniese [157], which builds upon the construction in [45]. The second variant is based on the work of Chow et al. [81]. A comparison of the relevant properties for both schemes is shown in Table 4.1. In the following sections, we provide more details of each variant.

Identity-Based PRE Overview

The identity-based PRE scheme (IBP2) from Green and Ateniese [157] is built on a modified version of Hierarchical Identity-Based encryption from Gentry and Silverberg [138] and the underlying identity-based encryption scheme designed by Boneh and Franklin [66]. The public parameters returned from the Setup procedure consist of a group generator g (in the symmetric pairing case) and element g^{msk}, where msk is the master secret key, as well as a tuple of hash functions used in the remaining procedures. Since unique identities take the place of public keys in identity-based schemes, the KeyGen procedure requires the public parameters, msk, and a unique identity id to output the corresponding secret key, sk_{id}. Encryption requires only the target recipient's identity id, public parameters params, and the actual message m. The $\mathsf{ReKeyGen}$ procedure must be performed by the entity in possession of msk and requires the public parameters params, the source identity id_i, and the destination identity id_j. The output of this procedure is the re-encryption key $rk_{\mathsf{id}_i \to \mathsf{id}_j}$.

Re-encryption takes the public parameters params, level 1 ciphertext $c^1_{\mathsf{id}_i}$ encrypted for the identity id_i, and re-encryption key $rk_{\mathsf{id}_i \to \mathsf{id}_j}$ as input and outputs a transformed level 2 ciphertext $c^2_{\mathsf{id}_j}$ that is encrypted under the identity of user j. Decryption of this level 2 ciphertext simply requires the public parameters params, and secret key of the identity id_j, sk_{id_j}.

In this PRE scheme, which works for both symmetric pairinngs $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ and asymmetric pairings $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, is proven IND-PrID-CCA secure assuming the DBDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_T)$ (resp. $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$) under the random oracle model. The benefit of this scheme is that the proxy remains an untrusted party that can perform (public) validity checks before re-encrypting ciphertext blocks. This prevents a dishonest proxy from acting as an attacker's re-encryption oracle.

ElGamal-Schnorr PRE Overview

Chow et al. [81] presented one of the few constructions in the literature that does not rely on pairings. It combines a "hashed" CCA-secure ElGamal-type encryption scheme with the Schnorr signature scheme using a token-controlled approach to enable single-hop re-encryption. Much of the scheme is influenced by previous PRE proposals, e.g., [103, 283].

The public parameters consist of a generator g for the group $\mathbb{G} \subset \mathbb{Z}_q^*$, where the number of bits in q is equal to the security parameter κ, two parameters l_0 and l_1 which are polynomial in κ (for a message space that is l_0 bits), and a set of hash functions. The consumer key pairs consist of a pair of secret keys $sk_{i,1}$,$sk_{i,2}$ and public keys $pk_{i,1} = g^{sk_{i,1}}$,$pk_{i,2} = g^{sk_{i,2}}$. The ReKeyGen operation generates a conversion key $rk_{i \to j}$ used to mask the original ciphertext encrypted with the secret keys of entity i to one that may be decrypted with only the secret key $sk_{j,2}$ of entity j. The construction of level 1 and 2 ciphertexts are publicly verifiable and delegator-safe, which means that the proxy may generate level 2 ciphertexts after verifying input level 1 ciphertexts without revealing or relying upon the secret keys of either party involved in the transformation. Finally, the Decrypt procedure for a level 2 ciphertext for user j extracts the information necessary to unmask the original plaintext using their secret key $sk_{j,2}$ and outputs the message if the final verification step passes, or nothing otherwise.

4.1.2 PRE-AC System Design

In this section we describe the reference architecture for PRE-AC, illustrated in Figure 4.1. Although our presentation is framed in the context of the previously described identity-based PRE scheme, it is general enough to work with any suitable PRE construction. Moreover, it can be used by any CCN application.

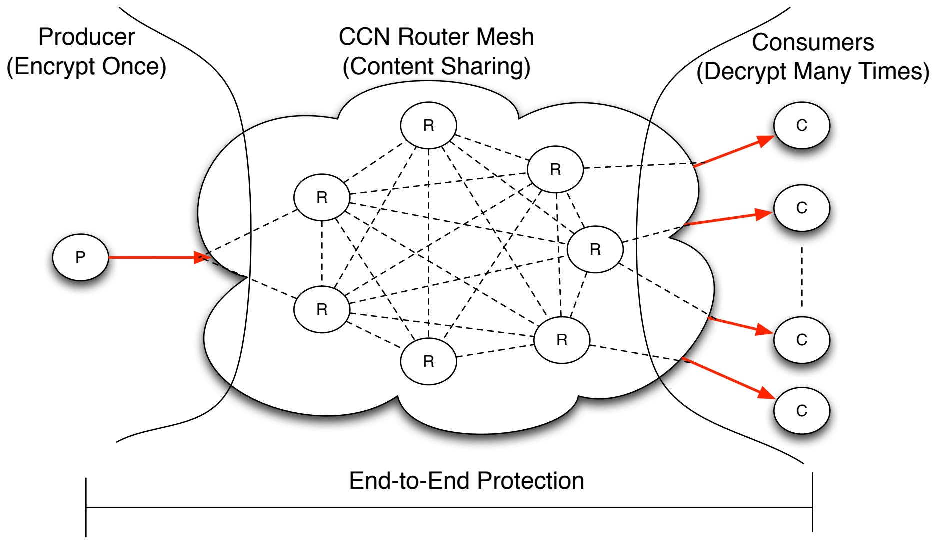


Figure 4.1: PRE-AC deployment

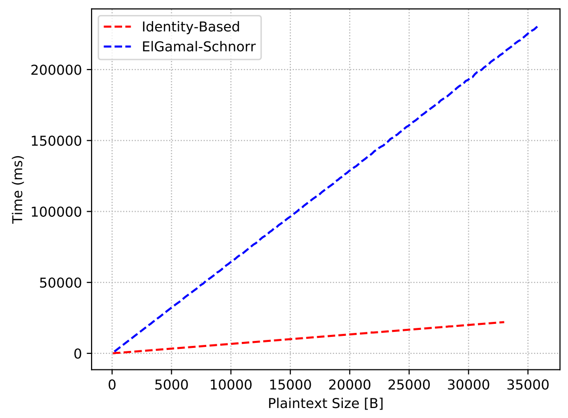
4.1.3 PRE-AC Protection

One naive way to use PRE for access control would be to individually protect all content with PRE using content names, i.e., encrypt $C(N)$ with N as the public identity. The producer would be the only entity that can generate and store the corresponding secret key for N. Thus, content remains secure as it moves through the network. This design consists of an offline and online phase. The offline phase configures the producer and each consumer with the appropriate cryptographic information, i.e., PRE public parameters and consumer secret keys. The online phase handles the distribution of content. For brevity, we omit the details of the setup phase as they can be implemented in a variety of ways, such as during software installation or device fabrication. When a consumer wishes to use the content they would need to request a corresponding re-encryption key from the producer. If privacy in terms of content-to-user linkability is a concern, the interest to request the key and and corresponding content could also be encrypted using the public keys of the producer and the consumer, respectively.

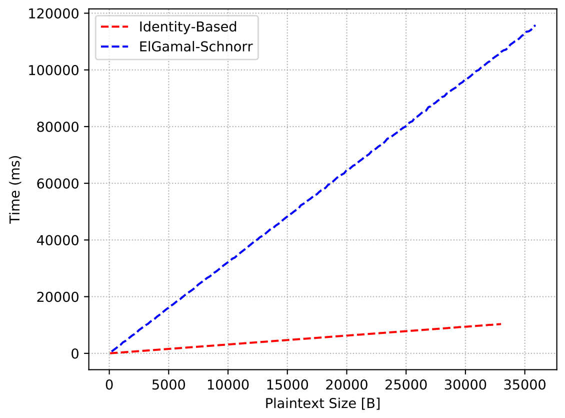
After receiving the re-encryption key, the consumer could re-encrypt the content. In the event the consumer is compromised and the original or transformed content are made publicly available, only the producer who owns the master secret key or the consumer who possesses the secret key for the transformed ciphertext may decrypt the content. Therefore, disclosure of encrypted content is safe.

To assess the feasibility of this fully PRE-based design, we implemented the PRE schemes discussed in Section 4.1.1. The identity-based scheme was implemented using the Java Pairing Based Cryptography library [97], and the ElGamal-Schnorr scheme was implemented using standard modular arithmetic operations provided by the Java `BigInteger` class. We instantiated each scheme with security parameters of similar strength: 256 bits for elliptic curve groups and 3072 bits for the Diffie Hellman groups, both of which are approximately providing 128 bits of security. We then measured Encrypt, ReKeyGen, ReEncrypt, and Decrypt procedure times (for level 2 ciphertexts only). Figure 4.2 shows our performance results (without pairing pre-computations) on a machine with an Intel(R) Core(TM) i5-3427U CPU operating at 1.80GHz with 8GB of main memory and using Ubuntu 14.04. The large security parameter needed to attain equivalent strength lead to worse performance than the identity-based scheme in our implementation. Despite the clear difference in performance, neither is practical for large digital media, such as movies or music albums that are MBs or GBs in size. This performance impediment led us to consider a hybrid encryption approach, wherein we use PRE to protect a symmetric key for efficient content encryption and decryption.

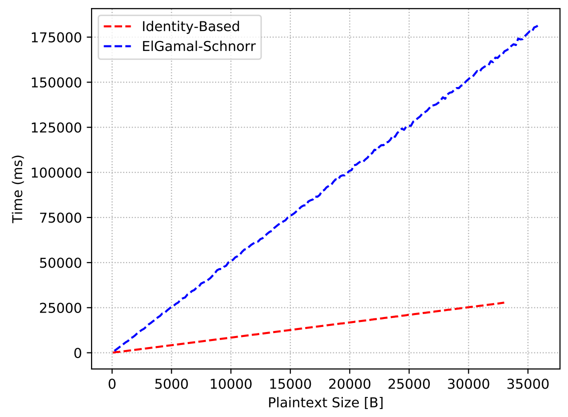
The flow of messages during the online phase are shown in Figure 4.3. This figure depicts the flow of traffic required to retrieve a piece of content $C(N)$ by two different consumers. The *params* argument is omitted from all PRE-related procedures for brevity. Also, we refer to a content tuple $(C(N)', k')$ as a piece of content encrypted with a symmetric key k and the encryption of k with PRE. As an optimization, consumers may fetch re-encryption keys in parallel with encrypted content, contrary to what is shown in Figure 4.3. This enables a



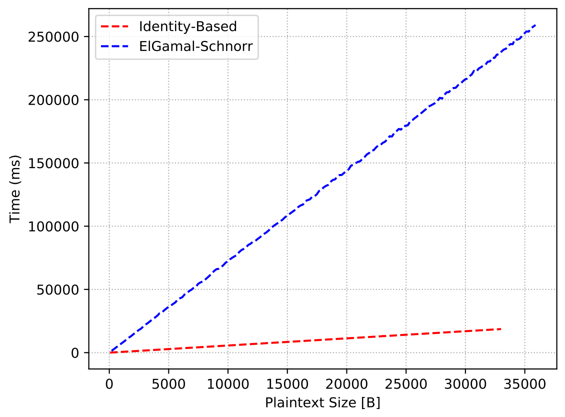
(a) Encrypt time comparison



(b) ReKeyGen time comparison



(c) ReEncrypt time comparison



(d) Decrypt time comparison

Figure 4.2: PRE-AC procedure performance

consumer start to decrypt chunks of the content without waiting to retrieve content in its entirety.

The details of the hybrid encryption procedures are captured in algorithms 1, 2, and 3. These list the explicit steps required for the Encrypt and ReKeyGen procedures in the producer and the ReEncrypt and Decrypt procedures in the consumer. $\mathsf{Enc}(\cdot,\cdot)$ and $\mathsf{Dec}(\cdot,\cdot)$ refer to symmetric-key encryption and decryption, respectively. Also, sk_A is the secret key for

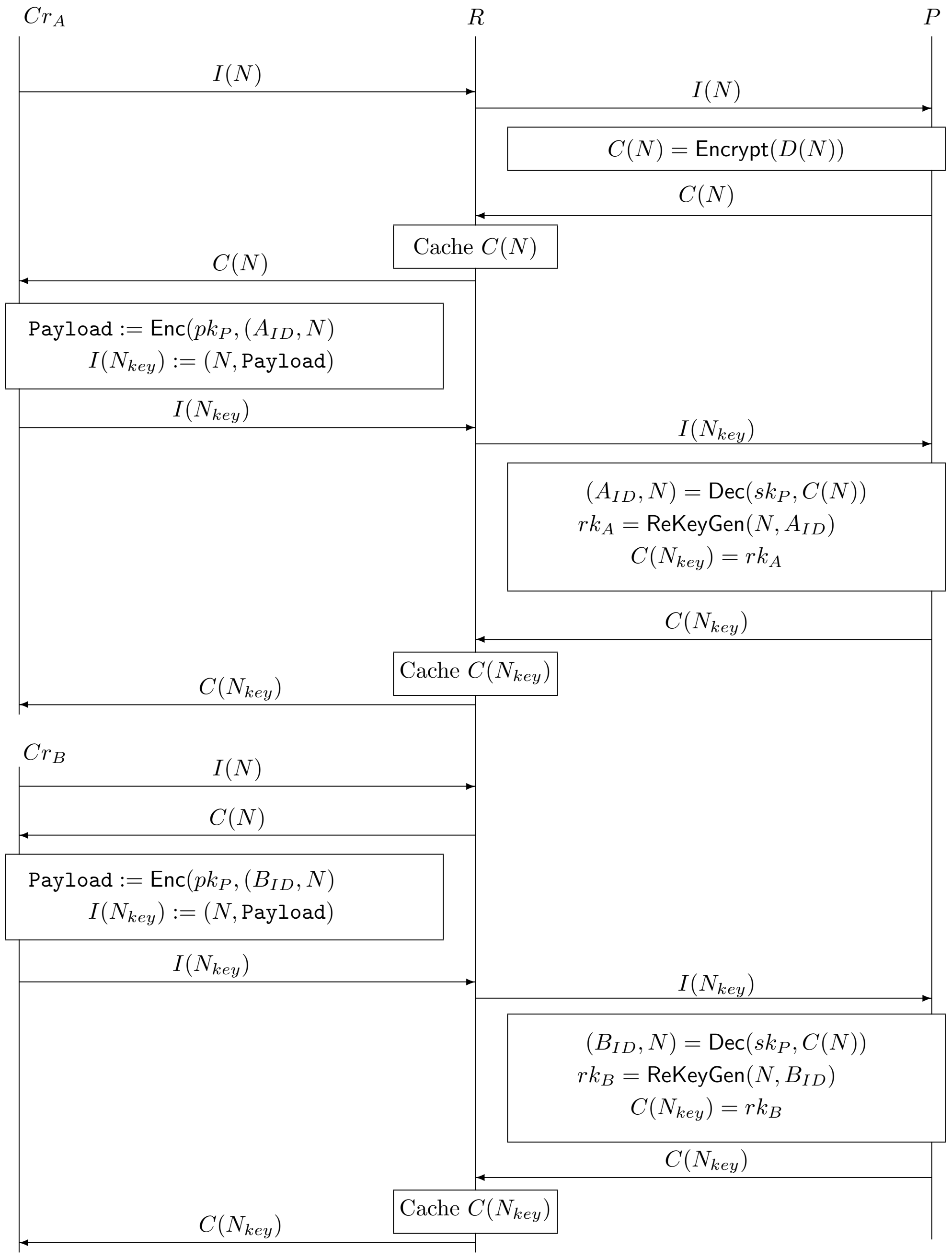

Cr_A
R
P
$I(N)$
$I(N)$
$C(N) = \mathsf{Encrypt}(D(N))$
$C(N)$
Cache $C(N)$
$C(N)$
$\mathtt{Payload} := \mathsf{Enc}(pk_P, (A_{ID}, N)$
$I(N_{key}) := (N, \mathtt{Payload})$
$I(N_{key})$
$I(N_{key})$
$(A_{ID}, N) = \mathsf{Dec}(sk_P, C(N))$
$rk_A = \mathsf{ReKeyGen}(N, A_{ID})$
$C(N_{key}) = rk_A$
$C(N_{key})$
Cache $C(N_{key})$
$C(N_{key})$
Cr_B
$I(N)$
$C(N)$
$\mathtt{Payload} := \mathsf{Enc}(pk_P, (B_{ID}, N)$
$I(N_{key}) := (N, \mathtt{Payload})$
$I(N_{key})$
$I(N_{key})$
$(B_{ID}, N) = \mathsf{Dec}(sk_P, C(N))$
$rk_B = \mathsf{ReKeyGen}(N, B_{ID})$
$C(N_{key}) = rk_B$
$C(N_{key})$
Cache $C(N_{key})$
$C(N_{key})$

Figure 4.3: PRE-AC control retrieval flow

consumer A generated from the (offline) KeyGen procedure. A_{ID} and B_{ID} are the identities of the two consumers.

Algorithm 1 PRE-AC Encrypt

```
Require: $N, \kappa, \mathsf{params}$
Ensure: $C(N)'$, $k'$
1: $k \xleftarrow{\$} \{0,1\}^{\kappa}$
2: $C(N)' \leftarrow \mathsf{Enc}(k, C(N))$
3: $k' \leftarrow \mathsf{Encrypt}(\mathsf{params}, N, k)$
4: return $(C(N)', k')$
```

Algorithm 2 PRE-AC ReKeyGen

```
Require: $N, \mathsf{params}, A$
Ensure: Re-encryption key from $N$ to $A$
1: $sk_N \leftarrow \mathsf{KeyGen}(N, \mathsf{params})$
2: $rk_{N \to A} \leftarrow \mathsf{ReKeyGen}(\mathsf{params}, sk_N, N, A)$
3: return $(rk_{N \to A})$
```

Algorithm 3 PRE-AC ContentDecrypt

```
Require: $\mathsf{params}, sk_A, rk_{N \to A}, (C(N)', k')$
Ensure: Content $C(N)$
1: $k'' \leftarrow \mathsf{ReEncrypt}(\mathsf{params}, rk_{N \to A}, k')$
2: $k \leftarrow \mathsf{Decrypt}(\mathsf{params}, sk_A, k'')$
3: $C(N) \leftarrow \mathsf{Dec}(k, C(N)')$
4: return $C(N)$
```

As an added note, PRE-AC assumes the producer is always available to generate a re-encryption key. However, as argued in [219], this may not always be true. When the producer is offline, the re-encryption key cannot be obtained. One way around this problem would be to introduce trusted intermediary nodes, e.g., content repositories, between the producer and consumers that proactively generate re-encryption keys on behalf of consumers. In the event that the producer falls offline, the consumer can still obtain their content re-encryption key from one of these intermediary nodes. This requires consumer identities to be sent in the clear (instead of encrypted). Thus, this particular enhancement may harm consumer privacy.

4.1.4 Prototype Implementation

To assess the correctness and implementation efficiency of the proposed architecture, we developed and tested a prototype implementation of PRE-AC as presented in Section 4.1.3. We used the legacy Java CCNx framework [10] which, as of this writing, is now deprecated.

In the implementation, there is a single content producer and multiple consumer processes that are spawned with a running instance of `ccnd`, the daemon process that implements the CCN protocol at the network layer in the CCN stack. Prior to initializing these processes, the public parameters used in the PRE scheme are generated and saved to persistent storage, i.e., a file on disk. When the producer process is started, it first reads the public parameters and initializes the rest of the internal components needed for PRE functions, registers the `/p/` prefix, and waits for incoming interests from consumers. Consumer processes also utilize the same public parameters to initialize the PRE functions.

The remaining five steps in the identity-based PRE scheme – KeyGen, $\mathsf{Encrypt}$, $\mathsf{ReKeyGen}$, $\mathsf{ReEncrypt}$, and $\mathsf{Decrypt}$ – are performed online between the consumer and the producer processes. Each consumer process requests a single piece of content from the producer, writes the content to disk, and then terminates upon completion. To accomplish this simple task, the consumer and producer complete the set of interactions below.

1. The consumer process requests its secret key in the PRE scheme corresponding to their identity, which is stored internally as a byte array, by encoding the byte array into a Base64 string A_{ID} that is CCN URL-friendly. The consumer builds and issues an interest with the name `/p/sk/`A_{ID}. Upon receipt, the producer decodes the consumer identity and then runs KeyGen with the encoded identity to construct a secret key sk_A. Finally, the producer encrypts the raw bytes of sk_A using the identity of the consumer and then sends the resulting ciphertext downstream.

2. The consumer encodes the target content name N as a Base64 string N and then builds and issues the interest `/p/media/`N. Upon receipt, the producer decodes the content name N as the last name segment and performs the following check. If a file with the name N exists, the contents of the file are read and encrypted in chunks using the PRE $\mathsf{Encrypt}$ function and then returned downstream.[1]

3. The consumer creates a tuple (A_{ID}, N), serializes it to a byte array, encrypts the byte array using the identity of the producer, encodes the corresponding ciphertext into a Base64 string C, and finally, builds and issues the interest `/p/rkey/`C. The producer then decodes C from the interest, decrypts the ciphertext using their secret key, and then deserializes (A_{ID}, N). The producer then runs $\mathsf{ReKeyGen}$ to generate $rk_{N \to A}$, encrypts the raw bytes of the key using the identity of the consumer, and returns the result to the consumer.

4. The consumer finishes the transaction by using the re-encryption key to re-encrypt the encrypted piece of content using the PRE $\mathsf{ReEncrypt}$ procedure, and then decrypts the content using their secret key sk_A.

Note that we omitted consumer authentication during for the first step as it will happen only once when a user first registers to the system. Authentication in later steps are unneeded as only the intended consumer can decrypt the content with its private key.

4.2 Interest-Based Access Control

In this section, we propose Interest-Based Access Control (IBAC), an alternative technique for implementing access control in CCN [141]. It is based on interest *name obfuscation* and

[1]If the name N does not match the name of a file, then the producer treats the bytes of N as an unsigned integer n and generates n random bytes to be encrypted and returned to the consumer to generate synthetic traffic.

authorized disclosure. Name obfuscation hides the *target* of an interest from eavesdroppers. As mentioned by Jacobson et al. [172], name obfuscation has no impact on the network since routers use only the binary representation of a name when indexing into PIT, CS, and FIB. As long as producers generate content objects with matching names, the network can seamlessly route interests and content objects with obfuscated names. However, interests with obfuscated names *must contain routable prefixes* so that the interests can still be forwarded from consumers to the producers. In other words, only a subset of name segments, e.g., the suffix of the name, can be obfuscated.

Another goal of name obfuscation is to *prevent unauthorized users from creating interests for protected content*. In other words, if a particular consumer Cr is not permitted to access content with name N, Cr should not be able to generate $N' = f(N)$, where $f(\cdot)$ is some obfuscation function that maps N to an obfuscated name N'. Possible obfuscation functions include keyed cryptographic hash functions and encryption algorithms. We explore both possibilities in this section.

Authorized disclosure is the second element of IBAC. This property implies that any entity serving content must authorize any interest for said content before it is served. This is necessary for IBAC since interests may be easily replayed by an adversary. In this context, authorization is necessarily coupled with authentication so that the entity serving the content can determine the identity of the requesting consumer. Therefore, consumers must provide sufficient authentication information, e.g., via an interest signature. Thus, to implement authorized disclosure (in the presence of router caches), any entity serving content must (a) possess the information necessary to perform authentication and authorization checks and (b) actually verify the provided authentication information. This issue is discussed at length in Section 4.2.3. Observe that disabling all content caching defers authorized disclosure checks to producers. In this case, all interests will be forwarded to producers that posses the information needed to perform these checks. However, by itself, prohibiting content from being cached is *not* a form of access control and reduces the efficacy of content retrieval.

IBAC may be used alongside CBAC to conceal both the name and the payload of content.[2] However, using IBAC in isolation is advantageous in scenarios where content payloads may need to be modified by an intermediary service, e.g., a media encoding application or proxy. In this case, content encryption prevents such modifications by services or applications besides the producer.

The main contributions of IBAC are:

- Architectural modifications to support IBAC without diminishing caching benefits.
- A mutual trust scheme wherein routers can verify whether consumers are authorized to access cached content.
- A security analysis of IBAC.
- Evaluation of router performance overhead when serving content via IBAC compared to publicly accessible content.

4.2.1 Threat Model

Let $\mathbb{U}(N)$ denote the set of authorized consumers for content with name N generated and controlled by producer P, and let $\bar{\mathbb{U}}(N)$ be the set of all unauthorized consumers. Let $\mathsf{Path}(Cr, P)$ be the set of all routers on the path between the consumer $Cr \in \mathbb{U}(N)$ and P. We first assume the existence of an adversary $\mathcal{A}$ who can deploy and compromise any router $R \notin \mathsf{Path}(Cr, P)$.[3] To keep this model realistic, we assume that the time to mount such

[2]As we will discuss, IBAC does not replace CBAC. It is a complementary form of access control.

[3]Any one of these actions can be performed adaptively, i.e., in response to status updates or based on observations.

an attack is non-negligible, i.e., longer than the average RTT for a single interest-content exchange. Table 4.2 summarizes the notation used in the rest of this section.

Formally, we define $\mathcal{A}$ as a 3-tuple: $(\mathcal{P}_{\mathcal{A}} \setminus \{P\}, \mathcal{C}_{\mathcal{A}} \setminus \mathbb{U}(N), \mathcal{R}_{\mathcal{A}} \setminus \mathsf{Path}(Cr, P))$ where the components denote the set of compromised producers, consumers, and routers, respectively. If $\mathcal{A}$ controls a producer or a consumer then it is assumed to have complete and adaptive control over how they behave in an application session. Moreover, $\mathcal{A}$ can control all of the timing, format, and actual information of each content through compromised nodes and links.

Let Guess denote the event where $\mathcal{A}$ correctly recovers the obfuscated form of a content name. Let Bypass denote the event where $\mathcal{A}$ successfully bypasses the authorization check for a protected content object. We define the security of an IBAC scheme with respect to these two events as follows.

Definition 4.1. *An IBAC scheme is* secure, but subject to replay attacks, *if* $\Pr[\mathsf{Guess}] \leq \epsilon(\kappa)$ *for any negligible function* ϵ *and a security parameter* κ*.*

Definition 4.2. *An IBAC scheme is* secure in the presence of replay attacks, *if* $\Pr[\mathsf{Guess} + \mathsf{Bypass}] \leq \epsilon(\kappa)$ *for any negligible function* ϵ *and a security parameter* κ*.*

Replay attacks are artifacts of the environment when a CCN access control scheme is deployed. In other words, in networks where links are insecure, passive eavesdroppers can observe previously issued interests and replay them. Consequently, these attacks are considered orthogonal to the security of the underlying obfuscation scheme used for access control. The authorized disclosure element of IBAC is intended to prevent such replay attacks.

To justify our adversarial limitation to off-path routers, consider the following scenario. If $\mathcal{A}$ can compromise a router $R \in \mathsf{Path}(Cr, P)$, then $\mathcal{A}$ can observe *all* content that flows along this path. Therefore, we claim that on-path adversaries motivate CBAC. Moreover, we exclude adversaries capable of capturing interests and replaying them in other parts of the network – see Section 4.2.3 for details.

Table 4.2: Relevant IBAC notation.

Notation	**Description**
$\mathcal{A}$	Adversary
Cr	Consumer
P	Producer
prefix	Producer prefix
N	Content name in cleartext
N'	Obfuscated content name
$I(N)$	Interest with name N
$C(N)$	Content object with name N
$\mathsf{ID}(\cdot,\cdot)$	Key identifier function
$f(\cdot)$	Obfuscation function
$\mathsf{enc}(\cdot,\cdot), \mathsf{dec}(\cdot,\cdot)$	Symmetric-key encryption and decryption function
$\mathsf{Enc}(\cdot,\cdot), \mathsf{Dec}(\cdot,\cdot)$	Public-key encryption and decryption function
$\mathsf{H}(\cdot)$	Cryptographic hash function
$\mathbb{U}(N)$	Set of authorized consumers
$\mathbb{G}_i$	Access control group i
$k_{\mathbb{G}_i}$	Obfuscation key of group $\mathbb{G}_i$
$pk^s_{\mathbb{G}_i}, sk^s_{\mathbb{G}_i}$	Public and private signing key pair associated with group $\mathbb{G}_i$
κ	Global security parameter
$\mathbb{C}$	Set of all content objects
r, t	nonce and timestamp
$\mathbb{B}$	Nonce hash table

4.2.2 Name Obfuscation Variants

Recall that the intuition behind IBAC is that if consumers are not allowed to access certain content, they should not be able to issue a "correct" interest for it. Specifically, only a consumer $Cr \in \mathbb{U}(N)$ should be able to derive the obfuscated name N' of an interest requesting content with name N provided by producer P. In this section, we discuss two types of name obfuscation functions: (1) encryption and (2) hash functions.

Encryption-Based Name Obfuscation

Let $\mathsf{Enc}(k, N)$ be a *deterministic* encryption function which takes as input a key $k \in \{0,1\}^\kappa$ and an arbitrary long non-empty binary name string N, and generates an encrypted name N'. Let $\mathsf{Dec}(k, N')$ be the respective decryption function. With encryption, the goal is for authorized clients to encrypt *segments* of a name so that the producer can perform decryption to identify and return the appropriate content object.[4] Obfuscation is based on knowledge of the encryption key and the content name under IBAC protection. In other words, even if an adversary knows N, it cannot generate N' since it does not possess the appropriate key.

To illustrate encryption-based obfuscation, assume Cr uses k to generate N' as $N' = \mathsf{Enc}(k, N)$. P then recovers N as $N = \mathsf{Dec}(k, N')$ to identify the content object in question and returns it with the *matching name* N' (not N). We prove the safety of this scheme below.

Theorem 4.1. *The IBAC scheme without authorized disclosure is secure,* but subject to replay attacks*, against $\mathcal{A}$ if an indistinguishably-secure (IND-secure) deterministic encryption algorithm is used for name obfuscation.*

Note: IND security is typically identical to CPA security in the public-key setting since the adversary is assumed to have access to the public key [181]. In this case, neither the encryption nor decryption key is known to $\mathcal{A}$.

Proof. Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-secure (deterministic) encryption scheme consisting of three probabilistic polynomial time algorithms Gen, Enc, and Dec for key generation, encryption, and decryption, respectively. Let k_e and k_d be the encryption and decryption keys produced by Gen. For any interest name N, it holds that $\mathsf{Dec}(k_d, \mathsf{Enc}(k_e, N)) = N$. Let $\mathcal{A}$ be any probabilistic polynomial adversary. The definition of the eavesdropping in-

[4] Recall that a cleartext name prefix is needed to route the interest to the intended producer.

distinguishability experiment, adapted for plaintext interest messages, denoted $\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}$, is as follows:

1. $\mathcal{A}$ is given input 1^κ and outputs a pair of interest names N_0 and N_1, and k_e and k_d are computed by running $\mathsf{Gen}(1^\kappa)$.
2. A single bit $b \leftarrow \{0, 1\}$ is chosen uniformly at random. The challenger computes the ciphertext $c \leftarrow \mathsf{Enc}(k_e, N_b)$, which is given to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a single bit b'.
4. The output of the experiment is said to be 1 if $b' = b$ and 0 otherwise.

Let $\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\kappa, b)$ be the same experiment run but where bit b is given as an input value. By the definition of IND-security, it follows that

$$|\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\kappa, 1) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\kappa, 0) = 1]| \leq \epsilon(\kappa),$$

for some negligible function ϵ. Recall that Guess is the event that $\mathcal{A}$ correctly guesses the obfuscated version a content name. The probability of $\mathcal{A}$ decrypting a message is at least $\Pr[\mathsf{Guess}]$. Therefore, the event when $\mathcal{A}$ successfully guesses the obfuscated version of the name, is when $\mathcal{A}$ outputs $b' = 1$ when $b = 1$ and $b' = 0$ when $b = 0$. Thus,

$$\Pr[\mathsf{Guess}] = |\Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\kappa, 1) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\kappa, 0) = 1]| \leq \epsilon(\kappa)$$

This concludes the proof. □

Supporting Multiple Access Groups: Thus far, we assumed that name encryption (obfuscation) keys are known to all authorized consumers in $\mathbb{U}(N)$. However, this might not be the case in practice. P might provide content under IBAC to several access groups each with different privileges.[5] Specifically, consumers in groups $\mathbb{G}_i(N) \subset \mathbb{U}(N)$, for $i = 1, 2, \ldots,$ might be allowed access to different resources. Therefore, several obfuscation keys, one for each group, should be utilized. For notation simplicity, we refer to $\mathbb{G}_i(N)$ as $\mathbb{G}_i$. Note that in an extreme scenario, each group would only contain a single consumer, i.e., each individual consumer has a unique key used to access the content in question.

To decrypt the obfuscated name N', P must identify the obfuscation key used to generate N'. This can be achieved if such consumers specify an identifier for the key used in the interest. Such an identifier could simply be the digest of the obfuscation key, i.e., $\mathsf{ID}_{\mathbb{G}_i} = \mathsf{H}(k_{\mathbb{G}_i})$, where $k_{\mathbb{G}_i}$ is $\mathbb{G}_i$'s encryption key. $\mathsf{ID}_{\mathbb{G}_i}$ can then be included in the interest payload.

Recall that CCN interest messages, by design, do not carry any source information, which provides some degree of anonymity. However, including $\mathsf{ID}_{\mathbb{G}_i}$ enables interest linkability by eavesdroppers (malicious or not). In other words, $\mathsf{ID}_{\mathbb{G}_i}$ can reveal the access group identities to which consumers belong, but not the identities of the consumers themselves. If this linkability is an issue for applications, $\mathsf{H}(k_{\mathbb{G}_i})$ can be encrypted using P's public key pk^P in the form $\mathsf{ID}_{\mathbb{G}_i} = \mathsf{Enc}(pk^P, \mathsf{H}(k_{\mathbb{G}_i}))$.[6] Note that for two identifier values of the same group, i.e., with the same k, to be indistinguishable, $\mathsf{Enc}(\cdot, \cdot)$ must be secure against chosen plaintext attacks [181].

Hash-Based Name Obfuscation

Let $\mathsf{H}(k, N)$ be a keyed cryptographic hash function. The obfuscated name N' can be generated as $N' = \mathsf{H}(k, N)$ for some key $k \in \{0, 1\}^\kappa$. Since hash functions are one-way, producers must maintain a hash table that maps obfuscated names to the original content

[5]We assume that each content object is only accessible by a single access group. However, this assumption will be relaxed later.

[6]Since a consumer cannot be expected to know the router from which content will be served, it is not plausible for them to encrypt these IDs with the public key of a (set of) router(s).

name, i.e., $\mathsf{M} : N' = H(k, N) \rightarrow N$ for all deployed keys.[7] In the worst case, the size of this hash table is $\mathcal{O}(|\mathbb{K}| \times |\mathbb{C}|)$, where $\mathbb{K}$ is the set of all keys and $\mathbb{C}$ is set of all content objects generated or published by P under IBAC protection. This approach provides the same security properties of encryption-based name obfuscation. Moreover, for long names with suffixes longer than the hash function output size, this approach reduces obfuscated interest size. However, it incurs additional computation overhead and storage at the producer. Thus, while keyed hash functions are viable for name obfuscation, deterministic encryption is a better approach.

4.2.3 Security Considerations

In this section we discuss the security of IBAC with respect to the adversary model described in Section 4.2.1.

Replay Attacks

Regardless of the obfuscation function, both IBAC schemes are susceptible to replay attacks. This is because both are deterministic. Therefore, an eavesdropper $\mathcal{A} \in \bar{\mathbb{U}}(N)$ could issue an interest with a captured N' and receive the corresponding content under IBAC protection from either the producer or a router cache. In other words, the same "feature" that makes it possible for authorized consumers to fetch IBAC-protected content from router caches also makes it susceptible to replay attacks.

Replay attacks are problematic in many access control systems. Standard countermeasures include the use of random, per-message nonces or timestamps. Nonces help ensure that each message is unique, whereas timestamps protect against interests being replayed at later points

[7]Producers do not have to keep hash tables for all *possible* keys of size κ, only tables of keys used by producers and issued to access groups.

in time. Thus, to mitigate replay attacks, we use both nonces and timestamps. In particular, each consumer $Cr \in \mathbb{U}(N)$ must issue an interest with (1) name N', (2) a randomly generated nonce r, and (3) a fresh timestamp t. The reason why we use both nonces and timestamps is to allow for loosely synchronized clocks and unpredictable network latencies. Note that if (1) clocks of consumers, producers, and involved routers in IBAC can be perfectly synchronized, and (2) network latencies can be accurately predicted, only timestamps are sufficient for replay detection. Moreover, since nonces and timestamps serve a purpose which is orthogonal to content identification and message routing, they are included in the interest payload.

Consumer nonces are random κ-bit values. If a router receives a duplicate nonce, it can drop the corresponding interest. Let w be a time window associated with authorized content.[8] To determine if a duplicate nonce was received, producers (or caches) must maintain a collection of nonces for each such content to prevent replay attacks. Timestamps themselves are not stored, they are only used to determine if the received interest is issued within the acceptable time window w. Once this time window elapses, all stored nonces are erased and corresponding content is flushed from the cache.

Although using nonces and timestamps allows detection of replayed interests, $\mathcal{A}$ capturing interests can still use their obfuscated names N' to fabricate another interest with legitimate r and t values. Therefore, we also stipulate that r and t should be authenticated via a digital signature, σ, which should be included in the interest payload. In order to bind r and t to their corresponding interest, N' is included in the signature computation. Signature generation and verification is performed using public and private key pairs associated with each access group $\mathbb{G}_i$. Collectively, interest payloads take the following form:

$$\left(\mathsf{ID}_{\mathbb{G}_i}, r, t, \sigma = \mathsf{Sign}_{sk^s_{\mathbb{G}_i}}\left(N'||\mathsf{ID}_{\mathbb{G}_i}||r||t\right)\right)$$

[8] Determining the proper value of w is outside the scope of this work. However, a logical approach is for routers to use the lifetime of authorized content as w.

where $\mathsf{ID}_{\mathbb{G}_i}$ is the identity of group $\mathbb{G}_i$, and $sk^s_{\mathbb{G}_i}$ is a signing key distributed to all consumers in $\mathbb{G}_i$. To verify σ, the matching public key $pk^s_{\mathbb{G}_i}$ is needed. For the remainder of this section, we use the term *authorization information* to refer to all information included in interest payloads for the purpose of supporting IBAC.

One alternative to digital signatures are keyed hashes or Message Authentication Code functions, e.g., HMAC [193]. In this case, consumers and routers would need to share the key used in the HMAC computation. This means that either consumers or producers need to distribute keys to all involved routers. This is problematic for two reasons: (1) compromising routers exposes shared keys, and, more importantly, (2) Cr must securely share pairwise keys with all routers on $\mathsf{Path}(Cr, P)$. Regardless of the distribution method, this incurs extra overhead and complexity compared to simply including, in cleartext, signature verification (public) keys in content objects.

Finally, consider the following scenario where two routers R_1 and R_2 cache content object $C(N')$ which is under IBAC protection. Assume that consumer Cr requests $C(N')$ by sending an interest $I(N')$ with valid authorization information that includes r and t. Assume that $I(N')$ is satisfied from R_1's cache. At the same time, $\mathcal{A}$, an eavesdropper between Cr and R_1, records $I(N')$. $\mathcal{A}$ can replay $I(N')$ to R_2 and receive $C(N')$ from the cache since routers do not synchronize stored nonces. Therefore, there is no way for R_2 to know that r and t were already used at R_1. One way to address this problem is for routers to share nonce lists for each content under IBAC they serve from cache. For this method to be effective, such nonces lists need to be securely shared with every single router in the network. This may be infeasible in large networks such as the Internet. Another approach is to have more accurately synchronized clocks, thereby allowing a smaller replay time window.

Authorized Content-Key Binding Rule

Although the aforementioned method for generating authorization information mitigates replay attacks, it also raises several questions. First, how does a router efficiently verify the signature in interest payloads? Second, if a router can *obtain* the key(s) necessary to verify this signature, how does it determine if such key(s) can be trusted?

To address these questions we propose a mutual trust framework for authorized disclosure. Ghali et al. [150] first studied trust in NDN, and ICNs in general, as a means of preventing content poisoning attacks [142, 137]. Even if routers can verify content signatures before replying from their cache, it does not mean that said content is actually *authentic*. Ghali et al. observed that this verification process requires trust in public (verification) keys that is only known to applications. Consequently, all interests must supply either (1) an identifier of the public verification key (KeyId), or (2) unique content identifier (ContentId). In effect, interests reflect the trust context of the issuing consumer in a form enforceable at the network layer. This framework can be viewed as one-way trust of content by routers. We extend this framework to allow producers to distribute information about authorized consumers, which can also be enforceable at the network layer. This allows routers to make trust decisions about individual interests.

Recall that in order for routers to verify which interests are authorized to access cached content protected under IBAC, the signature must be verified. To achieve this, producers should include the appropriate verification key with each IBAC-protected content object. To better understand this, assume the following scenario. Consumer $Cr \in \mathbb{G}_i$, for $\mathbb{G}_i \subset \mathbb{U}(N)$, requests content with name N by issuing an interest with obfuscated name N', and $\mathsf{ID}_{\mathbb{G}_i}$, r, t and σ as described in Section 4.2.3. Assume that the matching content is not cached anywhere in the network. Once this interest reaches the producer P, the latter verifies σ and replies with the content that also includes key $pk^s_{\mathbb{G}_i}$.[9] Router R will then cache $pk^s_{\mathbb{G}_i}$ along with the content itself. Once another interest for N' is received, R uses the cached $pk^s_{\mathbb{G}_i}$ to verify σ and returns the corresponding cached content.

We capture this with the following policy, called the Authorized Content-Key Binding (ACKB) rule:

ACKB: Cached content protected under IBAC must contain the verification key associated with the authorization policy.

Algorithm 4 Interest generation

```
1: INPUT: routable_prefix, $N$, $k_{\mathbb{G}_i}$, $pk^s_{\mathbb{G}_i}$, $sk^s_{\mathbb{G}_i}$, $\kappa$
2: $\mathsf{ID}_{\mathbb{G}_i} \leftarrow \mathsf{H}(k_{\mathbb{G}_i})$
3: $N' \leftarrow /\mathsf{routable\_prefix}/f(k_{\mathbb{G}_i}, \mathsf{Suffix}(N, \mathsf{routable\_prefix}))$
4: $r \overset{\$}{\leftarrow} \{0,1\}^\kappa$
5: $t \leftarrow \mathsf{CurrentTime}()$
6: $\sigma \leftarrow \mathsf{Sign}_{sk^s_{\mathbb{G}_i}}(N'||\mathsf{ID}_{\mathbb{G}_i}||r||t)$
7: $\mathtt{Payload} := (\mathsf{ID}_{\mathbb{G}_i}, r, t, \sigma)$
8: return $I(N') := (N', \mathtt{Payload})$
```

Algorithm 5 Content generation

```
1: INPUT: $I(N') := (\mathsf{routable\_prefix}, N', \mathtt{Payload})$
2: $(\mathsf{ID}_{\mathbb{G}_i}, r, t, \sigma) := \mathtt{Payload}$
3: $pk^s_{\mathbb{G}_i} \leftarrow \mathsf{LoopupVerificationKeyForID}(\mathsf{ID}_{\mathbb{G}_i})$
4: if $\mathsf{Verify}_{pk^s_{\mathbb{G}_i}}(\sigma)$ then
5:     $k^e_{\mathbb{G}_i} \leftarrow \mathsf{LookupDecryptionKeyForID}(\mathsf{ID}_{\mathbb{G}_i})$
6:     $N \leftarrow \mathsf{Dec}(k^e_{\mathbb{G}_i}, \mathsf{Suffix}(N', \mathsf{routable\_prefix}))$
7:     $\mathsf{data} \leftarrow \mathsf{RetrieveContent}(N)$
8:     return $C(N') := (N', \mathsf{data}, pk^s_{\mathbb{G}_i})$
9: else
10:    Drop $I(N')$
```

The protocol for IBAC-protected content retrieval relies on this rule. Algorithms 4 and 5 outline the interest and content object generation procedures. Note that the function $\mathsf{Suffix}(N, \mathsf{routable\_prefix})$ returns all name segments of N except the ones included in routable_prefix.[10] Also, the router verification procedure is outlined in Algorithm 6. If this procedure returns

[9]The content object signature must also be computed over $pk^s_{\mathbb{G}_i}$ to bind it to the message.

[10]For instance, Suffix(`/edu/uci/ics/home.html`, `/edu/uci/`) would return `ics/home.html`.

Algorithm 6 Router authorization check

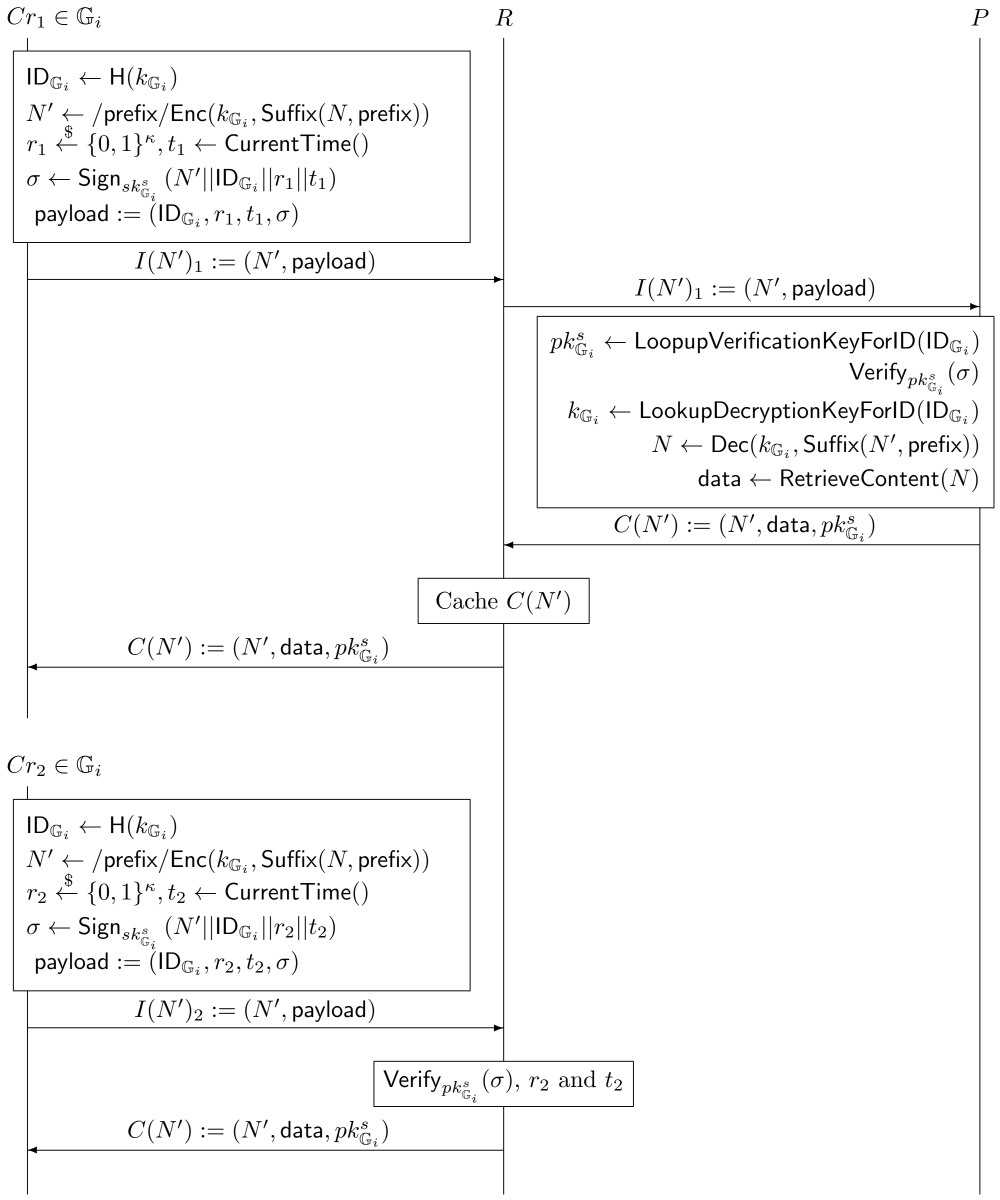
```
1:  INPUT: $I(N')$, cached $C(N')$, $\mathbb{B}$
2:  $(\mathsf{ID}_{\mathbb{G}_i}, r, t, \sigma) :=$ Payload
3:  $(N', \cdot, pk^s_{\mathbb{G}_i}) := C(N')$
4:  if $\mathbb{B}[N']$ contains $r$ then
5:      Drop $I(N')$; return Fail
6:  else
7:      if Timestamp $t$ is invalid then
8:          Drop $I(N')$; return Fail
9:      else
10:         if $\mathsf{Verify}_{pk^s_{\mathbb{G}_i}}(\sigma)$ then
11:             $\mathbb{B}[N'] := \mathbb{B}[N'] \cup r$
12:             return Pass
13:         else
14:             Drop $I(N')$; return Fail
```

Pass, then the content object found in the cache is forwarded downstream to the associated interface. Note that Algorithms 4, 5, and 6 use obfuscation key $k_{\mathbb{G}_i}$ and signing key pairs $(pk^s_{\mathbb{G}_i}, sk^s_{\mathbb{G}_i})$. For completeness, a complete sequence diagram showing multiple interest-content exchanges is shown in Figure 4.4. Both consumers belong to the same access group, i.e., $Cr_1, Cr_2 \in \mathbb{G}_i$.

We now prove that this variant of IBAC – with authorized disclosure – is secure in the presence of replay attacks.

Theorem 4.2. *The IBAC scheme with authorized disclosure is secure, in presence of replay attacks, against $\mathcal{A}$ if an indistinguishably-secure (IND-secure) deterministic encryption algorithm is used with an existentially unforgeable signature scheme.*

Proof. In Theorem 4.1, we proved that $\Pr[\mathsf{Guess}] \leq \epsilon(\kappa)$. It is easy to see that the additional `Payload` information – the random nonce, timestamp, and signature – are all distinct for each interest. Therefore, including this information leaks no information that improves the adversaries advantage or improves $\Pr[\mathsf{Guess}]$.



$Cr_1 \in \mathbb{G}_i$
R
P
$\mathsf{ID}_{\mathbb{G}_i} \leftarrow \mathsf{H}(k_{\mathbb{G}_i})$
$N' \leftarrow /\mathsf{prefix}/\mathsf{Enc}(k_{\mathbb{G}_i}, \mathsf{Suffix}(N, \mathsf{prefix}))$
$r_1 \overset{\$}{\leftarrow} \{0,1\}^\kappa, t_1 \leftarrow \mathsf{CurrentTime}()$
$\sigma \leftarrow \mathsf{Sign}_{sk^s_{\mathbb{G}_i}}(N' || \mathsf{ID}_{\mathbb{G}_i} || r_1 || t_1)$
$\mathsf{payload} := (\mathsf{ID}_{\mathbb{G}_i}, r_1, t_1, \sigma)$
$I(N')_1 := (N', \mathsf{payload})$
$I(N')_1 := (N', \mathsf{payload})$
$pk^s_{\mathbb{G}_i} \leftarrow \mathsf{LoopupVerificationKeyForID}(\mathsf{ID}_{\mathbb{G}_i})$
$\mathsf{Verify}_{pk^s_{\mathbb{G}_i}}(\sigma)$
$k_{\mathbb{G}_i} \leftarrow \mathsf{LookupDecryptionKeyForID}(\mathsf{ID}_{\mathbb{G}_i})$
$N \leftarrow \mathsf{Dec}(k_{\mathbb{G}_i}, \mathsf{Suffix}(N', \mathsf{prefix}))$
$\mathsf{data} \leftarrow \mathsf{RetrieveContent}(N)$
$C(N') := (N', \mathsf{data}, pk^s_{\mathbb{G}_i})$
Cache $C(N')$
$C(N') := (N', \mathsf{data}, pk^s_{\mathbb{G}_i})$
$Cr_2 \in \mathbb{G}_i$
$\mathsf{ID}_{\mathbb{G}_i} \leftarrow \mathsf{H}(k_{\mathbb{G}_i})$
$N' \leftarrow /\mathsf{prefix}/\mathsf{Enc}(k_{\mathbb{G}_i}, \mathsf{Suffix}(N, \mathsf{prefix}))$
$r_2 \overset{\$}{\leftarrow} \{0,1\}^\kappa, t_2 \leftarrow \mathsf{CurrentTime}()$
$\sigma \leftarrow \mathsf{Sign}_{sk^s_{\mathbb{G}_i}}(N' || \mathsf{ID}_{\mathbb{G}_i} || r_2 || t_2)$
$\mathsf{payload} := (\mathsf{ID}_{\mathbb{G}_i}, r_2, t_2, \sigma)$
$I(N')_2 := (N', \mathsf{payload})$
$\mathsf{Verify}_{pk^s_{\mathbb{G}_i}}(\sigma)$, r_2 and t_2
$C(N') := (N', \mathsf{data}, pk^s_{\mathbb{G}_i})$

Figure 4.4: IBAC content retrieval flow

We now assess Pr[Bypass]. Recall that this event occurs when $\mathcal{A}$ bypasses the authorization check at a router to recover content from a cache. Without knowledge of $sk^s_{\mathbb{G}_i}$, this only occurs if $\mathcal{A}$ is able to forge the `Payload` signature. By definition of the existentially unforgeable signature scheme, $\mathcal{A}$ is not able to generate an input set $(\hat{N'}, \hat{\mathsf{ID}_{\mathbb{G}_i}}, \hat{r}, \hat{t}) \neq (N', \mathsf{ID}_{\mathbb{G}_i}, r, t)$ such that $\mathsf{Verify}_{pk^s_{\mathbb{G}_i}}(\hat{\sigma})$ occurs with non-negligible probability. Thus, Pr[Bypass] $\leq \epsilon(\kappa)$. Finally, since the sum of two negligible probabilities is also negligible, then Pr[$\mathsf{Guess}+\mathsf{Bypass}$] $\leq \epsilon(\kappa)$. $\square$

Serving Content to Multiple Access Groups

One problem with encryption-based name obfuscation occurs when a content object with name N is accessible by different groups. According to Algorithms 4 and 5, the obfuscated name N' contains a suffix encrypted with keys associated with each access group. Therefore, a single content object might have several names depending on the number of groups authorized to access it. Since routers use exact matching for cache lookup[11], several copies of the same content could possibly be cached.

To address this problem, content objects should have the exact same name regardless of access control groups permitted access. This can be achieved using the hash-based name obfuscation function described in Section 4.2.2. However, per the ACKB rule, cached content needs to contain *every* authorization signature verification key that could be used to access said content. In other words, producers need to provide all possible public keys that can be used to access the content under IBAC protection. Consider the following: a content object $C(N)$ is accessible by two access groups $\mathbb{G}_i$ and $\mathbb{G}_j$. In this case, the producer needs to provide both $pk^s_{\mathbb{G}_i}$ and $pk^s_{\mathbb{G}_j}$ with $C(N')$, i.e.,

$$C(N') := (N', \mathsf{data}, pk^s_{\mathbb{G}_i}, pk^s_{\mathbb{G}_j})$$

[11]In CCN, not in NDN.

Whenever R caching $C(N')$ receives an interest issued by a consumer in any authorized access group, R uses the group identity included in the `Payload` field to determine σ's verification key.

Note that content object sizes might increase significantly depending on how many groups are allowed access. We do not discuss this issue further, since the trade-off between having multiple cached versions of the same content and having longer content objects carrying all verification keys is ultimately the application's decision.

IBAC Variations

We do not claim that any IBAC variation discussed above is superior to another. Each has its own strengths and weaknesses. However, to help determine which variation to use, we make the following claims based on the application needs and assumptions. Note that some claims provide privacy as well as access control.

1. If *replay attacks* are not a concern, then consumers only need to use a name obfuscation function and include their group identity in the `Payload`.
2. If *replay attacks* are plausible and *name privacy* is a concern, then name obfuscation must be used and authorization information, as described in Section 4.2.3, must be included in interest payloads.
3. If *replay attacks* are plausible but *name privacy* is not a concern, then only authorization information is sufficient.

Claim 3 might seem counterintuitive with the idea of IBAC. Recall, however, that router authorization checks prevent unauthorized consumers from retrieving cached content under IBAC protection. Even if content name is not obfuscated, $\mathcal{A}$ cannot forge payload authorization information, and therefore cannot violate IBAC protection guarantees.

Revocation

Generally speaking, revocation is a challenge in all access control schemes involving secrets shared among group members. Recall that all consumers belonging to the same access control group in IBAC share the same obfuscation keys. If one of them leaves the group[12], the producer must generate a new key and distribute it to all remaining authorized consumers. We will not discuss this issue further since we believe it is not part of the core access control *protocol*.

Moreover, cached content may be accessed by revoked consumers. Assume $C(N)$ is IBAC-protected and cached in R. Assume Cr, connected (directly or indirectly) to R, is authorized to access $C(N)$. While $C(N)$ is cached, Cr's access is revoked. At the same time, the latter sends an interest requesting $C(N)$. R will then grant access and reply with $C(N)$ from its cache. This is because the cached version of $C(N)$ is not updated with the correct authorization information, i.e., verification key(s). This may be partially addressed by setting $C(N)$'s expiration time to a value that reflects consumer revocation frequency.

Online revocation protocols, such as OCSP [234], would induce extra communication between R and P, which nearly defeats the purpose of the cache entirely. In this case, R would be better suited forwarding the interest upstream to P. Another option for P is to distribute certificate revocation lists (CRLs) [89] with every fresh content. This, however, introduces further issues for routers and consumers. First, routers would need to store CRLs and keep them updated frequently. Second, authorized consumers would need their own public and private key pair to compute σ. Lastly, routers would need to perform additional verifications against the CLR. Overall, this approach increases storage, consumer management, computation, and bandwidth complexity.

[12]For instance, consumers not renewing their subscription for a certain service.

Table 4.3: Overview of per-interest IBAC computational overhead.

IBAC Variation		**IBAC Computation Overhead**	
		Routers	**Producers**
Name Obfuscation	**Encryption**	None	One decryption
	Hash	None	One hash table lookup
Interest Signatures	**Encryption**	One nonce, timestamp, and signature verification	One decryption, one signature verification, Two hash table lookups (decryption key and signing key resolution)
	Hash	One signature verification, one nonce and timestamp verification, one hash table lookup (signing key resolution)	One signature verification, three hash table lookups (decryption key, signing key and name resolution)

4.2.4 Analysis and Evaluation

In this section, we analyze the overhead of each IBAC variation.

Computational Overhead

Computational overhead for routers and producers is expressed in terms of cryptographic and data structure operations, e.g., signature verification and hash table lookup costs.[13] Table 4.3 summarizes these results. To further understand the computational overhead, we compare two cases: (1) when routers perform authorization checks, and (2) when they do not. Let $\tau_{overhead} = \tau_{check} + \tau_{verify} + \tau_{update}$ be the authorization check overhead when routers receive interests, where τ_{check} is the time required to check for nonce duplication and timestamp staleness, τ_{verify} is the time to verify the payload signature, and T_{update} is the time to update the nonce collection. Since cache lookup and interest forwarding are performed regardless of whether or not routers perform authorization checks, we omit them from this

[13]It is assumed that the cost of any additional checks necessary to determine if an interest requires further IBAC processing is negligible. For example, this check be done using a simple flag in the interest. Thus, this overhead is omitted from our estimates.

equation. Similarly, τ_{check} and τ_{update} are negligible when compared to the cost of signature verification τ_{verify}; thus, they are also excluded.

A router incurs a computational cost of $\tau_{overhead}$ for every interest requesting content under IBAC protection. Therefore, we quantify $\tau_{overhead}$ by measuring the time to perform a single signature verification. We also experiment with batch verification techniques to amortize the cost of signature verification across multiple interests. While this naturally increases content retrieval latency since signatures are accumulated for batch verification, it reduces router computational overhead. Whether or not to use batch verification is up to the router's discretion. Furthermore, batch verification requires that IBAC-protected content objects for which interests are being verified cannot be evicted from the cache while the batch is collected. Table 4.4 shows the improvement using a variety of signature verification algorithms. Algorithms were implemented and evaluated with Crypto++ [3] and run on a machine with an Intel(R) Core(TM) i5-3427U CPU operating at 1.80GHz with 8GB of main memory and using Ubuntu 14.04. Note that, when modeling interest arrival rates using a Poisson distribution with arrival rate λ_i for $i \in \{40, 80, 160, 240\}$, both individual and batch signature verification incur nearly the same overhead in certain conditions, as we show below.

Table 4.4: Individual and batch ElGamal signature verification times.

Key Size	**Batch Size**	**Sig. Size**	**Indiv. Time**	**Batch Time**	**Improved**
1024b	10	512KB	0.599s	0.322s	46%
1024b	10	8MB	0.888s	0.615s	30%
1024b	50	512KB	2.918s	1.579s	46%
1024b	50	8MB	4.315s	2.991s	30%
2048b	10	512KB	4.065s	2.207s	46%
2048b	10	8MB	4.104s	2.269s	45%
2048b	50	512KB	20.081s	11.029s	45%
2048b	50	8MB	21.301s	12.536s	41%
3072b	10	512KB	12.406s	6.789s	45%
3072b	10	8MB	12.804s	7.122s	44%
3072b	50	512KB	60.174s	32.877s	45%
3072b	50	8MB	64.347s	35.601s	45%

Denial of service (DoS) is an obvious concern if routers perform authorization checks (the interest rate decays to 0 in our experiments as the need for verification increases). Let λ be the rate of arrival interests for IBAC-protected content cached in router R, and let μ be the service rate for interests, i.e., the rate at which interests are processed (parsed, verified, etc.). If $\mu < \lambda$, then R's throughput will degrade to zero over time [158]. We envision that in legitimate scenarios without malicious entities generating interests with fake authorization information, only a small percentage δ of arrival interests will be requesting content under IBAC protection. To assess DoS attacks in the presence of IBAC authorization checks, we empirically analyze the effect of δ on the interest service rate of a router. These service rates, which use different signature verification techniques – individual and batch – denoted μ_S and μ_B, respectively, are shown in Figure 4.5.

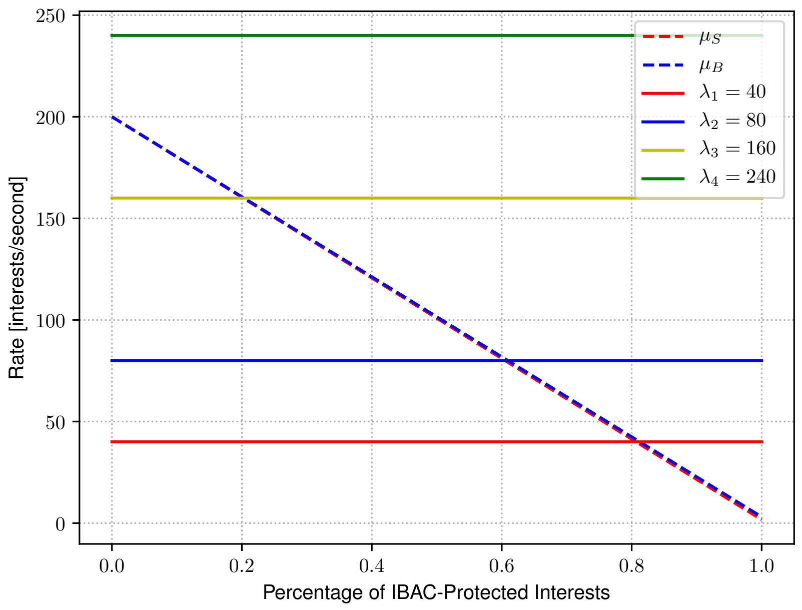


Figure 4.5: Interest service rates IBAC-protected interests

We assume that interests arrive at a base rate of $\lambda_1 = 40$ [74]; larger values for λ are provided to see at which point $\mu < \lambda$ due to authorization checks. By the exponential property of the

Poisson process, μ is calculated as follows:

$$\mu = \frac{1-\delta}{\tau_{process}} + \frac{\delta}{\tau_{process} + \tau_{verify}},$$

where $\tau_{process}$ represents interest processing time not including signature verification[14], and τ_{verify} is the time required to perform individual or batch signature verification. In our experiment, we assume a constant $\tau_{process} = 0.005$s and only vary τ_{verify}. To do so, we assume a key size of 1024b, batch size of 10, and signature size of 512KB. According to Table 4.4, this results in $\tau_{verify} = 0.599$s and $\tau_{verify} = 0.322$s for individual and batch verification, respectively. Our experiments show that decay of μ as a function of δ is almost identical for both batch and verification techniques. This is partly because only a small fraction of interests are verified. Furthermore, our results show that $\mu > \lambda$ is true, i.e., the router servicing process is stable for reasonable interest arrival rates. Our experiments show that $\mu < \lambda$ when $\lambda = 160$ and $\delta \geq 0.2$. Moreover, when a Poisson process is assumed, both individual and batch signature verification perform similarly for small values of δ. However, batch signature verification is advantageous with larger δ values. For instance, for $\delta = 0.2$, batch verification provides less than 1% service rate improvement, while it provides $3-46\%$ improvement for δ values $0.8-1$.

Storage Overhead

Storage overhead varies from producer to router. If hash-based name obfuscation is used, producers incur the cost of maintaining a hash table to map obfuscated names to their original values. However, if content name contains variable name segments, e.g., query string-like values in URIs, the hash table size can grow significantly since it has to contain all possible variations. Moreover, producers must bear the storage cost of IBAC access group keys if encryption-based obfuscation functions are used. Similarly, routers must bear the cost of

[14] $\tau_{process} = 1/mu$ for interests not requesting IBAC-protected content.

storing variable-length tuples of key identities $\mathsf{ID}_{\mathbb{G}_i}$ and the actual verification keys $pk^s_{\mathbb{G}_i}$, along with a theoretically unbounded collection of nonces for each IBAC-protected content. Moreover, these storage costs are proportionate to the number of unique producers that generates IBAC content.

Bandwidth Overhead

In terms of bandwidth overhead, each interest and content object is expanded to include additional authorization information, e.g., interest payloads with authorization information and content objects with authorization keys. Interests without authorization payloads will only increase (or decrease) by the expansion factor of the obfuscated name. If authorization payloads are included, then interest messages will grow by $|r|+|t|+|\sigma|+|\mathsf{ID}_{\mathbb{G}}|$, where $|r| = \kappa$. Content object $C(N)$ size increases with length $\sum_{i=1}^{L} |pk^s_{\mathbb{G}_i}|$, where L is the number of access groups allowed to access $C(N)$ and $|pk^s_{\mathbb{G}_i}|$ is the public key size associated with group $\mathbb{G}_i$.

Note that increased overhead may cause interests to exceed MTU sizes, which would induce interest fragmentation. However, given that names may themselves be unbounded, interest fragmentation seems unavoidable.

4.3 Best Effort Autonomous Deletion in CCN

One notable drawback of the libertarian approach to caching in CCN is that some content may need to be deleted *before* its expiration time. Consider content that frequently, yet sporadically, evolves over time, e.g., news articles. The appearance of breaking-news articles is unscheduled. As situations develop, updates and corrections to the content occur at unpredictable times. Such updates supersede previously distributed content by rendering it stale. In this case, producers need a way to remove old content. Another example is content which contains erroneous information. As errors are detected and corrected, a producer needs to flush the incorrect older version. Additionally, content may need to be flushed if access control policies change, e.g., a consumer's access to cached content is revoked.

Stale content occurs because `ExpiryTime` is the only way for a producer to communicate *anticipated* content lifetime to the network. However, a producer cannot change its mind after content has been published and distributed. Thus, there is a need for safe and secure in-network content deletion. To this end, we present BEAD: Best-Effort and Autonomous Deletion. In the process, we encounter and address several challenges for autonomous deletion, including efficacy, performance, and security. We also experimentally assess the proposed technique.

4.3.1 BEAD Requirements

One intuitive way of removing stale content from routers' caches is through the use of versioning, whereby the content naming format includes a segment that explicitly reflects the current version. For example, the content of BBC's World News web-page could be named: `/bbc/news/world/T_VERSION=2.4`. Alternatively, timestamps could be used. In that case, the same BBC page could be named `/bbc/news/world/1449187200`.[15] In either case, it is unclear how a consumer would determine (in advance) the current timestamp or version number, without which an interest cannot be formed.[16]

The main problem with versioning and timestamps is that they can not handle unpredictable content updates. In CCN, producers are oblivious to where and for how long their content is cached in the network. Although opportunistic caching is one primary advantage of CCN,

[15] 1449187200 is 12/04/2015 at 12:00am UTC.

[16] There is one trivial way: a consumer contacts the producer directly and asks for the most recent version number or timestamp. However, this would incur an extra round-trip delay per content retrieval.

it greatly complicates deletion of stale content. We believe that, in order to address the problem, producers need:

1. A way to communicate a single deletion request to all routers that might have cached offending content.
2. A way to efficiently authenticate deletion requests (allowing routers to quickly authenticate them) while avoiding trivial DoS attacks.

The first requirement is reminiscent of IP traceback – a class of techniques for identifying the original source of a (usually malicious) packets with traces. In the context of IP, this is often framed as a mechanism to mitigate DoS attacks. In this section, one goal is to learn *where* content was previously forwarded so that deletion requests can be routed along these same paths, which terminate at interest origins. Thus, ideas from IP traceback based on packet logging (e.g., [288]) and (deterministic or probabilistic) packet marking (e.g., [154, 50]) influence the design and forwarding strategies of BEAD messages.

We now show how to address these requirements with BEAD.

4.3.2 Authenticating Deletion Requests

Producers must prove content ownership to routers that receive deletion requests. Otherwise, an adversary can impersonate a producer and invoke content deletion, resulting in another form of DoS. One way to authenticate deletion requests is by a producer-generated digital signature on each request. However, besides being inefficient, forcing routers to verify signatures on deletion requests can be itself parlayed into DoS attacks [136, 150]. Moreover, it may necessitate public key retrieval, certificate handling, and other non-trivial (for routers) issues.

Our approach uses a lightweight token that proves content ownership. It works as follows. When a producer P creates a content object C, it generates a random λ-bit string x_C, called the *deletion token*. P then computes the digest of this token using a suitable cryptographic hash function[17], $y_C = H(x_C)$, and includes y_C in C. Later, if and when P wishes to delete C from the network, P includes x_C in the deletion request. (For now, we assume that P can route these requests to any router caching C.) Upon receipt, each R verifies that y_C (cached alongside the content) matches $H(x_C)$. If so, R knows that P must have issued the request and deletes C from the cache.[18]

4.3.3 Routing Deletion Requests

The remaining (though major) issue is how to route deletion requests from P to each router which could have cached offending content. Let d be the λ-bit hash of $C(N)$, i.e., $d = H(C(N))$. Let $E(N, d)$ be a deletion request, or erase message, for content named N and hash digest d. Let $\mathbb{R}_N$ be the set of routers that cached $C(N)$. Finally, let FIB^R be R's FIB. From here on, we use the term erase message to refer to a deletion request. Also, we assume that erase messages are authenticated using the method described in Section 4.3.2.

Flooding

We begin by considering the simplest approach: reverse-path controlled flooding [47] of deletion requests. When $R \in \mathbb{R}_N$ receives $E(N, d)$, it forwards it on all interfaces except those which have a matching FIB entry, i.e., to all interfaces through which the producer is not reachable.

[17]Suitable hash functions include those with pre-image resistance, which means that, given y, it is difficult to find an x such that $y = H(x)$.
[18]This is due to the randomness of x_C and the collision-resistance of $H(\cdot)$.

Flooding offers some advantages, the most important of which is the ability to reach network edges even if routers in $\mathbb{R}_N$ no longer cache offending content. This is important since routers do not cache content uniformly, and some may not even have caches. However, with flooding traffic volumes generated from a single `erase` message would be very high as most would be forwarded to routers that never even cached the target content.

Forwarder Histories for Content Traceback

Ideally, routers would only forward `erase` messages on interfaces to which the referenced content had been previously forwarded. In other words, `erase` messages should only be forwarded along the content distribution spanning tree where the producer is the root and leaves are interest origins. One way to forward `erase` messages along the edges of this tree is for each router $R \in \mathbb{R}_N$ to maintain a forwarding history of $C(N)$. There are several places where this history can be kept, including: (1) in the cache where $C(N)$ is stored, (2) in a separate forwarding log (similar to [288], as a form of IP traceback) at each router, and (3) in packets themselves. In each case, historical information constitutes a form of traceback that allows routers to identify where content was previously forwarded. We now describe each approach in more detail.

In-Cache Forwarding Histories : When a router caches $C(N)$ it can also remember the downstream interfaces where $C(N)$ was forwarded. We denote this set of interfaces as $\mathbb{F}_N$. When a router receives an interest $I(N)$ on interface F_i, it responds with $C(N)$ and adds F_i to $\mathbb{F}_N$. For a router with K interfaces, this additional state costs $\mathcal{O}(K)$ bits per cache entry. When a router caching $C(N)$ receives $E(N, d)$, it forwards it on all interfaces in $\mathbb{F}_N$.

In-cache forwarding histories are only effective for routers with large caches, since forwarding information lifetime is bound to cache entry lifetime, which can be small or even zero (if a router has no cache at all). Since $\mathbb{F}_N$ is deleted whenever $C(N)$ is flushed from the cache, this can cause future $E(N,d)$ messages to not be forwarded to downstream routers which might still cache $C(N)$.

Local Forwarding Logs : Long-term packet logs (histories) have their roots in IP traceback techniques from the early 2000s, e.g., [297, 288]. The problem here is similar: routers need long-term histories of packets (content) that were previously processed and forwarded.[19] In this context, a history is a set-like data structure that allows content objects to be inserted and then later queried for membership. There are two types of histories: *lossless* and *lossy*. The former only return "yes" for content objects that have previously been inserted. In contrast, a lossy history might return false positives or negatives. Routers use these structures by associating one history to each interface. When a router receives $E(N,d)$ and $C(N)$ is not cached, it forwards $E(N,d)$ on each interface for which the corresponding forwarding interface history has a record of $C(N)$, i.e., all histories for which membership query returns "yes". This procedure is outlined in Figure 4.6.[20]

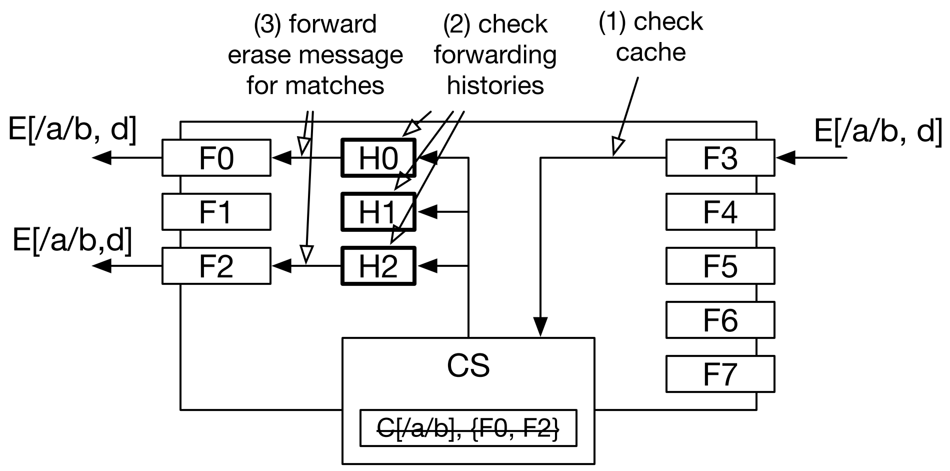


Figure 4.6: erase message forwarding strategy

[19]As indicated in [288], historical information for Internet-scale traffic (IP packets) can not last beyond a few minutes, which might still be less than what we needed for BEAD.

[20]Similar to the flooding algorithm, this check is not performed for interfaces via which the content producer can be reached.

We now describe some ways of implementing lossless and lossy histories that vary in their computation and memory requirements.

Lossless Forwarder Histories require a unique identifier to be kept after a content object has been forwarded. We assume that content hash d serves as such an identifier (with collision probability negligible in λ). Implementing this type of forwarder history can be done trivially with a hash set HS_R as follows. To insert a content object into the history, compute and store d in HS_R. To query the history, return "yes" if $d \in HS_R$ and "no" otherwise. Insertion and lookup each require constant time.

Lossy Forwarder Histories store historical information in memory-constrained systems at the cost of false positives and false negatives. Similar to SPIE traceback [288], we use BFs [63] to implement lossy forwarder histories. BFs enable probabilistic set membership queries.

The choices of BF properties, e.g., size and hash functions, impact efficacy of this technique. Filters that saturate too quickly result in high false positive rates. If all interface filters become saturated then `erase` messages are effectively broadcast. Therefore, it is important to eventually remove stale elements from filters. Unfortunately, normal BFs do not provide element removal. However, so-called Counting Bloom Filters (CBFs) [111] support set membership queries with removal. Instead of using bits to indicate set membership, CBFs use counters. When loading an element into CBF, the counters corresponding to the output of the hash functions are increased by one. Consequently, removing an element is done by decrementing the same counters. The problem with CBFs is that one must know the element to delete. Since routers would discard content after inserting them into these filters[21], they have no way of knowing what content is in the filter, and thus what elements to eventually delete. Their only recourse is to remove elements by decrementing counters at random. Intuitively, a router should delete random elements from the filter (the history) at

[21] This is because content is only added to histories upon its removal from the cache.

a frequency which reflects the average `ExpiryTime` of received content. This can increase the false negative probability and reduce the possibility of delivering erase messages to their corresponding destination.

Variants of CBFs, such as Time-Decaying (TDBFs) [343, 190] and Stable (SBFs) [102] BFs can also be used. TDBFs have the property that elements are slowly removed from the filter over time, thereby keeping the rate of false positives minimized. However, the natural decay property may lead to false negatives. SBFs are dynamically resized to keep the false probabilities minimized. Similar to CBFs and TDBFs, SBFs also suffer from false negatives.

Interest Marking for Content Traceback Packet marking is a standard technique for IP traceback [154]. In the context of this work, marking is performed on interests to capture interest origins. This information can be later used to learn the interface to which an erase needs to be sent. Specifically, erase messages can carry marking information so routers can identify destination forwarding interfaces without storing any local state.

One trivial marking method is to append the arrival interface to each interest. Specifically, when R receives $I(N)$ on face F_i, R prepends (R, F_i) to a list contained in the header of the interest. Producers record these traces (hop sequences) upon receipt and include them in future erase messages. When R receives an erase message with a trace it pops the last element (R, F_i) off the trace list and forwards it on F_i.

This technique distributes the forwarding history among messages in the network. Therefore, this information must be secure. To illustrate this requirement, assume router R_i receives $E(N, d)$ with the sequence of hops

$$[(R_i, F_i), (R_{i-1}, F_{i-1}), \ldots, (R_2, F_2), (R_1, F_1)]$$

from interface F_{i+1}. R_i needs a way to securely guarantee that (R_i, F_i) was previously prepended, by itself, to the subsequence:

$$[(R_{i-1}, F_{i-1}), \ldots, (R_2, F_2), (R_1, F_1)].$$

Otherwise, an adversary can forge unsolicited erase messages with apparently correct trace sequences. Alternatively, one can modify existing sequences in erase messages to prevent them from being routed towards their destination.

One way of authenticating hop-sequence traces is for R_i to compute and include a Message Authentication Code (MAC) [181, 193] tag t_i over (relevant) interest details, e.g., the name and hop sequence. Specifically, R_i computes t_i and adds (R_i, F_i, t_i) to each interest hop sequence before forwarding them. Since erase messages carry a name of content to be deleted, each router can verify its precomputed tag before forwarding erase messages downstream. Since routers compute and verify tags locally, a key management and distribution protocol is not required. We do, however, assume that routers are able to generate and maintain cryptographic keys of sufficient length necessary for MAC computation. As an added feature, hop-sequence information can also be used for detecting both interest and erase loops [134].

Although trace-based forwarding can deliver erase messages to all routers on between interest origins (consumers) and producers, the latter must store each unique trace from each interest. This is because (1) each trace corresponds to only one path in the network, and (2) interests issued by multiple consumers are most likely to traverse different paths to the producer. Producers can attempt to compile all collected traces in a data structure forming a spanning tree. This structure would be included in erase message headers, allowing routers to forwarder erase messages correctly. The main disadvantage of this approach is that the size of the data structure grows linearly with the number of consumers and is most likely to be greater than average link MTU. This means that erase messages would likely be fragmented (and possibly re-fragmented), and hop-by-hop reassembly is unavoidable. Another alternative is for producers to send multiple erase messages one for each set of traces correlated to a hop-sequence. In Section 4.3.4, we compare and evaluate the performance and resource consumption of these two techniques.

4.3.4 BEAD Analysis

We now assess some routing strategies for erase messages. Let n_t^R be the total number of content objects forwarded by R at time t and let μ_F^R be R's content forwarding rate. Clearly, n_t^R grows monotonically as a function of μ_F^R.

Flooding Analysis

Recall that the reverse path flooding algorithm works by only sending broadcast messages to interfaces through which the *producer* is not reachable. Though effective, this is not scalable. If each router flooded erase messages then they would reach a set of routers $\mathbb{R}* \supseteq \mathbb{R}_N$, where it is likely that $|\mathbb{R}*| \gg |\mathbb{R}_N|$. Therefore, flooding should always be a last resort for erase messages. We assess the actual overhead of this technique in Section 4.3.5.

Lossless History Analysis

The memory (and possibly computational) cost of a lossless forwarder history grows as a function of t. Thus, history collection will inevitably saturate memory at some point. Let n_{max}^R be the total size (in entries) of the history memory for R. Saturation is reached at time t such that $n_t^R \geq n_{max}^R$. We compute the time required to saturate a lossless forwarder history in two scenarios. Assuming each content object is $4,096$B and hash digests are 32B.

- **Consumer-facing router:** We assume a caching consumer-facing router (e.g., an access point) with 4GB of history storage and data rate of 100 Mbps. This data rate is equivalent to a content forwarding rate of $\mu_F^R = 3,200$ Cps (content packets per second). If R operates at full capacity with a full cache, i.e., storing every forwarded content requires eviction of an already cached one, saturation will take $41,943$ seconds (approximately 12 hours). This window of time might be longer than the `ExpiryTime` of content objects that are subject to erasure. For instance, news feed pages are likely to be updated with a frequency faster than 1/12 hours.
- **Core router:** We assume a non-caching CCN core router with 1TB of flash history storage and data rate of 10 Tbps equivalent to $\mu_F^R = 335$ MCps. If R always operates at full capacity, i.e., forwards at 10 Tbps, saturation occurs in 102 seconds. In this case producers have a time window of less than 2 minutes to issue an `erase` message for content C after it was last served.

R's saturation time can be lengthened by increasing the size of the forwarder history. However, at this rate, the cost of adding more memory to make saturation time useful is far too expensive: 1TB for 2 minutes of history in a core router.

A very natural question arises: what happens when R's history storage is saturated? R can evict old history entries randomly, or according to some policy, e.g., LRU. However, keeping track of history entries' ages might lead to reduced performance. Another alternative is to divide history storage into smaller chunks, each corresponding to a set time window of history entries. Once history storage is saturated, the oldest chunk is erased to provide space for new entries. Using the consuming-facing router example above, 4GB of history storage can be divided into 12 chunks, each corresponding to one hour. The router could then erase the history recorded 12 hours ago in order to store history entries for the coming hour.

Lossy History Analysis

Lossy histories are useful when lossless ones are too expensive, e.g., in core network routers. Our lossy forwarder history construction uses BFs. Given an m-bit BF that stores n elements, the number of input hash functions k can be optimized and false positive probability can be estimated using Equation 4.1 [70]. The optimal value of k is also given as a function of m and n.

$$f(m, \cdot, n) \approx (0.6185)^{\frac{m}{n}}, \quad k = \ln(2) \cdot \frac{m}{n} \tag{4.1}$$

In practice, a router can optimize the number of hash functions to lower the false positive probability. An upper bound of k can be set to limit hashing overhead.

We now analyze lossy forwarder histories in the context of the two scenarios mentioned above with the same history storage and data rates. We also assume that each content object added to a BF changes the value of *new* distinct k bits from 0 to 1. Clearly, this is unrealistic, since we do not consider the possibility of overlapping of hash function outputs for different input elements. However, this assumption captures the worst-case scenario.

- **Consumer-facing router:** To maintain a maximum false positive probability of 10^{-32}, a BF of size 4GB can fit $n \leq 2 \times 10^8$ elements. Based on Equation 4.1, this requires $k = 120$ hash functions. Thus, saturation takes $89,478$ seconds (a little over one day).
- **Core router:** To maintain the same false positive probability, a BF of size 1TB can accommodate $n \leq 5.7 \times 10^8$ elements, which corresponds to $k = 107$ hashes. Here, saturation occurs in 245 seconds.

One major drawback of using BFs for lossy forwarder histories is that saturation is more difficult to resolve. Recall that, with lossless histories, a router can remove old entries in order to add new ones. A router could also delete the oldest chunk of the history once it is saturated. However, with lossy histories, a router can either: (1) flush the entire lossy history and start over, or (2) use CBFs which support element deletion with the use of counters. Unfortunately, both approaches introduce false negatives.

Packet Marking Analysis

Packet marking is computationally inexpensive since it requires a single MAC computation per interest and erase. Its main drawback is an increased memory footprint in interests along each hop. Recall that traces in the hop-sequence consist of: (1) router identifier, (2) interface identifier, and (3) MAC tag. Assuming a 2-byte interface identifier and a SHA-256-based HMAC [193], the total size of each hop sequence element is 38 bytes. Assuming a 16-hop router-level path,[22] this corresponds to an extra 608 bytes for each interest.

We now compare two hop-sequence techniques described in Section 4.3.3. Assume a tree topology with producer P at the root with height h, 2^h consumers at the leaves with height 0, and $2^h - 2$ routers. Assume all consumers request content C and all routers append hop-sequence traces to the corresponding interests. In this case, P receives 2^h interests, each with $h - 1$ traces. If P includes all these traces in a single erase message, its size would grow by $\left(2^h \cdot (h-1)\right) \times 38$ bytes. This grows to 35 MB for $h = 16$, which is clearly impractical.[23] On the other hand, if P decides to send a separate erase to each consumer it would generate 2^h erase messages. The same overall volume of traces (35 MB) will be sent from P to consumers. However, it would be split into numerous erase messages. One advantage here is that each erase messages size will likely not exceed the path MTU and therefore not require fragmentation.

[22]The average Internet hop-count is currently 16 [49].

[23]We defer designing a more efficient scheme for combining hop-sequence traces to future work.

Summary of BEAD

As follows from the above, BEAD is not a single, concrete protocol. It is a set of techniques for generating and distributing erase messages to routers which may have cached offending content. We presented several alternatives, each of which are practical in different networks and network locations. For instance, consumer-facing (caching) routers can keep lossless or lossy histories for at least a day. Meanwhile, interest marking is better suited for core network routers. Therefore, we believe that all aforementioned techniques can be used, in combination, for routing erase messages. Our specific recommendations for forwarding erase messages are as follows:

1. If R supports interest marking, the first tuple in the hop-sequence traces is valid and appended by the router itself, then information in the tuple is used to route the erase message downstream.
2. If the content is in R's cache, then in-cache history is used to route the erase message.
3. If the content is not in R's cache, but R keeps lossless or lossy histories, they are used for erase message routing.
4. Otherwise, R floods received erase messages.

Recommendation 1 is most appropriate for core network routers, 2 and 3 for less busy edge network routers, and 4 as a failover mechanism. Most routers would likely prefer to drop erase messages instead of flooding them. This is why BEAD is *best-effort*: it does not guarantee that each erase message will be delivered to all entities caching the offending content.

As mentioned before, not all published content is subject to future deletion. If routers can make this distinction, there is no need to record history entries about content that will not be deleted. Such distinction can be achieved by adding an optional `CanERASE` flag to content object headers. If this flag is not present, the default behavior is to assume that no erase messages will ever be sent for the corresponding content. Moreover, interests requesting content that will not be deleted are not required to be marked by routers. Producers could tell consumers what content is subject to deletion by overloading manifests. Specifically, each pointer in a manifest can contain the `CanERASE` flag. In this case, the interest header format should be modified to include this optional flag. Moreover, since it is not guaranteed that all content objects will be fetched from a manifest, the default behavior of (core) routers should be to append hop-sequence traces to interests if the `CanERASE` flag is missing.

4.3.5 Performance Assessment

Our simulations focused on two properties of BEAD: network overhead, i.e., additional bytes added for `erase` messages, and forwarder overhead for processing `erase` messages, i.e., amount of time it takes to process each `erase` message.

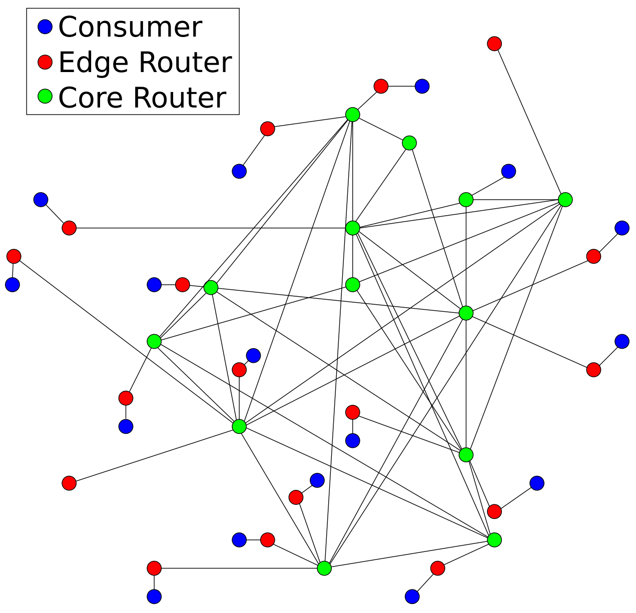


Figure 4.7: DFN topology

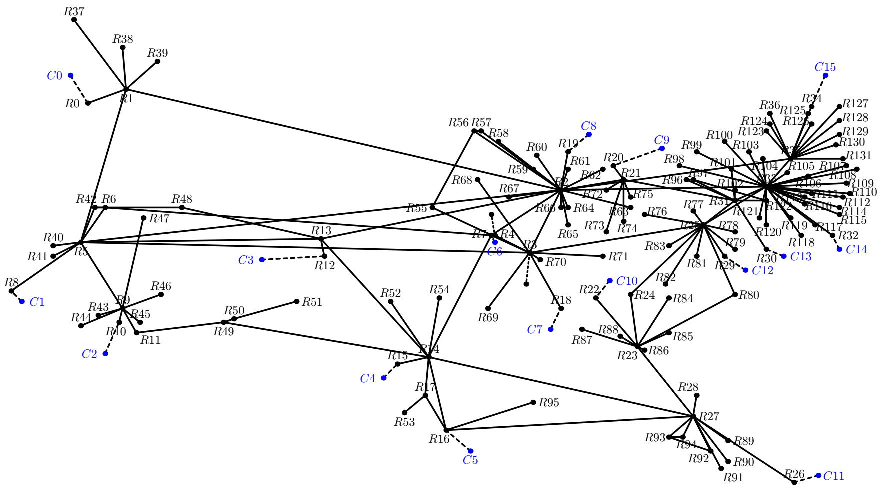
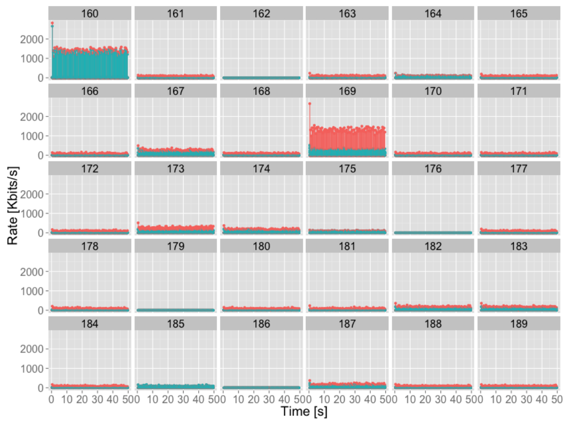


Figure 4.8: AT&T topology

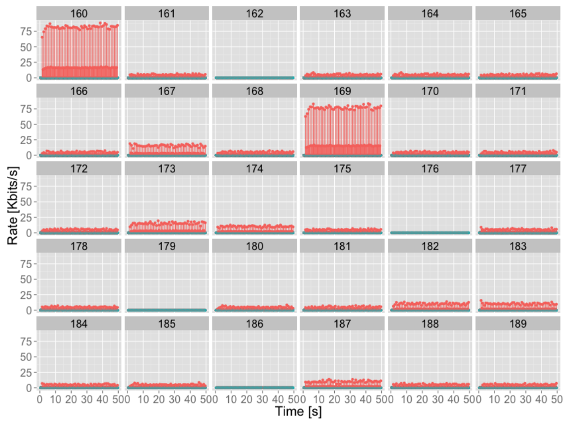
Network Overhead

To assess network overhead due to generating and forwarding `erase` messages we study the most costly scenario next to broadcasting: BEAD with lossless histories and routers with lossless links. To do so, we extended ndnSIM 2.0 [215], an implementation of NDN architecture as a NS-3 [9] module for simulation purposes, to support `erase` messages. With this modification, we ran two sets of experiments using the following topologies (shown in Figures 4.7 and 4.8, respectively):

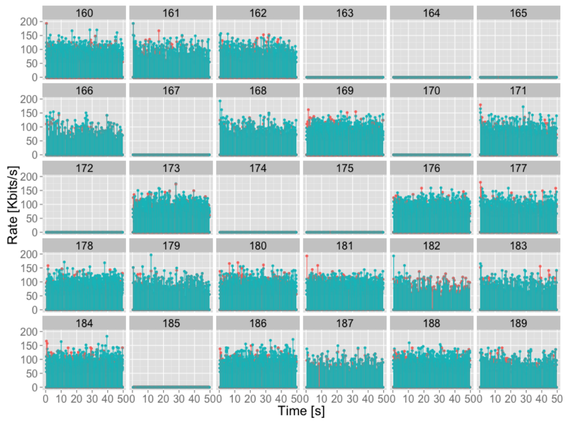
- The DFN network, Deutsches ForschungsNetz (German Research Network) [5, 6]: a German network developed for research and education purposes which consists of 30 connected routers positioned in different areas of Germany. The blue dots in the figure represent group of consumers (10 consumers per blue dot) connected to edge routers (red dots), while the green dots represent core network routers.
- The AT&T backbone network [85]. This consists of over 130 routers. Each logical consumer in the figure represents multiple (5) physical consumers connected to an edge router.



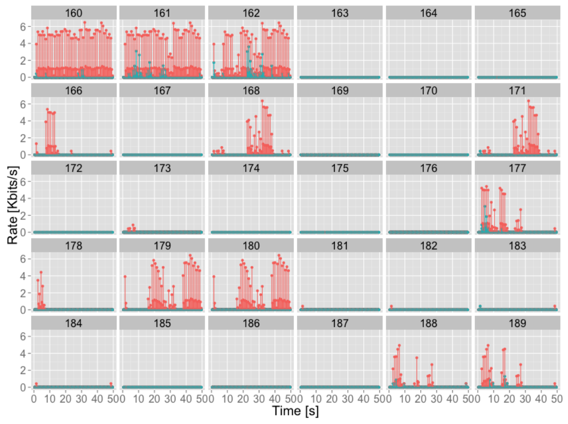
(a) Data processing overhead in the DFN topology with 160 consumers.



(b) `erase` message processing overhead in the DFN topology with 160 consumers.



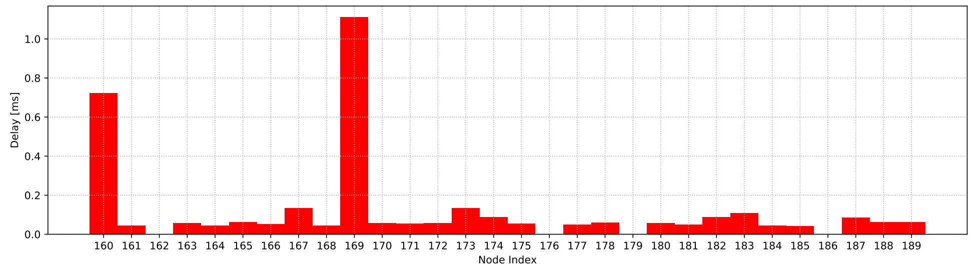
(c) Data processing overhead in the AT&T topology with 160 consumers. Not all routers are present in the image.



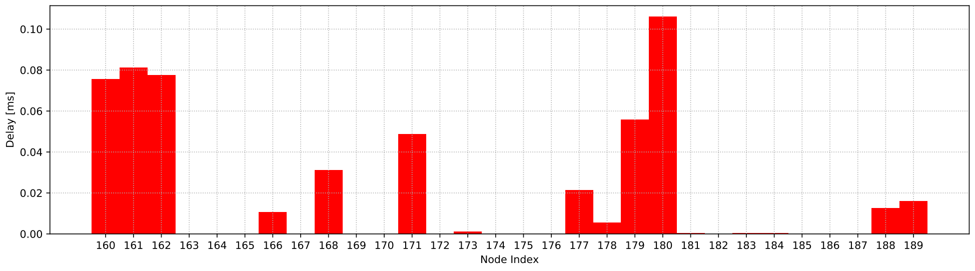
(d) `erase` message processing overhead in the AT&T topology with 160 consumers. Not all routers are present in the image.

Figure 4.9: `erase` message network overhead

In all experiments, consumers issue requests at a rate of 10 interests per second for content with the name prefix `/prefix` and monotonically increasing sequence number suffix. Every router uses a lossless history to record previously forwarded content objects for `erase` forwarding. Routers communicate over lossless links. Lastly, producers issue `erase` messages for 50% of their content every 1 second. (This may cause a producer to send a BEAD more than once.) Under these conditions, we measure router packet processing overhead with respect to content objects and `erase` messages. Figures 4.9a and 4.9b compare the overhead



(a) DFN topology with 160 consumers.



(b) AT&T topology with 160 consumers.

Figure 4.10: erase message forwarding overhead

of processing content objects and erase messages in the DFN topology with 160 consumers. Similarly, Figures 4.9c and 4.9d show the same type of overhead in the AT&T topology with the same number of consumers. Routers are identified by integers in the range [160..189]. InData (OutData) and InErase (OutErase) correspond to the amount of content object and erase traffic received from (sent to) an upstream (downstream) node, respectively. Ingress data is shown in red and egress data is shown in blue. Comparatively, we find that erase messages contribute very little overhead to the network with respect to the bandwidth consumed by content objects. Specifically, the total amount of erase message traffic in the DFN topology is 1.8% of the total content objects traffic, whereas it is only 0.09% in the AT&T topology. To understand these differences, consider Figures 4.9c and 4.9d. In Figure 4.9c, core routers receive and forward more content packets than those not in the core. In Figure 4.9d, those same core routers receive erase messages but do not forward all of them since they have were not in the history. This means that the content had previously been deleted. This is why the amount of egress traffic is less than the amount of ingress traffic.

We also assessed the computational overhead incurred by each router in these scenarios. The average time to process a single erase message for the DFN and AT&T scenarios is shown in Figures 4.10a and 4.10b. We see that only routers closest to the producer incur greater than 1.0ms to process an erase message since they almost always receive, store, and forward them.

4.3.6 Enabling Content Deletion

We now discuss potential economic incentives for routers and ISPs to support content deletion and implement BEAD.

BEAD and Accounting

Recall that BEAD is best-effort, unless flooding is used, which is undesirable. Barring major architectural changes, this seems to be optimal. However, if producers knew exactly where content was cached, then erase messages could be more accurately routed. For example, if a producer knew that a particular AS had a copy of the content cached *by some node in the system*, then the producer could specifically ask the AS to distribute an erase internally. This is far superior to routing erase messages across the core of the network in hopes that they *might* reach this AS (and any others with a cached copy).

We believe that it is possible to distribute content caching location information along with accounting information. A scheme for secure accounting in CCN [149] suggests that routers should notify producers of content they serve from caches by sending a so-called "push interest" or *pInt*. This approach can be modified such that: (1) AS gateways send *pInt* messages when content is cached in their domain and (2) *pInt* messages carry the prefix of an AS accounting management server within the AS.[24] Whenever a producer wants to delete

[24]Accounting management servers are centralized entities that manage accounting activities inside the AS.

certain content, it sends an erase message to each accounting management server – one per AS – that previously reported caching corresponding content. Then, the latter distribute the erase message within their ASes. Intra-AS distribution can be achieved via techniques described in Section 4.3.3. In fact, flooding might well be appropriate for that purpose since erase messages would not traverse AS boundaries.

The relationship between accounting and BEAD is natural. This is because one of the important applications of accounting is to bill for cache space. From an economic perspective, it would not be surprising for in-network caching to become a paid service. Routers and ASes could offer caching services for producers. A reasonable extension to this service would be to also offer a deletion service via BEAD.

BEAD in the Core

Flooding in the network core is not viable as a means of distributing erase messages. Moreover, forwarder histories and packet marking are (relatively) expensive operations and too costly for the fast path in the core. ISPs will likely just drop these messages due to a lack of economic incentive to forward them. Thus, in any plausible CCN network – where producers and consumers are at the edges of a network, while most traffic is routed through the core – erase messages are most likely to be propagated along only half of producer-to-consumer path(s). This is troublesome since content is most likely to be cached near consumers in edge (or near-edge) routers, and erase messages might never reach these routers.

To address this issue, core routers must be encouraged (with incentives) to carry and forward erase messages from producers to consumers. Since erase messages will typically amplify traffic, producers should be expected to pay for this increase. As before, this effectively turns BEAD into a service provided by ISPs that complements monetized caching; producers who pay for cache space may also have the choice to pay for on-demand deletion via BEAD.

Chapter 5

Privacy

Privacy is a primary goal of FIAs and similar next-generation network architectures. CCN partly achieves this goal. Consumers can retrieve named content without ever exposing their location via a source address. Interests only reveal immediate downstream hops from which they emanate. In contrast to IP, which mandates host-to-host source and destination packet addresses, this can obviate the need for anonymizing services such as Tor [13].

Despite this improvement, privacy in CCN is far from ideal. As discussed by Chaabane et al. [76] and Fotiou et al. [120], many privacy threats exist in CCN and related ICN architectures, including: traffic monitoring, inference, and invasion. Many core features of the architecture, sold as a manner of operational simplicity, usability, and efficiency, harm user privacy. For example, opportunistic and transparent caching is a core architectural feature. A cache hit only occurs if two consumers request identical content stored in a router. Thus, if an adversary wants to know if a nearby consumer requested a specific content, it can issue an interest for said content and check to see if it has been satisfied by a cache (by measuring retrieval latency).[1] This type of *cache probing* attack was first presented by

[1]This attack is significantly worse in NDN due to interest exclusion filters. Using these filters, an adversary can effectively probe, or harvest, a cache for all local content by requesting content with the default name '/' and a hop limit of 0.

Lauinger et al. [197, 196] and extensively studied independently by Mohaisen et al. [222, 221] and Acs et al. [22, 23]. Proposed countermeasures include randomly delaying requests to mask cache presence. However, this technique negates an important cache benefit: reduced observable latency. As such, Lauinger et al. [197] proposed more specialized countermeasures to cache probing attacks, including selective caching and traffic tunneling. The former can be implemented locally at each router, whereas the latter requires modifications to the CCN architecture. Chaabane et al. [76] proposed similar tunneling ideas. Compagno et al. [84] extended these probing attacks to a distributed adversary. They showed it is possible to use probing to geolocate specific consumers in the network.

Another privacy problem relates to names themselves. By default, names are not encrypted between consumers and producers. This means that interests and content objects carry *application names*, e.g., `/edu/uci/ics/sprout/caw/fileX`, in cleartext. Routers and other network nodes may learn information about content using these names, even if payloads are encrypted. Eavesdropping adversaries can observe which content is sought after by *some* downstream consumer. Georgen et al. [153] built name-based firewalls for CCN exploiting this visibility. In Section 5.1, we elaborate on privacy issues that stem from cleartext names. Section 5.2 further explores these issues and shows how statically encrypted interest and content packets are subject to classical frequency analysis attacks. Interestingly, caching helps mitigate this type of attack while simultaneously exacerbating cache probing attacks.

The literature abounds with approaches to address privacy problems, ranging from end-to-end encryption protocols, similar to TLS, to fundamental architectural changes. Techniques and countermeasures vary based on adversarial models and desired privacy goals. For example, Tor-like systems help when consumer and producer anonymity are desired. Anonymous communication over TCP/IP has a long and extensive history, mostly centered around two fundamental techniques: centralized anonymity proxies and distributed anonymity layering services. The Lucent Personalized Web Assistant [130] is one example of a centralized proxy interposed between communicating end-points. Unfortunately, such techniques are susceptible to passive eavesdropping attacks that monitor proxy activity. Their centralized nature also leads to a single point of trust and failure.

Most distributed approaches are based on Chaum's mix network approach to secure (anonymous) email [78]. The main idea is that several layers of concentric public-key encryption are applied to outgoing messages to traverse a specified set of mixes, each of which iteratively unwraps one layer of encryption and forwards the resulting payload to the next mix hop. By design, each mix buffers incoming messages that are decrypted, shuffles the buffer when a certain threshold is reached, and sequentially forwards each message after a random delay. This "mixing" strategy serves to thwart passive eavesdropping attacks since (given sufficient volumes of incoming messages) an adversary cannot correlate outgoing and incoming messages.

Other low-latency solutions based on mix networks are Babel [160] and Mixminion [94]. The main difference is that their goal is to provide anonymity with respect to a *global eavesdropper* adversary. To do so, each mix generates cover (chaff) traffic in addition to randomized delays. However, such unpredictable traffic delivery patterns make these solutions unsuitable for applications with low-latency requirements.

Low-latency anonymous communication techniques aim to minimize latency by avoiding batching (delaying) and re-ordering of messages as well as chaffing. As shown by Serjantov [279], traffic patterns in low-latency anonymity systems can be used to deanonymize clients. Notable examples include: Crowds [262] (vulnerable to local eavesdroppers and predecessor attacks [333]), Morphmix [267], Tarzan [128], and Tor [300] as well as its variant I2P [341]. Crowds is unique in that each mix probabilistically chooses whether to forward a decrypted message or send it to its final destination. Morphmix is distinct from Crowds in that it does not use a lookup service to track all participating nodes. It is also unique in that circuits are dynamically created by each mix. In particular, clients first pick a mix (entry point) for a circuit, which then randomly selects the next hop in the circuit along which messages are forwarded. Tarzan's unique property is that it uses globally verifiable *mimics* for each node to imitate bidirectional cover traffic, thus further obfuscating message flow between senders and receiver. Tor uses a centralized directory to locate and establish circuits through nodes (circuits of length three are sufficient for the required anonymity claims). Furthermore, these circuits are short-lived. The amount of available bandwidth at each node is taken into account during circuit establishment and multiple TCP connections are multiplexed over one circuit so as to achieve better performance. To improve throughput and mask traffic, communication between adjacent Tor nodes (in a given circuit) is secured via TLS. Finally, Tor does not introduce any decoy traffic or randomization to hide traffic patterns.

ANDāNA was the first mixnet-like anonymizing application for CCN [105]. Inspired by Tor, ANDāNA onion encrypts interest and content packets using anonymizing circuits. By default, onion encryption is done with public keys. Chung et al. [278] presented an alternative to ANDāNA wherein the hash of each interest is included in the public-key interest onion. This permits each hop of the circuit to inspect their PIT and cache as normal. Content objects are not onion-encrypted along the reverse path; pairs of anonymizing routers in a circuit share keys used to encrypt content in transit. Receiving routers decrypt content and store plaintext variants, indexed by corresponding name hashes. Section 5.3 describes $\mathsf{AC}^3\mathsf{N}$[309], a further enhancement to ANDāNA that replaces stateless public-key cryptography with stateful symmetric-key cryptography. Unlike Seo et al. [278], interest identifiers are not exposed to any hop in the circuit except the terminal node.

Arianfar et al. [42] presented another strategy for preserving consumer anonymity. In their system, producers mix sensitive content with so-called "cover content." Consumers then obtain content by requesting this mixture, hiding (to some extent) their real request. Aside from storage and bandwidth increases, this technique neither provides protection against malicious producers nor offers unlinkability between consumers and producers. Tao et al.

[303] described a similar approach wherein content is mixed together via randomized linear network coding. Both interest and content packets are chunked and encrypted for a reconstructing proxy that handles packet forwarding. Misra et al. [306] proposed a similar mechanism wherein interests are encoded with a huffman code. A trusted proxy decodes the interests and fetches content on behalf of the consumers. Another proxy-based design was described by Fotiou et al. [125]. Their approach involves a hierarchy of encrypted content brokers. Producers submit encrypted content to a broker hierarchy. Consumers query brokers for content and retrieve either a pointer to a child broker or original content producer(s). Each query is encrypted via additively homomorphic encryption, which permits brokers to operate on it without decryption.

Martinez-Julia et al. [214, 213] proposed a similar scheme that uses an overlay network and infrastructure for privacy and untraceability in CCN. Each network entity, e.g., router, has a digital identity and is part of a domain. A special component called the Domain Trusted Entity (DTE) manages entity-to-identifier associations and authentication. DTEs form an overlay network that enable so-called "identity-based" communication. Entities who wish to communicate with another tunnel communication through DTE overlays using identities to specify recipients. Elabidi et al. [110] proposed a privacy-preserving extension to CCN based on expiring identities. Their system involves identity providers, trust verification providers, and digital identity protection authorities. Identity providers assign and rotate identities for entities. Users query trust verification providers to check identifier validity. Publishers give users access to content through trust verification providers.[2]

Rembarz et al. [263] proposed a tunnel-based approach for private publishing in NetInf. Publishers and subscribers communicate through intermediary gateways. Publishers upload data to a private name resolver (PNR). Subscribers resolve names to gateways with whom they authenticate themselves. In turn, gateways resolve names to content for subscribers

[2]This is also a system for access control.

using the PNR. A different variant removes gateways and has the PNR create a private binding between content IDs and unlinkable IDs sent to the global NR. The PNR authenticates content requests and grants Kerberos-style tokens to subscribers to fetch from publishers.

Katsaros et al. [179] enumerated a subset of information leaked through CCN names, including: service type, ownership, caching properties, service class, scope, and content format. Certain pieces of information must be exposed for QoS purposes, e.g., service class and content format. The rest should be hidden. Chaabane et al. [76] suggested using BFs to mask real application names. They propose using a single BF for each segment in a hierarchy. BF parameter selection is ignored. Kazmi et al. [183] studied, in part, tradeoffs of privacy and network robustness in ICEMAN. Since ICEMAN publishers can control which nodes (brokers) have access to certain metadata tags (by encrypting these tags), access restrictions for improved privacy come at a cost of limited routing options. The authors experimentally evaluate this impact on routing in a simple grid topology using a variety of routing strategies, including promiscuous routing, i.e., with no privacy, universal routing, i.e., where all nodes shared access rights to decrypt metadata tags, and circumference routing, i.e., where only edge nodes can decrypt metadata. Time to fetch data increased as metadata access scope decreased, with a more significant increase for circumference routing.

In many cases, tunneling may be a simpler alternative when some measurable privacy is desired.[3] Tunneling can be done in the network or end-to-end between applications. (For comparison, these variants are analogous to IPsec and TLS in IP-based networks.) Each presents their own set of challenges. Section 5.4 describes our approach for (network-layer) namespace tunneling in CCN. Tunneling may also occur at the application layer, e.g., via secure sessions bootstrapped by CCNxKE [227, 228]. A different session-based scheme was designed by Wang et al. [327]. In contrast to CCNxKE, these session secrets are not forward secure. Moreover, their system requires a hosting service to function. CCNxKE is

[3] ANDāNA and similar schemes may be viewed as nested tunnels, and thereby provide greater privacy and anonymity benefits.

also application-agnostic, similar to TLS. The secure transport protocol of Renault et al. [265, 266] is also similar to CCNxKE. However, their scheme requires routers to participate with a trusted *security controller* to authenticate and authorize users and establish a shared secret. CCNxKE requires no router involvement.

Asghar et al. [44] designed a system called PROTECTOR that decouples application names from those used to route packets in the network. Specifically, edge routers and access points are given proxy re-encryption keys for every possible producer and user. Consumers encrypt interest names before sending them to the network. When an encrypted interest arrives at an edge router, the latter re-encrypts it to a "network ciphertext space," via proxy re-encryption [62]. Routers forward interests based on (now encrypted) names as per usual. Aside from key management and scalability problems, edge routers can only process 34 interests/second, which is inadequate for real-world applications. Moreover, consumers are required to obtain unique proxy re-encryption keys from some key manager at startup.

One downside to tunneling and related techniques is that it requires endpoint applications or network operators to opt-in. Section 5.5 describes a mechanism called TRAPS for obfuscating interests and encrypting content similar to PROTECTOR that does not have such requirements. Such transparent encryption is prevalent in IP-based networks. Protocols such as IPsec [186] and tcpcrypt [60] protect individual packets from eavesdroppers by encryption performed deep in the network stack. They both involve some handshake protocol, e.g., tcpcrypt key establishment or IPsec IKE, to establish a secure channel with a shared cryptographic key. In effect, IPsec and tcpcrypt encrypt all data above the network and transport layer of the TCP/IP stack, respectively, which protects packets sent between tunnel endpoints.

Per-packet encryption via tcpcrypt and IPsec provides confidentiality and privacy for communication with well-known hosts; only source and destination addresses are visible in cleartext to eavesdroppers. However, discovering host addresses via DNS often reveals information that breaks these privacy assurances. DNSCurve [56] and DNS-over-TLS [347] are two approaches to this leakage problem which seek to provide DNS query privacy by encrypting the contents of each request. These approaches are different from those that use Tor [300] to hide the location or origin of DNS queries since they only hide the query contents.

5.1 Data Privacy Challenges

At present, the baseline for privacy in the Internet is end-to-end transport encryption via, e.g., TLS [268]. Correlating information across multiple flows and their packets is infeasible in the absence of traffic and timing analysis. Clearly, CCN and related ICN architectures do not offer this degree of privacy. Names, signatures, and even payloads are transported in plaintext. To the best of our knowledge, this disparity has not been adequately addressed by the CCN community. To this end, this section analyzes CCN *data* privacy issues and shows how requests, responses, and a single request-response exchange can be made private. The contributions are[4]:

- Privacy analysis of the CCN request and response protocol.
- Evaluation of *weak* privacy techniques based solely on CCN names and their secrecy.
- Requirements for *strong* privacy with properties similar to TLS.
- Discussion of practical techniques that can address identified privacy issues.

One of our primary conclusions is that data encryption, by itself, is insufficient for privacy. Request names must have no correlation with data carried in a response, which strongly contradicts the name-based model of CCN and other ICNs. This implies that there must be some mapping between standard CCN names and information conveyed to the network. Moreover, in the presence of powerful adversaries, this transformation cannot be deterministic, since that would lead to frequency analysis attacks. (We explore this attack in Section

[4]These issues are also discussed by Ghali et al. [146].

5.2.) This effectively invalidates caches and is functionally no different from end-to-end encryption such as TLS in today's Internet. In total, we find that, if privacy similar to IP is desired, then many of the claimed benefits of CCN are lost, barring major architectural changes made to accommodate enhanced levels of privacy.

5.1.1 Data Privacy Pitfalls

According to Pinkas et al. [253], "the common definition of privacy in the cryptographic community limits the information that is leaked by the distributed computation to be the information that can be learned from the designated output of the computation." In networking and communication protocols, this notion of privacy refers to the limits of information leaked by traffic. This is a growing concern in recent years since pervasive monitoring of network traffic was found to be standard practice. Such eavesdropping is now considered a fundamental attack on privacy [114]. Other threats to privacy include correlation among a user's traffic flows or behaviors, identification of the user, disclosure of their personal information, and secondary-use of personal information [88].

With the shift from host-based to data-centric communication, CCN changes how data is retrieved and the way peers communicate. Recall that consumers issue a request for data D with the name N, which we denote as $D(N)$. In this context, N is the *application name* of D. The *network name* $\bar{N}$ carried in the wire-encoded packet and used to forward this request need not equal N. However, in standard CCN, $N = \bar{N}$. We use the notation $D(N)$ and $D(\bar{N})$ to refer to the data identified by the given application and network names, respectively. These requests may contain other information to identify the desired content object, such as KeyId or ContentId. In this section, we consider these and all other identifiers to be contained in N and subsequently encoded in $\bar{N}$.

Content object $C(\bar{N})$ carries $D(N)$. Consumers may use different network names $\bar{N}^0$ and $\bar{N}^1$ when requesting $D(N)$, in which case $C(\bar{N}^0) \neq C(\bar{N}^1)$. This can occur if $D(N)$ is uniquely encrypted for each consumer. Conversely, it always holds that if $C(\bar{N}^0) = C(\bar{N}^1)$ then both responses carry identical application data $D(N)$. Recall that it is not a requirement for a content object to carry a CCN name. However, a content object always carries an explicit or implicit identifier that can be matched to a value derived from the corresponding interest. For example, if the content object $C(\bar{N})$ does not carry a name, then its hash digest must match ContentId in $\bar{N}$. Thus, for presentation clarity, we do not distinguish between content objects with and without a name.

Privacy in CCN must be defined and assessed with respect to the requests and responses that are conveyed in the network, i.e., $\bar{N}$ and $C(\bar{N})$. Generally, an adversary $\mathcal{A}$ may try to recover N or $D(N)$ from this information. Our goal in the remainder of this section is to show what type of privacy is attainable based on the properties of $\bar{N}$ and $C(\bar{N})$. Before doing so, we specify our notion of data privacy and how it differs from its IP counterpart. We then describe the adversarial model and define relevant privacy terms.

Separating Privacy and Anonymity

In general, privacy is framed in terms of the endpoints that participate in a data exchange rather than a property of the data itself. For example, privacy might mean that the communicated data leaks no private information about the user. Another notion of privacy might be that identities of communicating endpoints, e.g., IP addresses, host names, and user IDs, remain hidden. We claim that elements of data privacy and personal privacy, or anonymity, are inappropriately mixed in general discussions of privacy. Therefore, we separate these two notions and focus solely on problems surrounding data privacy.

To show why this separation is useful, consider the following scenario. Suppose an adversary controls a pair of consumer- and producer-adjacent routers. Any (unmodified) requests and responses forwarded through the compromised routers link the consumer and producer. The adversary learns that (a) the consumer is fetching data from the producer or (b) the consumer and producer are communicating. However, **we do not consider this a data privacy leak** because consumer and producer linkability does not reveal information about data transported between them. Assuming appropriate request and response protection, consumer and producer linkability reveals no more than what is revealed by clients and servers engaged in a TLS session in today's Internet. Although the consumer and producer do not have anonymity, their traffic remains private in absence of traffic analysis and other side channels. If anonymity is required, a TOR-like mechanism such as ANDāNA [105] or AC^{3}N [309] can be used to decouple consumers from producers. In the rest of this section we focus solely on the problem of data privacy.

Adversarial Model

We now define the adversarial model against which privacy is assessed. According to Cooper et al. [88], there are at least three types of adversarial goals:

1. **Correlate**: Determine whenever any two consumers retrieve the same application data $D(N)$.
2. **Identify**: Discover or recognize that $D(N)$ was retrieved.
3. **Learn**: Obtain information about $D(N)$.

Learning information about $D(N)$ from $\bar{N}$ or $C(\bar{N})$ is harder than a correlation or identification attack. If an adversary learns information about $D(N)$ given $C(\bar{N})$ or $\bar{N}$, then it can also perform a correlation or identification attack. The converse is not necessarily true. We denote the adversaries with these goals as $\mathcal{A}^C$, $\mathcal{A}^I$, and $\mathcal{A}^L$, respectively.

An adversary with any subset of these goals may have different capabilities. One type might only be able to observe a single request and response from a consumer, while another type might observe all traffic from a set of consumers. One example of the latter would be a malicious Wi-Fi hot spot observing traffic of all hot spot users. Adversaries are also classified based on whether they are off-path or on-path, i.e., honest-but-curious (HbC) routers. An adversary that can only capture traffic without being on the consumer-to-producer path has a distinct disadvantage compared to the one that forwards traffic between a consumer and producer. For instance, an off-path adversary eavesdropping on an encrypted link can only observe encrypted traffic. Conversely, routers incident to that encrypted link can observe the original packets. In that case, the adversary can correlate requests with responses more easily and can also determine when two downstream consumers request identical or related content. A more powerful adversary controls multiple routers on the consumer to producer path. For example, an adversary which controls routers adjacent to communicating consumers and producers can easily learn when a particular consumer is requesting a specific content by inspecting the name.

We consider these capabilities to be characteristic of three distinct adversaries: an off-path attacker, a single on-path honest-but-curious router, and a collection of at least two (distributed) on-path honest-but-curious routers. Table 5.1 shows examples of these adversaries.

Response Privacy

Response privacy is centered on preventing information leakage from data responses. Ideally, an adversary should learn nothing from a response. We use a game-based definition to capture this notion of privacy. Let $\mathbb{N}_A$ and $\mathbb{N}_N$ be the set of application and network names, respectively, that can be assigned to data and bound to content objects. For every request for $D(N)$, $N \in \mathbb{N}_A$, there is a function $r(N)$ that generates and returns a response $C(\bar{N})$, where $\bar{N} \in \mathbb{N}_N$ (i.e., a network name for $D(N)$). In this game, $\mathcal{A}$ is given access to $r(\cdot)$

Table 5.1: Data privacy adversary examples.

	Goals	
Capabilities	**Correlate**	**Extract**
Eavesdropper	A user spying on encrypted traffic between a hotspot and neighbors with the goal of identifying traffic patterns and commonalities.	A user spying on encrypted traffic to identify specially marked or flagged content.
On-path HbC	An access point gathering statistics about how frequently content is accessed.	An access point censoring or restricting content based on its name or payload.
Distributed on-path	A pair of access points adjacent to consumers and a single producer trying to discern when certain content is requested and by whom.	A pair of access points logging when specific content is requested.

via an oracle $\mathcal{O}_r(\cdot)$. Upon receipt of a name N, $\mathcal{O}_r(N)$ returns $C(\bar{N})$. $\mathcal{A}$ also has access to an oracle to compute the inverse of $r(\cdot)$, $\mathcal{O}_r^{-1}(\cdot)$, which, upon receipt of $C(\bar{N})$ will return $N = r^{-1}(C(\bar{N}))$. The game works as follows.

- The challenger initializes and gives $\mathcal{A}$ access to $\mathcal{O}_r(\cdot)$, $\mathcal{O}_r^{-1}(\cdot)$, and $\mathbb{N}_A$.
- $\mathcal{A}$ issues a series of application names $N_0, \ldots, N_{i-1}$ to $\mathcal{O}_r(\cdot)$ and obtains $C(\bar{N})_0, \ldots, C(\bar{N})_{i-1}$, respectively.
- $\mathcal{A}$ generates two names $N_i^0 \neq N_i^1$ and sends them to the challenger. The challenger then generates a random bit b and returns $C(\bar{N})^b = r(N_i^b)$ and sends it back to $\mathcal{A}$.
- $\mathcal{A}$ continues to query $\mathcal{O}_r(\cdot)$ (including, if needed, names N_i^0 and N_i^1). $\mathcal{A}$ can also query $\mathcal{O}_r^{-1}(\cdot)$ for any data response (except $C(\bar{N})^b$). When done, $\mathcal{A}$ outputs a single bit b'.
- The game outputs 1 if $b = b'$ and 0 otherwise.

We denote the output as $\mathsf{DataGame}(\mathcal{A}, r)$. $\mathcal{A}$ wins if $\mathsf{DataGame}(\mathcal{A}, r) = 1$. We also consider one other variant of this game where $\mathcal{A}$ cannot access either oracle, denoted as $\mathsf{LIMDataGame}(\mathcal{A}, r)$.

We define privacy with respect to $r(\cdot)$, e.g., data is private with respect to correlation attacks if it is generated by a function $r(\cdot)$ that is also secure against correlation attacks.

Definition 5.1. $r(\cdot)$ *is secure against correlation attacks if for any probabilistic polynomial time (PPT) $\mathcal{A}$ it holds that*

$$|\Pr[\mathsf{DataGame}(\mathcal{A}, r) = 1] - \Pr[\mathsf{DataGame}(\mathcal{A}, r) = 0]| \leq \epsilon(\lambda)$$

for security parameter λ and where $\mathcal{A} = \mathcal{A}^C$.

Similarly, $r(\cdot)$ is secure against leakage and identification attacks if for any PPT $\mathcal{A}$ it holds that

$$|\Pr[\mathsf{LIMDataGame}(\mathcal{A}, r) = 1] - \Pr[\mathsf{LIMDataGame}(\mathcal{A}, r) = 0]| \leq \epsilon(\lambda)$$

for security parameter λ where $\mathcal{A} \in \{\mathcal{A}^L, \mathcal{A}^I\}$.

We now define weak and strong response privacy with respect to correlation and leakage privacy.

Definition 5.2. *A response has* weak privacy *if it is generated by a function secure against leakage and identification attacks. A response has* strong privacy *if it has weak privacy and is generated by a function secure against correlation attacks.*

Request Privacy

Similar to response privacy, request privacy is about information leaked by requests and network names. However, in contrast to response privacy, contents of a response may compromise privacy of the corresponding request. This is possible if $C(\bar{N})$ reveals information about N or $D(N)$, e.g., if $C(\bar{N})$ is part of a well-known media file. This complicates our notion of request privacy since we must also assume $\mathcal{A}$ can observe a response associated with each request. Specifically, let $\mathcal{A}$ be an adversary similar to $\mathcal{A}^I$ whose goal is to determine $D(N)$ given $\bar{N}$, i.e., to recognize that name $\bar{N}$ corresponds to some data item $D(N)$. We assume that application names are transformed by some function $q(\cdot, \cdot)$ to a network representation before interests are issued. We model this transformation as $q : \mathbb{N}_A \times \mathbb{N}^+ \to \mathbb{N}_N$, where $\mathbb{N}_N$ is the set of network names. (We require $|\mathbb{N}_A| \leq |\mathbb{N}_N|$.) The second parameter (from $\mathbb{N}^+$) denotes the length of the application name prefix that *is not modified by the transformation*. For example, $q($`/a/b/c`$, 1)$ would translate the suffix `/b/c` but leave the prefix `/a/` intact. In current CCN, $q(\cdot, \cdot)$ is simply the identity function: an application name N is the same name that would be carried by an interest in the network.

We now define a request indistinguishability game, wherein $\mathcal{A}$ is given access to $q(\cdot, \cdot)$ via an oracle $\mathcal{O}_{q,i^*}(\cdot)$ that computes the transformation after segment i^* of the name. That is, given a name N, $\mathcal{O}_{q,i^*}$ returns $\bar{N} = q(N, i^*)$. $\mathcal{A}$ also has access to an oracle to compute the inverse of $q(\cdot, \cdot)$, $\mathcal{O}^{-1}_{q,i^*}(\cdot)$, which, upon receipt of $\bar{N}$ returns $N = q^{-1}(\bar{N}, i^*)$. The game proceeds as follows.

- The challenger initializes and gives $\mathcal{A}$ access to $\mathcal{O}_{g,i^*}(\cdot)$ and $\mathcal{O}^{-1}_{g,i^*}(\cdot)$. $\mathcal{A}$ is also given $\mathbb{N}_A$ and $\mathbb{N}_N$.
- $\mathcal{A}$ issues a series of names $N_0, \ldots, N_{j-1}$ to $\mathcal{O}_{q,i^*}(\cdot)$ and collects the results $\bar{N}_0, \ldots, \bar{N}_{j-1}$.
- $\mathcal{A}$ presents a pair of names N^0_j and N^1_j to the challenger. It is required that the first i segments of N^0_j and N^1_j are identical. The challenger generates a random bit b and returns $\bar{N}^b_j = q(N^b_j, i^*)$ to $\mathcal{A}$.
- $\mathcal{A}$ continues to make queries to $\mathcal{O}_{q,i^*}(\cdot)$. $\mathcal{A}$ can also issue a request for any name as well as query $\mathcal{O}^{-1}_{q,i^*}(\cdot)$ with transformed names except $\bar{N}^b_j$. Next, $\mathcal{A}$ outputs a single bit b'.
- The game outputs 1 if $b' = b$ and 0 otherwise.

We denote the output as $\mathsf{NameGame}(\mathcal{A}, q)$. $\mathcal{A}$ succeeds if $\mathsf{NameGame}(\mathcal{A}, q) = 1$. We also consider a variant of this game where $\mathcal{A}$ has no access to the oracles called $\mathsf{INDNameGame}$. Here, $\mathcal{A}$ relies solely on the response from the challenger and its queries to the network when making its decision.[5] We use it to define name privacy with respect to $q(\cdot, \cdot)$.

Definition 5.3. *$q(\cdot, \cdot)$ is secure against correlation attacks if for any PPT $\mathcal{A}$ it holds that*

$$|\Pr[\mathsf{NameGame}(\mathcal{A}, q) = 1] - \Pr[\mathsf{NameGame}(\mathcal{A}, q) = 0]| \leq \epsilon(\lambda)$$

for security parameter λ where $\mathcal{A} = \mathcal{A}^C$.

Similarly, $q(\cdot, \cdot)$ is secure against leakage and identification attacks if for any PPT $\mathcal{A}$ it holds that

$$|\Pr[\mathsf{INDNameGame}(\mathcal{A}, q) = 1] - \Pr[\mathsf{INDNameGame}(\mathcal{A}, q) = 0]| \leq \epsilon(\lambda)$$

for security parameter λ where $\mathcal{A} \in \{\mathcal{A}^I, \mathcal{A}^L\}$.

We now define weak and strong request privacy.

Definition 5.4. *A request has* weak privacy *if it is secure against leakage and identification attacks. A request has* strong privacy *if it has weak privacy and is secure against correlation attacks.*

Communication Privacy

If a request and response are both private, then the entire exchange is private. As before, there is a strong and weak form of communication privacy. We define both with respect to request and response privacy as follows.

[5] This variant corresponds to $\mathcal{A}$ that does not possess offline computation resources.

Definition 5.5. *A request and response have* weak communication privacy *if* **at least** *the request and response have weak privacy. Similarly, a request and response have* strong communication privacy *if* **both** *the request and response have strong privacy.*

5.1.2 Eavesdropping Adversaries

Eavesdropping is the weakest attack on privacy. An eavesdropping adversary might not be able to capture any packet at will and may not be able to observe all request and response pairs. It can, however, collect some traffic for offline analysis. In this section we show how this simple capability has strong implications on mitigations for the main privacy threats outlined in Section 5.1.1.

Data Generation Functions and Response Privacy

Privacy against leakage and correlation attacks for responses is closely related to the concepts of indistinguishable and chosen-ciphertext-attack (CCA) security [180]. We show this in Theorems 5.1 and 5.2.

Theorem 5.1. *Responses must be encrypted with IND-secure encryption to have weak privacy.*

Proof. We define $r(\cdot)$ in $\mathsf{LIMDataGame}$ to be an IND-secure encryption function over generated data. To show that responses are secure against leakage attacks, we must show that $\mathcal{A}$ only wins the $\mathsf{LIMDataGame}$ with probability negligible in λ. In this case, $\mathcal{A}$ generates and submits N^0 and N^1 to the challenger which generates a random bit b, and returns $D^b = r(N^b)$, as before. Now, consider an alternate IND-secure encryption function $h(\cdot)$ that encrypts N^0 or N^1 instead of data to which these names are bound. Since there is a one-to-one correspondence between names and data (i.e., N^0 and N^1 are bound to D^0 and D^1, respectively), encrypting D^0 or D^1, using $h(\cdot)$, is no different from directly encrypting N^0 or N^1, using $r(\cdot)$. Therefore, $\mathcal{A}$'s probability of successfully guessing b by examining D^b is no more than its probability of correctly guessing b by examining encrypted N^0 and N^1. The latter is bounded by the probability of $\mathcal{A}$ correctly guessing b in the IND-secure encryption game, which is negligible. $\square$

Theorem 5.2. *Responses must be encrypted with CCA-secure encryption to have strong privacy.*

Proof. We define $r(\cdot)$ in $\mathsf{DataGame}$ to be a CCA-secure encryption function over generated data. As in the proof of Theorem 5.1, names form a one-to-one correspondence with data. Thus, there is no difference between encrypting names or corresponding data using $r(\cdot)$. Also, if $r(\cdot)$ were to encrypt names, then $\mathsf{DataGame}$ would be identical to the standard CCA-secure encryption game. Therefore, since the probability of $\mathcal{A}$ winning such a game is negligible in λ, it follows that $\mathcal{A}$'s probability of winning $\mathsf{DataGame}$ is also negligible in λ. $\square$

Name Transformations & Request Privacy

Information leaked from the request gives $\mathcal{A}$ some advantage in winning $\mathsf{NameGame}$ and $\mathsf{INDNameGame}$. Therefore, to assess the capabilities of $\mathcal{A}$ we must capture information leaked by $q(\cdot,\cdot)$. To begin, observe that the vanilla CCN identity function $q(\cdot,\cdot)$ provides no privacy since the identity function does not change names in any way. $\mathcal{A}$'s probability of correctly matching the challenge name to one of its inputs is always 1.0.

Clearly, the identity function is a trivial transformation, so we explore alternatives. Transformation functions may operate on one or more name segments at a time. For example, consider the translation function in Algorithm 7. F is a cryptographic hash function, which transforms each segment after a given prefix segment number into its hash digest. We call this type of transformation is *structure preserving* since it does not change the hierarchy of

Algorithm 7 SuffixHash name transformation

```
Require: $N$, $i$, $F(\cdot)$
Ensure: $\bar{N}$
1: $\bar{N} = N[1:i]$, $l = |N|$
2: for $j = i+1 \to l$ do
3:     $\bar{N}[j] = F(N[j])$
4: return $\bar{N}$
```

name segments; it only changes each name segment value. A transformation is not structure preserving if it modifies hierarchical information in a name. For example, the transformation in Algorithm 8 is not structure preserving. This is because it replaces the suffix after index i with the hash of that suffix. Thus, any names that have more than $l > i$ segments will always be transformed into a name with exactly $(i+1)$ segments.

Algorithm 8 SuffixHashFlatten name transformation

```
Require: $N$, $i$, $F(\cdot)$
Ensure: $\bar{N}$
1: $\bar{N} = \phi$, $l = |N|$
2: for $j = 1 \to i$ do
3:     $\bar{N}$.Append($N[j]$)
4: $\bar{N}$.Append($F(N[i+1]||\ldots||N[l])$)
5: return $\bar{N}$
```

A transformation is *uniform* if, for every input, it produces output of the same length. This means that uniform structure preserving transformations transform each individual segment of a name to some fixed-length value. Conversely, a uniform non-structure-preserving transformation yields suffixes of equal length.

We now consider some guiding criteria for achieving different levels of request privacy.

Theorem 5.3. *Weak and strong request privacy transformations are uniform and non-structure-preserving.*

Proof. Let $q(\cdot,\cdot)$ be a transformation that is neither uniform nor structure-preserving. Moreover, assume it also enables weak and strong request privacy. Let N^0 and N^1 be two $\mathcal{A}$-chosen names with different numbers of segments. Upon receipt of N^0 and N^1 in NameGame, the challenger returns $\bar{N}^b = q(N^b, i^*)$. Since $|N^0| \neq |N^1|$, it is trivial for $\mathcal{A}$ to determine b since either $|\bar{N}^b| = |N^0|$ or $|\bar{N}^b| = |N^1|$. This contradicts the assumption that $q(\cdot, \cdot)$ provides strong and weak privacy.

Now consider N^0 and N^1 that have the same number of name segments but, for at least one segment i, $|N^0[i]| \neq |N^1[i]|$. Upon receipt of N^0 and N^1 in NameGame, the challenger returns $\bar{N}^b = q(N^b, i^*)$. To determine b, $\mathcal{A}$ looks for the segment i where $|\bar{N}^b[i]| = |N^0[i]|$ and $|\bar{N}^b[i]| \neq |N^1[i]|$ (or vice versa). This exists since $q(\cdot, \cdot)$ is not a uniform transformation. Therefore, $q(\cdot, \cdot)$ does not provide strong or weak privacy since $\mathcal{A}$ can always win NameGame.

$\square$

We conclude that structure-preserving transformations offer neither weak nor strong request privacy. Taking this into consideration, there are at least two types of cryptographic primitives we can use to build transformations suitable for request privacy: hash and encryption functions. Hash functions are uniform by definition. However, encryption functions are not necessarily uniform. Therefore, encryption-based transformations must involve some form of padding to ensure uniformity.

Consider the SuffixHashFlatten transformation in Algorithm 8. This function replaces the suffix of the input name with its hash. In practice, however, using F does not yield request privacy. Since F is a publicly computable function, the unpredictability (entropy) of its output is directly related to the entropy of its input.

Let $x \in X$ denote an element in the support of a random variable X. Traditional Shannon entropy H of X is defined as

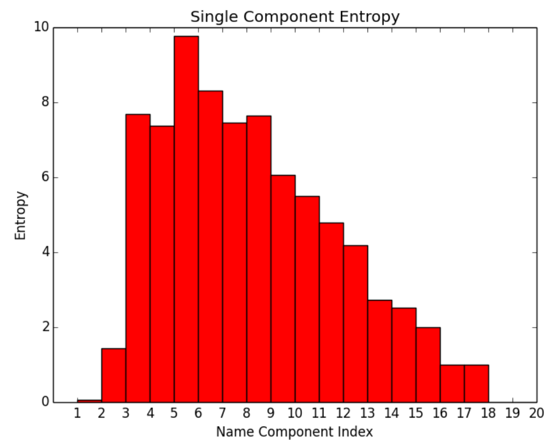
$$H(X) = -\sum_{x \in X} P(X = x) \log(P(X = x)).$$

The chain rule is useful to quantify entropy lost when new information is presented. Formally, the entropy of X, if conditioned on the entropy of Y, can be decreased by at most the latter, i.e., $H(X|Y) = H(X,Y) - H(Y)$, where $H(X,Y)$ is the joint entropy of X and Y defined as

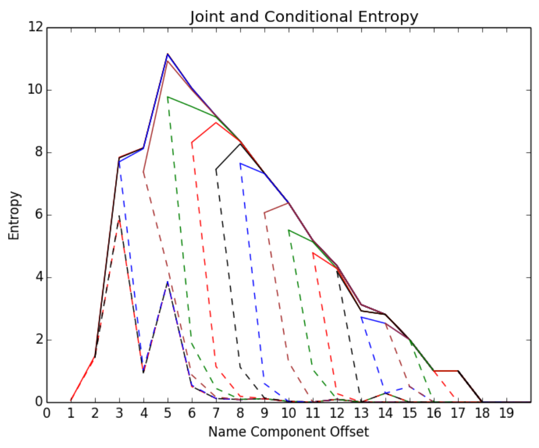
$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} \Pr(X = x, Y = y) \log(\Pr(X = x, Y = y)).$$

This formulation can be generalized to support computing the conditional and joint entropy of an arbitrary number of random variables. With it, we can quantify the entropy of names if we treat individual name segments as Discrete Random Variables (DRVs) and an entire name as a sequence of DRVs. For example, names of k segments can be viewed as a sequence of k DRVs. By using a large sample set of names, we can compute the entropy of the i-th segment DRV as well as its conditional entropy based on all $j < i$ segment DRVs. As input data we used the Cisco URI dataset available from [11]. It consists of $13,549,122$ unique URIs. The average length of each URI is 57.4B with median 52B and standard deviation 33.182B. Across all URIs, the average number of segments is 6.67 with median length 6 and standard deviation 2.212 segments. Entropy of individual name segments, which are plotted in Figure 5.1a, show that single-segment entropy is skewed with a peak at the 5th name segment. When name segments are considered in unison the results are quite different. Figure 5.1b shows how conditional entropy significantly degrades as more name segments are considered. The implication is that the prefix of a name leaks a significant amount of information about its suffix. While this dataset may not be representative, we believe that results would be similar for larger and even more diverse datasets.

These results lead us to conclude that (parts of) content names have very little entropy and are highly predictable. This is intuitive since URIs are meant to be meaningful and therefore predictable. Thus, we need strong randomness guarantees for the name translation function. It must not leak any information about the input application names. Consequently, it must be at least an IND-secure encryption function. Or, more generally, it must be a (non



(a) Single segment entropy



(b) Joint and conditional entropy across multiple segments

Figure 5.1: Name segment entropy

length-preserving) cryptographic pseudorandom function (PRF). PRFs are derived from PRF families that take additional inputs, such as a random seed or key to produce a specific PRF. Someone with knowledge of this additional input can compute the output of the PRF for any input since these functions are deterministic. For weak request privacy, the PRF cannot be computed by $\mathcal{A}$. Otherwise, $\mathcal{A}$ could compute the PRF value without the oracle and easily win NameGame. This is captured in the following theorem.

Theorem 5.4. *A name transformation function must be a PRF to enable weak request privacy.*

Proof. If the transformation is a PRF then $\mathcal{A}$ cannot compute the transformation output without the oracle. Moreover, by the properties of the PRF, the outputs are independent and uniformly distributed across the range of the function. This means that $\mathcal{A}$'s probability of distinguishing the translation of N^0 from N^1 is negligibly small. $\square$

Responses must also be encrypted for request privacy. If responses are not weakly private then they may reveal information about data or application names. Consequently, a request could identify specific application data.

We now make two final remarks with respect to the translation function. First, it must be easily invertible producers can recover N from $\bar{N}$. Thus, ideally, it should be a cryptographic pseudorandom permutation (PRP), which is an invertible PRF. Second, PRFs and PRPs are length-preserving by definition. This means that they are *not uniform*, which violates our previous criteria. There are two ways to make the translation uniform: (1) compute the PRP of the cryptographic hash of N, i.e., $PRP(F(N))$, or (2) pad the input prior to translation. We discuss (2) in Section 5.1.4. For option (1), it is reasonable to expect the producer to pre-compute a hash table that maps $F(N)$ to N. This would allow the producer to determine the pre-image of $F(\cdot)$.

Strong Request Privacy

In practice, a PRF is insufficient for strong request privacy. Since PRFs are deterministic, an adversary with oracle access can check the challenger's response and always win. Ideally, the oracle's output should always be randomized and reveal nothing about the input. In practice, this means that the oracle must provide semantic security for its outputs [180]. Such an oracle will never, with overwhelming probability, produce the same output given the same input. This is captured in the following theorem.

Theorem 5.5. *Semantically secure encryption is necessary for strong request privacy.*

Proof. Let $q(\cdot,\cdot)$ be a transformation that is semantically secure [180]. Access to $\mathcal{O}_{q,i^*}(\cdot)$ therefore does not aid $\mathcal{A}$ in $\mathsf{NameGame}$. Moreover, since $q(\cdot,\cdot)$ behaves as a PRF, $\mathcal{A}$'s advantage in distinguishing $\bar{N}^0$ and $\bar{N}^1$ is bounded by $\epsilon(\lambda)$. Thus, $q(\cdot,\cdot)$ provides strong request privacy. $\square$

5.1.3 On-Path Honest-but-Curious Adversaries

We now turn to on-path HbC adversaries that can observe requests and responses in transit between consumers and producers. Recall that communication privacy considers both the request and response of a data exchange. The composition of these elements determines the achievable degree of privacy. In Section 5.1.1, we showed that weak response privacy requires an IND-secure encryption scheme. Similarly, in Section 5.1.1, we proved that weak request privacy requires at least a name translation function using a keyed PRF. For strong privacy, requests and responses require a semantically secure (CCA-secure) encryption scheme. These composition rules are reflected in the following corollaries.

Corrolary 5.1. *A request-response pair have weak communication privacy if the latter is protected by an IND-secure encryption scheme* **and** *the former is transformed by a keyed PRF.*

Corrolary 5.2. *A request-response pair has strong communication privacy if the former is transformed using a CCA-secure encryption scheme* **and** *the latter is protected using a similar scheme.*

Note that our definitions do not depend on the exact capabilities of $\mathcal{A}$. For example, consider the strongest adversary that controls *every* router on the path between multiple consumers and a single producer. If requests and responses are strongly private, $\mathcal{A}$ cannot succeed in a correlation, identification, or leakage attack. $\mathcal{A}$ can only link the communicating parties. As discussed in Section 5.1.1, this is more of an issue of anonymity than privacy.

5.1.4 Privacy in Practice

We now discuss application design patterns that can achieve privacy variants discussed above. However, since we can not view requests and responses in isolation, we only focus on design patterns for communication privacy. This is because most realistic eavesdropping adversary would be capable of observing bi-directional traffic, i.e., both requests and responses.

Assumptions

As discussed earlier, a PRF or semantically secure encryption scheme is needed for request privacy. Hence, the consumer must have some information about the producer before issuing the request. Clearly, a request transformation function is only useful if the producer can *efficiently* compute its inverse. Also, we assume that the name transformation index, i.e., the minimal routable prefix, is known to both consumer and producer.

Weak Privacy

Recall that weak privacy allows correlation and no identification or leakage. This degree of privacy is unsuitable for highly sensitive data exchanges, such as online banking or e-commerce transactions. However, it might be suitable for less sensitive applications, e.g., content distribution networks (CDNs) which distribute *static* content. Concrete examples include Netflix, Spotify, Imgur and Flickr. Applications can use CDNs by asking for content with the same name. As long as a name does not reveal any information about the corresponding content, the CDN does not need to know to what data it refers. A CDN node maps a request to a response based on exact name match. We believe this general model is a good choice for weak communication privacy, wherein requests are protected by a keyed PRF and responses by an IND-secure encryption scheme.

To support this type of privacy, there are two cases in terms of the producer and consumer(s) relationship. In the first one, they share a secret, such as a previously established key. Let k be a unique key derived, e.g., via a PRF, from this shared secret. We could then instantiate $q(\cdot, \cdot)$ and $r(\cdot, \cdot)$ as an IND-secure encryption scheme based on a PRP indexed by k.

Now suppose the producer and consumer have no pre-shared secrets. Then, before requesting content from a producer, a consumer must know the former's public key. In this case, $q(\cdot, \cdot)$ can be any IND-secure (and thus CPA-secure) public key encryption function, e.g., RSA. The response must be likewise protected using IND-secure encryption. One way is to encrypt it using the consumer's public key, which the consumer could include in the request. Alternatively, the consumer could pick a one-time symmetric key and also include it in the original request. Of course, this does not provide forward secrecy, which is another aspect to consider.

This is clearly an inefficient solution since it effectively removes the utility of router caches, which is the primary reason one would choose weak privacy. A better approach would be to encrypt the content under a broadcast encryption scheme, e.g., [67]. Such schemes are CCA-secure and therefore suitable for our setting since IND-secure encryption follows from CCA-security. However, they only work if the consumer already has the decryption keys, which violates our assumption that consumers and producers have no pre-shared secrets. Therefore, it is unclear how to enable efficient weak privacy with caching for *public content*, i.e., content that can be requested by anyone without authentication or authorization.

Name Padding

Since IND-secure encryption is required for weak privacy, we must also address the issue of name padding to make these transformations uniform. Theorem 5.3 states that every encrypted name in a given namespace must be of the same length. This is only possible if all names are padded to some maximal length. The current CCN packet format limits the total name length to 64KB [225]. In practice, names are much smaller. To verify this, we analyzed names in the Unibas dataset from The Content Name Collection [11], which contains *unique* URLs submitted by users to URL shortener websites. We converted these URLs into a CCN-compatible name format. For example, the URL `http://www.domain.com/file.html` is converted into `/com/domain/file.html`. Table 5.2 shows some characteristics of the Unibas dataset. The standard deviation of the number of segments in a name is 8.14 and the mean length per segment is 10.39B. Even with these smaller sizes, performing padding universally across all namespaces would result in significant overhead: the size of an encrypted name would be about 800KB (based on the maximum segment length, which is well beyond the maximum threshold for CCN packets). For links with small(er) MTUs, e.g., in the range $[1,500\text{B}, 9,000\text{B}]$, the performance impact would be very heavy since it would almost always induce fragmentation. Even with secure fragmentation schemes such as those in [139] and [230], the overhead would be non-negligible.

Fortunately, names are not distributed uniformly. Table 5.3 illustrates the name distribution per number of segments in each name. It shows the number of names in the dataset that contain n segments for $n \in [1, 20]$. Note that: (1) almost 30% of names have 5 segments, and (2) names of up to 20 segments account for 99.876% of the dataset. Thus, maximum padding size is much smaller than the maximum name size.

Furthermore, padding could be applied on a per-namespace basis. For example, in namespace `/`, the maximum padding length is around 800KB. However, under the namespace `/netflix/`, the maximum name length is likely much smaller. Since an application has complete control over its namespace, it can specify a maximum length for consumers to use in padding.

As a final note, this requires i^* to be as long as the minimal routable (unencrypted) prefix needed to forward encrypted requests to the producer. For example, suppose that all interests in the `/netflix/` namespace were routable. We would need to set $i^* = 1$, since anything beyond that would leak information about the request. This highlights the relationship between namespace ownership, routing, and privacy.

Table 5.2: Unibas dataset characteristics.

Names		**Name segments**		**Segments per Name**	
Number of names	$870,896,633$	Total number of segments	$4,855,203,042$	Total number of segments	$4,855,203,042$
Average name length (bytes)	57.95	Average segment length (bytes)	10.39	Average segments per name	5.57
Name length standard deviation	77.60	Segment length standard deviation	30.02	Segments per name standard deviation	8.14
Minimum name length (bytes)	1	Minimum segments length (bytes)	1	Minimum segments per name	1
Maximum name length (bytes)	**$764,867$**	Maximum segments length (bytes)	**$764,867$**	Maximum segments per name	$210,658$

Strong Privacy

Strong privacy is suitable for applications for which security and privacy are more important than caching. Examples include: banking, e-commerce, and voting. Fortunately, in such cases, there are fewer design decisions, since both requests and responses require semantic or CCA-secure encryption. This immediately implies the need for a session-based protocol in which *no* request and response can be correlated. To the best of our knowledge, there is only one such protocol for CCN: CCNx Key Exchange (CCNxKE) [228]. Assuming only knowledge of the minimal routable prefix for a given of content, CCNxKE allows consumers and producers to create a secure session with forward-secure keys to encrypt both requests and responses using a CCA-secure encryption scheme. Clearly, strong privacy destroys any benefits of shared caching for consumers. We believe this is an important takeaway from our work.

Table 5.3: Name distribution per # of segments.

Segment count n	**Number of names**	**Percentage**
1	$13,952$	0.002%
2	$141,904$	0.016%
3	$71,327,647$	8.190%
4	$187,307,048$	21.507%
5	$253,852,565$	29.148%
6	$144,130,578$	16.550%
7	$93,837,904$	10.775%
8	$70,875,144$	8.138%
9	$25,611,959$	2.941%
10	$10,464,092$	1.202%
11	$3,973,961$	0.456%
12	$4,546,842$	0.522%
13	$1,206,905$	0.139%
14	$835,124$	0.096%
15	$844,552$	0.097%
16	$195,491$	0.022%
17	$121,486$	0.014%
18	$317,628$	0.036%
19	$168,228$	0.019%
20	$50,742$	0.006%
Total	$\mathbf{869,823,752}$	$\mathbf{99.876}\%$

5.1.5 Privacy and Auxiliary Information

Weak request and response privacy is attainable only up to a certain extent. If $\mathcal{A}$ has additional information about requests or responses, it can gain an advantage in weak privacy games. This is because such games permit correlation, which leaks information about content. Suppose that $\mathcal{A}$ has some *a priori* information about the popularity of some content. $\mathcal{A}$ can use *frequency* of requested content and *popularity* of known content to gain a non-negligible advantage in winning the game.

In more detail, consider request privacy in the presence of auxiliary information. We assume that $\mathcal{A}$ can eavesdrop on links and thus learn requests and responses. Let $\mathbb{L}$ be the set of links that $\mathcal{A}$ controls and let $\mathsf{Peek}(\mathbb{L})$ be a function that returns a packet from one of those links.[6] Let $\mathbb{Q}(N)$ and $\mathbb{R}(N)$ be sets of requests and responses, respectively, for content with name prefix N. We model mappings from $\mathbb{Q}(N)$ to $\mathbb{R}(N)$ as a weighted and directed bipartite graph with edges from $\mathbb{Q}(N)$ to $\mathbb{R}(N)$. An edge (Q, R) with weight l, where $Q \in \mathbb{Q}(N)$ and $R \in \mathbb{R}(N)$, means that request Q returns response R with probability $l/|\mathbb{R}(N)|$. In effect, weight indicates popularity of a given request-response pair.

In this model, request privacy is only possible if the in-degree (deg^-) for every R and the out-degree (deg^+) for every Q are equal **and** the weights of every edge are equal. This means that auxiliary information (frequency and popularity) does not help $\mathcal{A}$ distinguish between two different responses R and R' since each is equally likely to be the output from a randomly chosen request. This leads to the following observation.

Theorem 5.6. *Weak request privacy requires that:*

1. $deg^+(Q_i) = deg^+(Q_j)$ *for all* $Q_i, Q_j \in \mathbb{Q}(N)$.
2. $deg^-(R_i) = deg^-(R_j)$ *for all* $R_i, R_j \in \mathbb{R}(N)$.
3. *The distribution of edges from* $\mathbb{Q}(N)$ *to* $\mathbb{R}(N)$ *is uniform.*
4. *Every edge from* $\mathbb{Q}(N)$ *to* $\mathbb{R}(N)$ *has equal weight.*[7]

Proof. Assume $\mathcal{A}$ has additional information about $\mathbb{R}(N)$, e.g., that the edge distribution from $\mathbb{Q}(N)$ to $\mathbb{R}(N)$ is *not uniform*. We need to show that $\mathcal{A}$ can build a distinguisher $\mathcal{D}$ that wins $\mathsf{NameGame}$ with probability greater than $\epsilon(\lambda)$. Let $k = max\{1, \lceil |\mathbb{Q}(N)|/|\mathbb{R}(N)| \rceil\}$, be the expected in-degree for each response $R \in \mathbb{R}(N)$. That is, for a uniform edge distribution, k is the number of requests that map to each response. Let $\mathcal{D}$ be a distinguisher created using $\mathsf{Peek}(\mathbb{L})$ and given to $\mathcal{A}$ in $\mathsf{NameGame}$. $\mathcal{D}$ observes the network using $\mathsf{Peek}(\mathbb{L})$ to determine the frequency of individual *requests*. $\mathcal{A}$ runs $\mathcal{D}$ for a polynomial amount of time in order

[6]Note that this *is not* the same as $\mathcal{A}$ accessing either oracle in request or response privacy games. Also, $\mathcal{A}$ can only peek on unencrypted links.

[7]If the weight were not uniform then some responses would be more popular than others. This is what we exploit when constructing a distinguisher in the proof.

to collect this frequency information. Then, $\mathcal{A}$ samples two requests (names) Q_0 and Q_1 with the *lowest* and *highest* probabilities, respectively, from its a priori known distribution. Specifically, if $\Pr[Q_i]$ is the popularity of a given name (query) Q_i, then Q_0 and Q_1 are chosen as:

$$Q_0 = \arg\min_{Q_i} \Pr[Q_i]$$
$$Q_1 = \arg\max_{Q_i} \Pr[Q_i]$$

$\mathcal{A}$ then provides N_0 and N_1 – the names of Q_0 and Q_1 – to the challenger, which responds with N_b. Upon receipt, $\mathcal{A}$ queries $\mathcal{D}$ with Q_b, obtained from N_b, to determine the relative frequency f of Q_b based on the collected frequency statistics. If $f > k/\mathbb{R}(N)$, then $\mathcal{A}$ outputs $b' = 1$; otherwise, it outputs $b' = 0$.

The winning probability can be expressed as:

$$\Pr(b' = b) = \Pr(b = 1 \wedge b' = 1) + \Pr(b = 0 \wedge b' = 0)$$

Using Bayes' Theorem, this can be rewritten as:

$$\Pr(b = 1 \wedge b' = 1) + \Pr(b = 0 \wedge b' = 0) =$$
$$\Pr(b = 1) \Pr(b' = 1 | b = 1) + \Pr(b = 0) \Pr(b' = 0 | b = 0)$$

From the construction of $\mathcal{D}$, it follows that $\Pr(b' = 1|b = 1) = \Pr[Q_1]$ and $\Pr(b' = 0|b = 0) = \Pr[Q_0]$. Let p_1 and p_0 denote these respective probabilities. Also, since b is sampled at random, $\Pr(b = 1) = \Pr(b = 0) = 0.5$.

In order for $\mathcal{A}$ to win $\mathsf{INDNameGame}$, it must be true that $0.5p_0 + 0.5p_1 > k/\mathbb{D}(N) + \epsilon(\lambda)$. We can rewrite this as $p_0 + p_1 > 2k/\mathbb{D}(N) + \epsilon(\lambda)$, where p_0 and p_1 are the minimum and maximum probabilities in the popularity distribution. Clearly, there are distributions where this inequality holds for any k. One example is the Poisson distribution. $\square$

To summarize, weak request privacy is difficult to achieve, since some requests are always more popular than others. Moreover, since content popularity is not controllable by the producer, it cannot expect to enforce uniform popularity.

Implications The frequency analysis attack succeeds because responses are not generated with equal probabilities. An intuitive way to mitigate the attack is to ensure that every response is unique. One way to realize this is by requiring a producer to encrypt each response separately and uniquely. However, this immediately obviates the main benefit of caching.

An alternative is for a router to individually re-encrypt cached content before passing it downstream. (Thus, it would be infeasible to correlate content coming into a router with another content later leaving the same router.) Unfortunately, this seems to be possible only via so-called *proxy re-encryption* or re-randomizable encryption techniques which are typically based on public key schemes, e.g., ElGamal. More importantly, these techniques would require *the entire content* to be re-randomized, which is likely to be prohibitively expensive. Specifically, this would rule out the use of traditional (and efficient) hybrid encryption. Therefore, bulk content encryption, re-encryption and eventual decryption would all have to be performed using purely public key cryptography.

Even if the above were not an issue, there would remain a problem due to requests for various names not being uniformly distributed. One counter-measure is to encrypt each request using a CPA-secure encryption scheme. Then, two requests for the same content would carry different names that cannot be correlated. This would, once again, negate the main benefit of caching. Thus, in the presence of $\mathcal{A}$ that has auxiliary information, weak privacy only seems possible if both requests and responses are individually encrypted, making router caching useless, except for re-transmissions.

5.2 Static Content and Frequency Analysis Attacks

In the previous section we showed that, if the adversary has additional auxiliary, information about the requested content such as its popularity within a given namespace, it can recover the content name even if PRF-transformed names are used. The cause is interest linkability, i.e., ability to determine when two interests refer to the same content. We consider it a privacy leakage when the adversary can learn any information about underlying interests based on their PRF-transformed names.

This type of leakage is not unique to CCN. If we consider CCN as a generic key-value store where PRF-transformed interests are keys, and corresponding content objects are values, the problem at hand is analogous to privacy leakage in encrypted databases. This topic has been extensively studied in recent years [237]. In this section, we apply to CCN attacks from the research literature on privacy of encrypted databases. Specifically, we study adversarial ability to learn request and response plaintext using information learned from eavesdropping on encrypted traffic. In doing so, we try to answer the following questions:

- How does the accuracy of adversary's auxiliary information influence effectiveness of privacy attacks?
- How does router caching affect attacks, and how does it relate to topological distribution of the adversary?
- Can replicating or partitioning content among multiple producers decrease effectiveness of these attacks, and if so, to what degree?

This section makes the following contributions[8]:

- Given accurate auxiliary information about content popularity distribution and sufficiently many request samples, the adversary can, with very high probability, correctly map *some* encrypted content packets to their plaintext counterparts.
- Privacy attacks are hindered by router caching and content replication since they effectively reduce the sample size for an adversary.
- Knowledge of a namespace is sufficient for the adversary to learn the perceived popularity of content published under that namespace. This has strong implications on how much of a namespace should be public or otherwise discoverable by anyone, especially, when content names correspond to private data.

5.2.1 Threat Model

This section describes the attack on privacy and all relevant notation. We also present the assumed adversary model and auxiliary information.

Notation

Let $\mathcal{D}(\mathbb{U})$ be a probability distribution over some universe of elements $\mathbb{U}$. When it can be inferred from context, we omit $\mathbb{U}$ from $\mathcal{D}(\mathbb{U})$. Let X denote a random variable for a distribution over $\mathbb{U}$. When X is discrete, $f_X(x)$ is the corresponding probability mass function (PMF). For simplicity, we also use $\mathcal{D}(x)$ to denote $f_X(x)$. Given any two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over the same finite domain $\mathbb{U}$, their statistical distance is computed as:

$$\Delta(\mathcal{D}_1, \mathcal{D}_2) \triangleq \sup_{x \in \mathbb{U}} |\mathcal{D}_1(x) - \mathcal{D}_2(x)| = \frac{1}{2} \sum_{x \in \mathbb{U}} |\mathcal{D}_1(x) - \mathcal{D}_2(x)|$$

[8]These are also discussed by Ghali et al. [148].

(This is equivalent to the well-known Kolmogorov-Smirnov statistic.) Frequency distribution $\mathcal{F}_{\mathbb{U}}(n, x)$ represents the number of times $x \in \mathbb{U}$ occurs after taking n samples from $\mathcal{D}(\mathbb{U})$. For any $n > 0$, let r_i denote the i-th largest value of $\mathcal{F}_{\mathbb{U}}(n, x)|_{x \in \mathbb{U}}$, where $r_1 = \max_{x \in \mathbb{U}}\{\mathcal{F}_{\mathbb{U}}(n, x)\}$ and $r_{|\mathbb{U}|} = \min_{x \in \mathbb{U}}\{\mathcal{F}_{\mathbb{U}}(n, x)\}$.

Privacy Primer

As previously discussed, weak privacy is preferable in order to retain most benefits of CCN. However, it is subject to so-called frequency analysis attacks. If $\mathcal{A}$ has auxiliary information about underlying content [146]. Such information can be extracted from a variety of sources, including the target application, e.g., Netflix or Spotify, publicly-available statistics, e.g., income statistics for a certain demographic or public financial records, or prior versions of content, e.g., a history of Netflix or Spotify media catalogs. In these attacks, $\mathcal{A}$ uses its knowledge about content popularity, along with observed interests, to infer which interest corresponds to which content. As mentioned above, this is similar to attacks on encrypted databases. In that model, $\mathcal{A}$ corrupts a server storing an encrypted database and observes database queries. $\mathcal{A}$'s goal is to determine the plaintext value of each encrypted record based on auxiliary information and observed queries [237].

The attack scenario in the database scenario does not map directly to the frequency analysis attack outlined in [146]. In the former, once $\mathcal{A}$ compromises the target server, it can observe all queries. In contrast, in CCN, $\mathcal{A}$ is a collection of one or more compromised routers that observe network traffic. Therefore, by compromising a single router, $\mathcal{A}$ does not automatically get access to all queries for the target content. This is a critical fact in CCN: distributed nature of the network (particularly, existence of router caches) means that $\mathcal{A}$ has far less information than in the database scenario. In this work, we explore how this gap affects $\mathcal{A}$'s success in attacking CCN privacy.

Adversarial Model

We assume that $\mathcal{A}$ is a distributed and active adversary that aims to learn information about statically encrypted, or weakly private, content shared among multiple consumers. Encryption is not ephemeral, i.e., packets are not encrypted in transit between producers and consumers. Thus, we assume that $\mathcal{A}$ can correlate interest and (encrypted) content packets referring to the same application data. $\mathcal{A}$ can compromise a subset of routers on the path between arbitrary consumers and the nearest copy of the requested content. Once a router R is compromised, $\mathcal{A}$ can observe all packets that R processes. $\mathcal{A}$ can also spawn malicious consumers that probe the network for content and can populate non-compromised routers' caches with copies of that content. However, $\mathcal{A}$ can not compromise either consumers that issue interests for content or producers that respond to interests with encrypted content packets. A realistic example of $\mathcal{A}$ could be a state-sponsored entity or a set of colluding Internet Service Providers (ISPs).

Adversarial Information

Let $\mathbb{P}$ be a set of application data items, and let $\mathbb{C}$ be the set of encrypted content packets used to carry items in $\mathbb{P}$ through the network. That is, for each $p \in \mathbb{P}$, there is an encrypted form in $\mathbb{C}$. Let $T : \mathbb{C} \rightarrow \mathbb{P}$ be the *truth mapping* from the encrypted content to their plaintext data counterparts. We assume that all consumers issue interests for data in $\mathbb{P}$ according to some *real popularity distribution* $\mathcal{D}_R(\mathbb{P})$.

$\mathcal{A}$ is given access to some fixed auxiliary information about this popularity distribution, called $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$. Moreover, at any time t, $\mathcal{A}$ has access to a *snapshot frequency distribution*, denoted $\mathcal{F}^{\mathcal{A}} : \mathbb{T} \times \mathbb{C} \rightarrow \mathbb{N}$. That is, for each item $c \in \mathbb{C}$, $\mathcal{F}^{\mathcal{A}}(t, c)$ is the number of times c was observed by $\mathcal{A}$ up t. The set of items from $\mathbb{C}$ observed by $\mathcal{A}$ at time t is $\mathbb{O}(t)$. Using $\mathcal{F}^{\mathcal{A}}$, $\mathcal{A}$ can create an *empirical distribution* $\mathcal{D}_E(\mathbb{C})$ of the popularity of each observed item.

If $\mathcal{A}$ is a global adversary, then $\mathcal{D}_E(\mathbb{C})$ approximates $\mathcal{D}_R(\mathbb{P})$ under the truth mapping T. Essentially, $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$ is $\mathcal{A}$'s approximation of $\mathcal{D}_R(\mathbb{P})$.

5.2.2 Frequency Analysis Attack Overview

We now describe the frequency analysis attack adapted from the encrypted databases scenario [237]. In our setting, encrypted or otherwise obfuscated interests are analogous to queries for encrypted database records. The entire network, comprised of caches and content, is equivalent to one giant database. The adversary can eavesdrop on all (or parts of) the network (database) and, as a result, can view all (or some) interests and content (queries and records). This access, along with auxiliary information about popularity of content (records), is sufficient to perform the attack. More concretely, the core idea is as follows:

$\mathcal{A}$ learns (or is given) some auxiliary information about popularity distribution of application names, or content, i.e., $\mathbb{P}$. $\mathcal{A}$ also observes empirical popularity distribution of encrypted interests for encrypted content, i.e., $\mathbb{C}$. In doing so, $\mathcal{A}$ seeks to learn T, i.e., which items in $\mathbb{C}$ map to items in $\mathbb{P}$. $\mathcal{A}$ succeeds if it learns any of these with non-negligible success probability.

In a frequency analysis attack at time t, $\mathcal{A}$ combines $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$ and $\mathcal{F}^{\mathcal{A}}(t, c)$, as follows: First, $\mathcal{A}$ ranks items in $\mathcal{F}^{\mathcal{A}}(t, c)$ in order of descending popularity. Then, for each $c_i \in \mathbb{O}(t)$ in decreasing order according to $\mathcal{F}^{\mathcal{A}}(t, c)$, $\mathcal{A}$ guesses that c_i corresponds to the i-th most popular item p_j based on $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$.

Let sort be a function that sorts a histogram in descending order of frequency. Algorithmically, the attack works as follows:

1. $\rho \leftarrow \mathsf{sort}(Hist(\mathbb{O}(t)))$

2. $\pi \leftarrow \mathsf{sort}(Hist(\mathbb{P}))$

3. Compute a mapping $\alpha : \mathbb{C} \to \mathbb{P}$ such that, for all $c \in \mathbb{C}$:

$$\alpha(c) = \begin{cases} \pi[Rank_\rho(c)] & \text{if } c \in \mathbb{O}(t) \\ \perp & \text{if } c \notin \mathbb{O}(t) \end{cases}$$

The result of the attack is α, the guessed truth mapping from encrypted data items to their plaintext form.

Accuracy of the attack is defined as follows: Let $R(\alpha, T)$ be a function that counts the number of $\mathcal{A}$'s correct guesses. A correct guess is such that $\alpha(c) = T(c)$. $R(\cdot)$ computes the *total* number of guesses by $\mathcal{A}$. We say the *match ratio* is the total number of correct guesses divided by $|\mathbb{P}|$. By itself, the match ratio may be misleading, e.g., if the dataset is large and has a long tail with items that have near-equal popularities. Thus, we are also interested in *partial accuracy* of the attack. We define partial accuracy as a function $S(\cdot)$ that takes an index $i \leq |\rho|$ along with ρ, α, and T and computes:

$$S(i, \rho, \alpha, T) = \sum_{j=1}^{i} \frac{\sum_{k=1}^{j} \mathsf{Match}(j, \rho, \alpha, T)}{|\rho|},$$

where:

$$\mathsf{Match}(i, \rho, \alpha, T) = \begin{cases} 1 & \text{if } \alpha(\rho[i]) = T(\rho[i]) \\ 0 & \text{if } \alpha(\rho[i]) \neq T(\rho[i]) \end{cases}$$

Intuitively, at index i, this computes fraction of guesses that are correct up to i. For example, it is likely that $S(1, \rho, \alpha, T) \approx 1.0$ if $\mathcal{A}$'s auxiliary information and observances are accurate. In some cases, the total accuracy of the attack might not matter, while it might be important if $\mathcal{A}$ can correctly guess only a small number of items with a very high probability.

5.2.3 Simulation Methodology

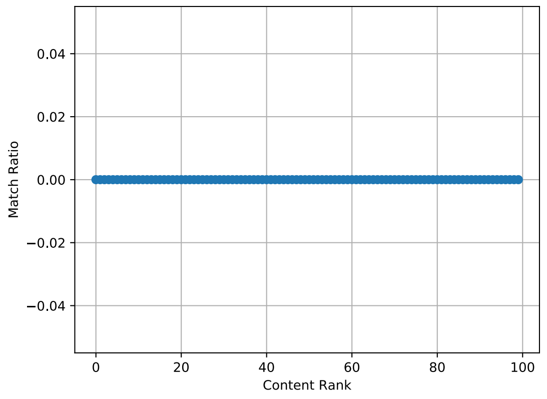
We now describe the simulator for evaluating the frequency analysis attack. Its source code is available online at [329].

Content Distributions

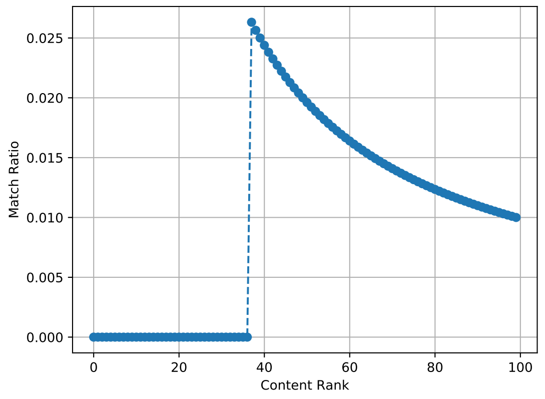
To assess the attack we need realistic information about popularity distributions. Since there are no real-world deployments of CCN (or other ICN architectures), we must rely on information from current web content traces. Fortunately, there has been a great deal of work studying the popularity of web content. Breslau et al. show in [69] that web content does not follow a strict Zipf distribution, as often suggested. Instead, it adheres to a Zipf-like distribution where the i-th most popular page is requested with probability proportional to $i^{-\alpha}$, where $\alpha \in [0.6, 2.5]$ [51, 178, 270, 233, 198, 173, 99]. Thus, unless stated otherwise, we hereafter use the Zipf distribution to model real content popularity.

Simulator Overview

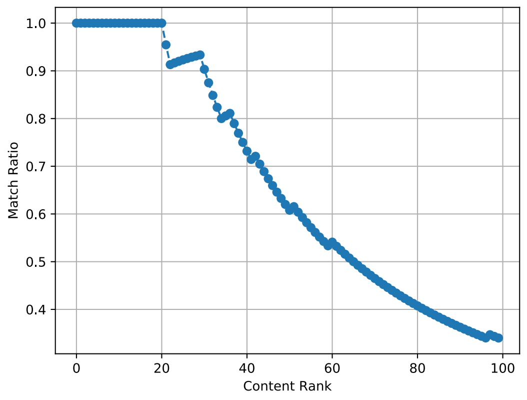
We implemented a custom CCN simulator for this study. We chose not to use available ccns3Sim or ndnSim because we do not need to take into account network behavior beneath CCN. The attack is sufficiently generic that we only need a way to control interest and contents. Our simulator allows a user to create an arbitrary network topology G composed of sets of: consumers $\mathbf{C}$, routers $\mathbf{R}$, and producer(s) $\mathbf{P}$. Once created, $\mathcal{A}$ is realized as a subset of compromised entities. Each router has a cache with probability p_c. We represent a network configuration by its topology graph $G = (\mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{R}_{\mathcal{A}})$, where $\mathbf{R}_{\mathcal{A}}$ represents routers



(a) Uniform popularity and auxiliary information



(b) Zipf popularity and uniform auxiliary information



(c) Zipf popularity and auxiliary information

Figure 5.2: Attack accuracy with varying auxiliary information and content popularity

controlled by $\mathcal{A}$. A network with cache probability[9] p_c is denoted as $\mathsf{Net}((\mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{R}_{\mathcal{A}}), p_c)$. The next step is to create the content universe $\mathbf{U}$. A probability (popularity) distribution is assigned to $\mathbf{U}$, denoted $D(\mathbf{U})$. Consumers sample their interests from this distribution.[10] We also allow auxiliary information distribution $\mathcal{D}_{A}^{\mathcal{A}}(\mathbb{P})$ to be imposed on the content universe. This distribution is given to $\mathcal{A}$.

[9]By cache probability we mean the probability that any node in the network employs a cache. The size of which is bounded to $10,000$ items. This size was chosen so that caches did not suffer constant churn and content was retained for longer than a single consumer-to-producer round trip.

[10]Currently, only Zipf and Uniform distributions are supported. However, it is easy to add, and experiment with, new distributions.

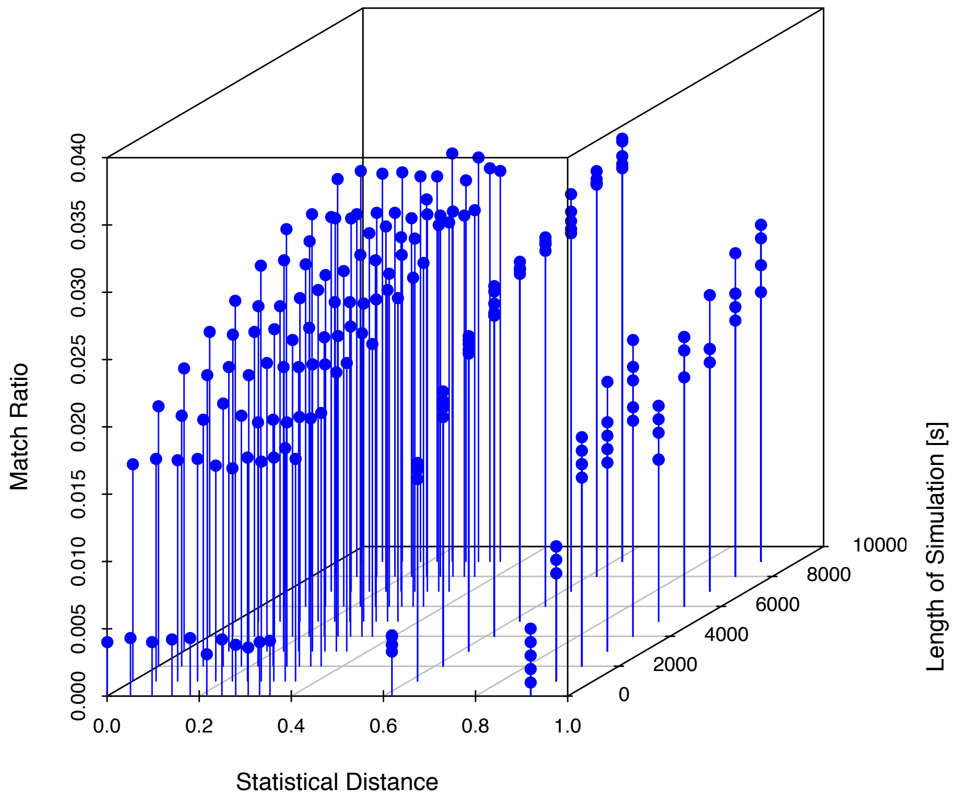


Figure 5.3: Attack accuracy as a function of $\Delta(\mathcal{D}_R, \mathcal{D}_A)$ and simulation time

Once the simulation is configured, it runs for a number of epochs. At each epoch, a random consumer $c \leftarrow_\$ \mathbf{C}$ sends an interest for content $C \leftarrow_\$ \mathbf{U}$, sampled according to $\mathcal{D}_R(\mathbb{P})$. An interest is forwarded until it: (1) results in a router cache hit, or (2) reaches the producer. Then, a content packet is sent back to the consumer. Each $\mathcal{A}$-controlled node records the interests it sees during this process. When the simulation completes, observed results from each $\mathcal{A}$-controlled node are merged to form $\mathcal{A}$'s complete view of the network. (Specifically, frequency histograms are merged together into one.) This is then fed into the frequency analysis attack along with true popularity distribution and auxiliary information. The output of the attack is the match ratio and $\mathcal{A}$'s accuracy, as described in Section 5.2.2

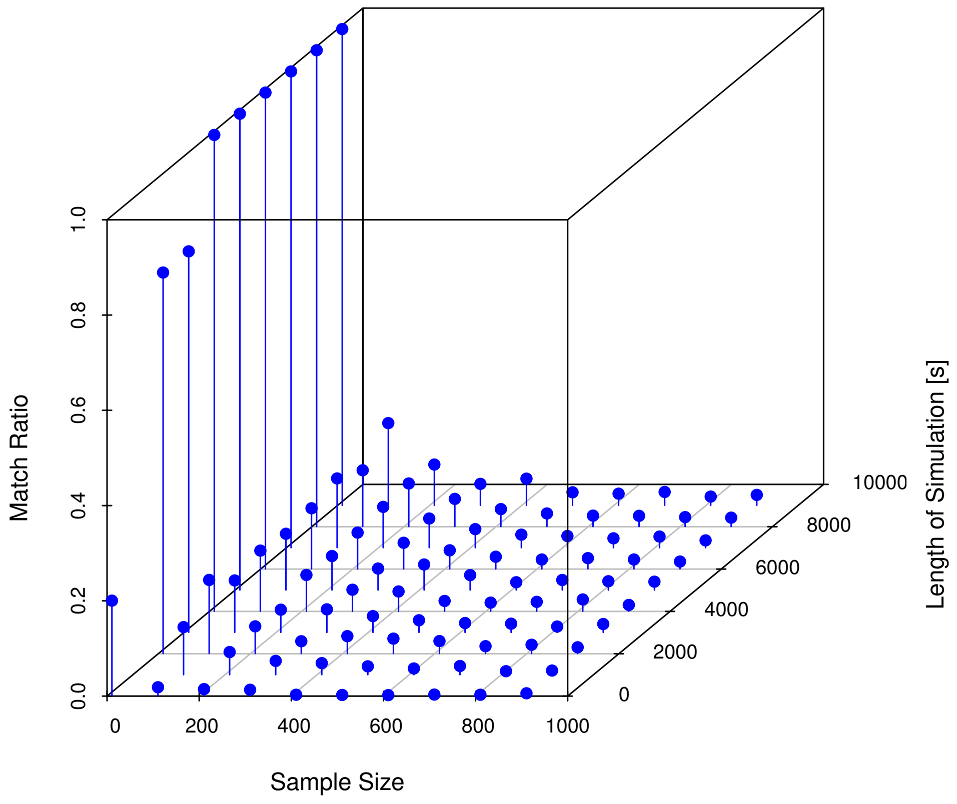


Figure 5.4: Attack accuracy as a function of content sample size and simulation time

5.2.4 Global Eavesdropping Adversaries

In this section we experimentally assess efficacy of the frequency analysis attack by a global $\mathcal{A}$, denoted by $\mathcal{A}_G$, which is assumed to have access to every interest issued by every consumer in the network. In this model, we need to answer the following question:

> Given content popularity distribution $\mathcal{D}_R$ and $\mathcal{A}_G$ with auxiliary information distribution $\mathcal{D}_A$, to what extent can $\mathcal{A}_G$ successfully correlate encrypted interest and content packets with their plaintext counterparts?

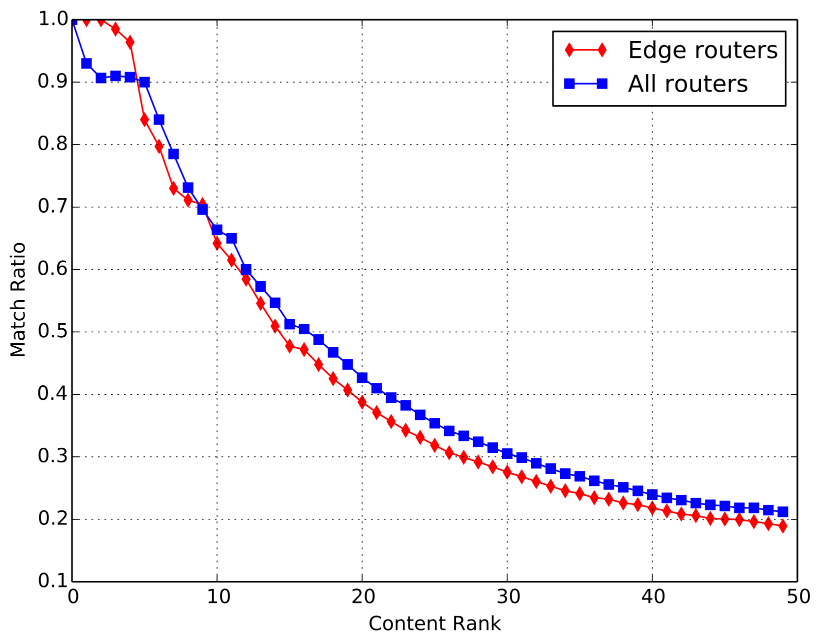


Figure 5.5: Match ratio for $\mathcal{A}$ distributed across edge and network routers

As mentioned in Section 5.2.2, we consider total and partial success by $\mathcal{A}_G$, since encryption protects every packet equally. We first assess attack accuracy with various $\mathcal{D}_R$ and $\mathcal{D}_A$. Results are shown in Figure 5.2. With the exception of simulation noise, accuracy is very low when either distribution is uniform. However, when $\mathcal{D}_A$ is statistically close to $\mathcal{D}_R$, attack accuracy is high.

Next, to understand the extent to which statistical distance affects this attack, we conducted the following experiment. First, we created a content universe $\mathbb{U}$ of size N. Then, for each considered probability distribution, we created $\mathcal{D}_R$ and $\mathcal{D}_A$ for $\mathbb{P}$. We considered uniform distribution as a baseline, i.e., the case of $\mathcal{A}_G$ having no auxiliary information, and Zipf distribution with parameters $\alpha \in [0.5, 2.5]$. We then ran the simulator for τ time steps. Finally, we simulated the frequency analysis attack, measured accuracy of resultant guesses, and computed $\Delta(\mathcal{D}_R, \mathcal{D}_A)$. Figure 5.3 shows the matching probability as a function of $\Delta(\mathcal{D}_R, \mathcal{D}_A)$, for various $\mathcal{D}_R$. It illustrates that, as $\Delta(\mathcal{D}_R, \mathcal{D}_A)$ increases, matching ratio decreases, as expected. However, the rate of decline is low, meaning that even some statistical equivalence is sufficient for the attack.

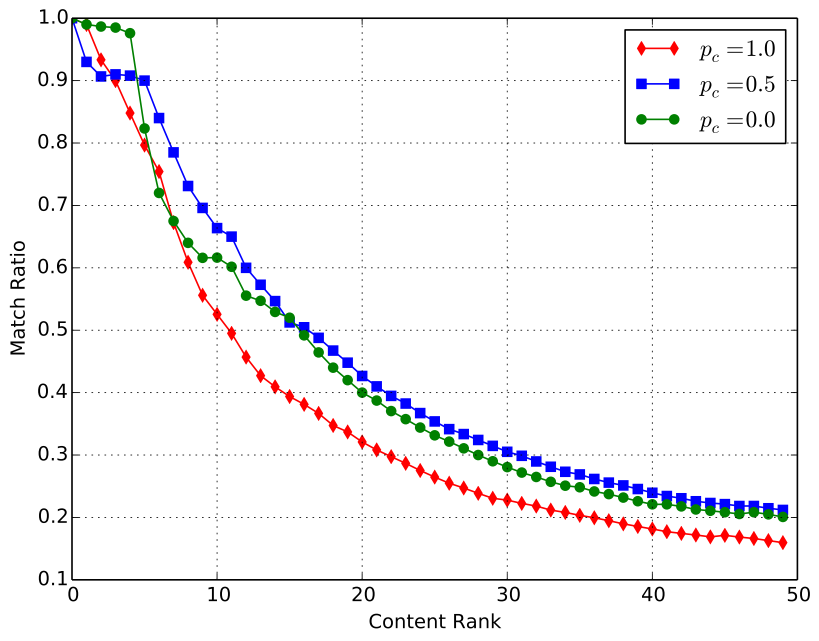


Figure 5.6: Attack accuracy with varying network caches

Size of content universe has a non-negligible effect on attack accuracy. Intuitively, with more options to choose from, $\mathcal{A}_G$'s task of building the correct mapping becomes more difficult. To show this, we repeated the same experiment as above except with fixed $\mathcal{D}_R$ and $\mathcal{D}_A$, while varying N. Figure 5.4 shows the results. As expected, as N increases, matching ratio quickly decreases. This is because each mapping entry becomes more sensitive as the probability space thins.

5.2.5 Distributed Eavesdropping Adversaries

Admittedly, $\mathcal{A}_G$ is not the most realistic adversary. In practice, adversaries will likely be localized in small groups of possibly collocated routers. For example, $\mathcal{A}$ could exploit software running in edge access points to observe traffic closest to consumers, or it could subvert an AS and compromise some or all of its routers. We now consider a distributed adversary, in order to assess the relationship between network caching, content location, and $\mathcal{A}$'s topological distribution. Each of these variables impacts the type and number of samples observed by $\mathcal{A}$, which are the main components of the attack. Intuitively, as quality of this information degrades, so should attack accuracy.

We conducted all experiments described below over a topology based on Deutsches ForschungsNetz (DFN) [5, 6]. It consists of 160 consumers, multiple producers attached to edge routers, and multiple routers (more than 30).

Adversary Distributions and Caching Effects

Attack accuracy increases as a function of $\mathcal{A}$'s coverage. As shown in the previous section, accuracy can be quite high if $\mathcal{A}$ can observe all traffic. However, as $\mathcal{A}$'s presence declines, so does the number of samples observed. We consider two $\mathcal{A}$ topological configurations: (1) distributed among some fraction of *edge* routers, and (2) distributed among a random fraction of *all* routers. To explore the impact of $\mathcal{A}$'s topological distribution, we conducted an attack experiment on $\mathsf{Net}((\mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{R}_{\mathcal{A}}), p_c)$, where $p_c = 0.5$ and $\mathbf{R}_{\mathcal{A}}$ is nearly 25% of edge routers or 25% of all routers. Results in Figure 5.5 show that, in the first edge case, $\mathcal{A}$ attains higher accuracy for high ranking content. This is because its knowledge of interest frequency is more complete, due to duplicate interests not being masked by caches. In the second case (among all routers), overall match ratio is higher than in the edge case since $\mathcal{A}$ has access to more traffic in the network.

Caching also plays an important role: if enabled in every router, there should be, in theory, less traffic traversing the network. Thus, $\mathcal{A}$ would *observe fewer* samples of encrypted content[11], and attack accuracy would necessarily decline. This is an interesting relationship explored in [23]. In some scenarios, caching can be easily exploited to violate privacy of individual consumers. However, with respect to content, caching complicates the attack.

To explore this relationship, we experimented with $\mathsf{Net}((\mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{R}_{\mathcal{A}}), p_c)$, where $p_c \in \{0.0, 0.5, 1.0\}$. Each router has a 0.25 probability of being compromised. Results in Figure 5.6 show that, when caching is disabled, $\mathcal{A}$ is correct (for high ranking content) with greater probability than if caching is globally enabled. This is a direct result of observing fewer samples.

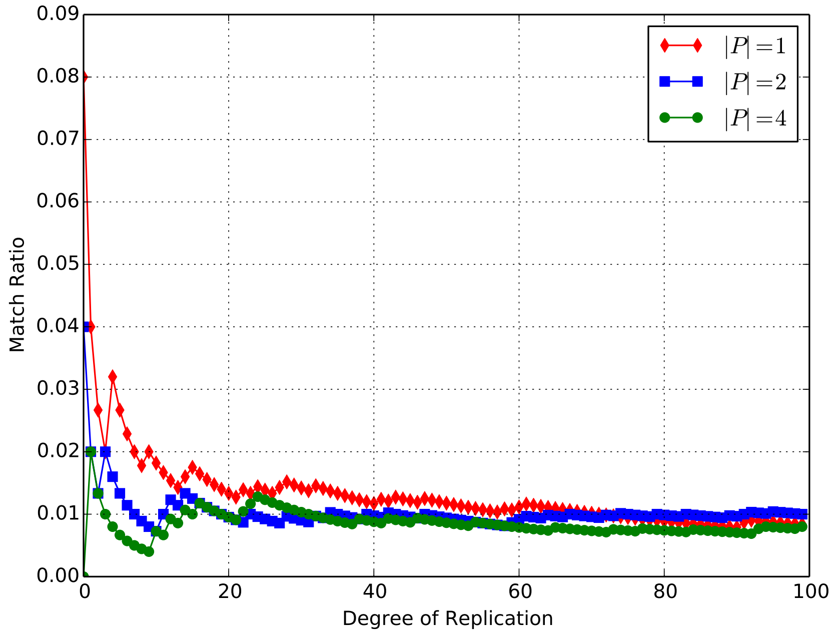


Figure 5.7: Attack accuracy with producer replication

Replication Effects

Replication, i.e., serving content from multiple network locations, is another important factor in attack efficacy. Similar to caching, replication allows interest and content packets to bypass $\mathcal{A}$ and cause it to observe less traffic. To understand the extent to which replication deters

[11] Assuming that $\mathcal{A}$ is not located at the edge.

the frequency analysis attack, we conducted experiments on $\mathsf{Net}((\mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{R}_{\mathcal{A}}), p_c)$, where $p_c = 0.5$ and $|\mathbf{P}| \in \{1, 2, 4\}$. A replicated producer publishes the same content under a different prefix[12] in a different part of the network. (We forced each replica to be sufficiently disjoint topologically in order to to reduce the chance of two interests for different replicas traversing the same path.) As shown in Figure 5.7, attack accuracy decreases as the degree of replication increases. With more replicas, consumers are free to stripe their interests across multiple prefixes and, by doing so, potentially bypass $\mathcal{A}$.

5.2.6 Probing Popularity Inference

The frequency analysis attack requires $\mathcal{D}_E(\mathbb{C})$ and $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$. The former is needed to approximate $\mathcal{D}_R(\mathbb{P})$. The latter is not always readily available by eavesdropping. Fortunately, by exploiting properties of CCN, it can be learned. Armed with knowledge of the public namespace for a set of content, i.e., network names $\mathbb{N}$, $\mathcal{A}$ (acting as a malicious consumer) can build $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$ by probing the network. (Recall that, in this work, network names are encrypted, and the adversary can not learn their contents. Thus, knowledge of the network name does not lead to knowledge of the corresponding application name.) Specifically, $\mathcal{A}$ can query for known names and, based on response time, infer whether corresponding content is cached. This sort of inference attack is similar to the invasive cache probe attack in [197], which works as follows: $\mathcal{A}$ aims to learn whether some nearby consumer asked for content named N. To do so, it requests N and relies on timing information to learn whether this content was served from a cache. (Response time lower than the consumer-to-producer RTT means that requested content was served from a cache.)

The rate at which it probes the network is important since $\mathcal{A}$ must be able to differentiate between cached copies that it itself injected into the network by probing, and actual cached

[12]For example, an Akamai replica might use the `/akamai`, while a Fastly replica might use the `/fastly` prefix.

copies previously requested by nearby consumers. Let t_c be the characteristic time of a cache [79]. If LRU cache eviction policy is used, then t_c is average time elapsed between the last request for an item and its eviction. If FIFO or random cache eviction policy is used, then t_c is average time between insertion and eviction. For simplicity, we assume that all routers use LRU. t_c tells $\mathcal{A}$ the expected time interval before an item is evicted. In turn, this can be used as a lower bound on probe frequency f_p, i.e., $f_p > 1/t_c$.

Assuming $\mathcal{A}$ knows t_c, the probing algorithm runs as follows. Let $\mathsf{AppendRandomComponent}(N, \lambda)$ be a function that, given input name N, samples a random bit-string of length λ and appends it to N as the last name segment. This effectively creates a unique name for which a cache hit is impossible (with overwhelming probability). Given name universe $\mathcal{N}$, $\mathcal{A}$ iterates through the set and, for each $N \in \mathcal{N}$, issues a pair of interests (probes): $N_h = N$ and $N_m = \mathsf{AppendRandomComponent}(N, 128)$, and records time t_N. It then waits to receive the corresponding content packets, named N_h and N_m, and records their respective arrival times t_N^h and t_N^m.[13] When both are received, $\mathcal{A}$ computes $\Delta_N = ||(t_N^h - t_N)| - |(t_N^m - t_N)||$. If $\Delta_N > \epsilon$ for some constant ϵ, $\mathcal{A}$ learns that an interest for N_h was satisfied faster than that for N_m, which went all the way to the producer. Therefore, a cache hit occurred. $\mathcal{A}$ then sleeps for t_c before proceeding to the next name in $\mathcal{N}$. This process is repeated to incrementally build $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$. This procedure is shown in Algorithm 9, where r is the number of namespace iterations to complete before producing the estimated auxiliary popularity map ρ, and t_c is a known characteristic time.

Before assessing this algorithm, we consider its runtime, which is approximately $r|\mathcal{N}|t_c\mathsf{RTT}_{max}$, where RTT_{max} is the maximum RTT for any interest probe pair. This is clearly infeasible as $\mathcal{N}$ grows. Therefore, we recommend implementing a parallelized variant of this algorithm. Specifically, instead of traversing $\mathcal{N}$ in sequence, $\mathcal{A}$ can do it in parallel and only sleep for

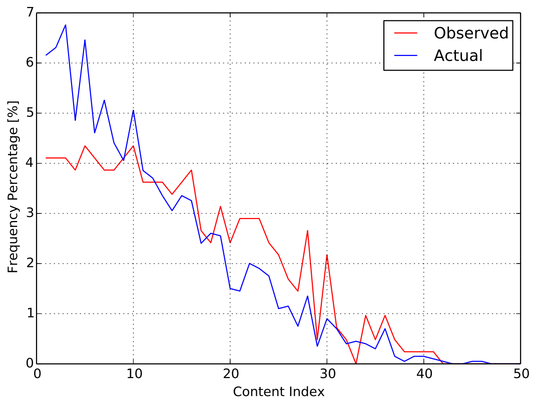
[13]Recall that Interest Returns are sent in response to interest requesting non-existing content. This guarantees that every request receives a response, barring any packet loss.

Algorithm 9 Popularity inference algorithm

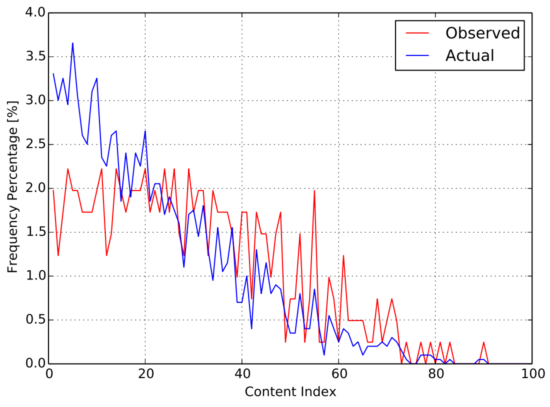
```
1: Input: $\mathcal{N}$, $r$, $t_c$, $\epsilon$
2: Output: $\alpha : \mathcal{N} \to \mathbb{N}$
3: for $N \in \mathcal{N}$ do
4:     $\alpha[N] = 0$
5: for $i = 1, \ldots, r$ do
6:     for $N \in \mathcal{N}$ do
7:         $N_h = N$; $N_m = \mathsf{AppendRandomComponent}(N, 128)$
8:         $t_N = \mathsf{now}()$
9:         Send requests for $N_h$ and $N_m$ in parallel and record their time of arrival in $t_N^h$ and $t_N^m$
10:        $\Delta_N = ||(t_N^h - t_N)| - |(t_N^m - t_N)||$
11:        if $\Delta_N > \epsilon$ then
12:            $\rho[N] = \rho[N] + 1$
13:        Sleep for $t_c$
   return $\alpha$
```

t_c in between each successive probe for N. Given P processing units, runtime is lowered to $rt_c\frac{|N|}{P}\mathsf{RTT}_{max}$, since each name can be processed in parallel.

We evaluate this algorithm as follows. Using ccns3Sim [247], we created a simulation with n consumers $Cr_1, \ldots, Cr_n$, one "monitor" Cr^* ($\mathcal{A}$), and a single producer with S content objects. We chose $S = 50$ and $S = 100$ for our simulations here to illustrate the efficacy of this attack. Each consumer is given access to all these content names. Popularity of content in this collection followed a Zipf distribution with $\alpha = 1.5$. Nodes were arranged in a star topology with a single caching router between them. Storage size of the caching router matched that of the content collection. Each consumer, Cr_i,$i = 1, \ldots, n$, requested a random name from the content collection every second. Meanwhile, Cr^* executed Algorithm 9. When finished, the simulation outputs the *actual* frequency distribution of content (as perceived) by the producer and the *observed* frequency distribution of Cr^*. Figure 5.8 shows these two distributions. Results indicate that the attack algorithm can learn, with fairly high accuracy (for the given values of S), the actual popularity distribution solely by exploiting router caches. (Of course, this accuracy may decline as S increases further. We plan to explore this degradation in future work.)



(a) $S = 50$ items



(b) $S = 100$ items

Figure 5.8: EDF inference algorithm accuracy

The popularity inference attack works if the namespace is *static* and enumerable. However, it no longer applies if the namespace is dynamic or otherwise unpredictable.

Estimating Characteristic Time

Probing frequency f_p must be large enough such that all router caches on the $\mathcal{A}$-to-producer path evict stale content between sequential probes for the same content. Therefore, $\mathcal{A}$ needs to know maximum t_c* for all on-path routers. Assuming interests for N issued by other consumers follow a Poisson Arrival process with rate λ_N, probability that a probe for N results in a cache hit (for any cache) can be characterized as in [101]:

$$h_N = 1 - e^{-\lambda_N t_c*}$$

For a given R_i on the $\mathcal{A}$-to-producer path, t_c^i is a constant that satisfies:

$$\sum_{N \in \mathbb{N}} (1 - e^{-\lambda_N t_c^i}) = C,$$

where C is capacity of the LRU cache. To approximate t_c*, $\mathcal{A}$ must therefore know: (a) capacity of each on-path router and (b) namespace from which content can be requested. Though possible, it is highly unlikely that $\mathcal{A}$ would learn all this information. Therefore, it must be approximated. One trivial way to do this is to assume a large C and modest λ_N to represent each cache and all names. This would not yield a value close to t_c*. However, since the inference algorithm does not require precision in the characteristic time, we deem this acceptable.

5.2.7 Attack Ramifications

We now discuss the efficacy of the privacy attack and importance of caching and namespace enumeration in mitigating this attack.

Attack Efficacy

Success of the attack relies on three key pieces of information: $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$, $\mathcal{D}_R(\mathbb{P})$, and $\mathcal{D}_E(\mathbb{C})$. If $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$ is *perfect*, i.e., $\Delta(\mathcal{D}_A^{\mathcal{A}}(\mathbb{P}), \mathcal{D}_R(\mathbb{P})) \approx 0.0$, the attack's success depends on accuracy of empirically observed popularity distribution. This is because $\mathcal{A}$ uses its knowledge of observed popularity to map items in $\mathbb{P}$ to $\mathbb{C}$. Put another way, the attack is most successful when the following conditions hold:

$$\Delta(\mathcal{D}_A^{\mathcal{A}}(\mathbb{P}), \mathcal{D}_R(\mathbb{P})) \approx 0.0$$
$$\Delta(\mathcal{D}_E(\mathbb{C}), \mathcal{D}_A^{\mathcal{A}}(\mathbb{P})) \approx 0.0$$

As shown earlier, $\mathcal{A}$'s topological placement, network caching, and distribution of content across the network all play a role in widening the gap between these distributions. Any countermeasure requires forcefully widening the gap between any of these distributions. If $\mathcal{A}$

is localized within a part of the network, then distributing content across multiple locations can help bypass it. However, if $\mathcal{A}$ is topologically wide-spread, enabling caching limits the number of events observed, thus making attacks more difficult.

Caching and Privacy

Based on our discussion thus far, it is evident that caching can both help and hurt privacy. Lauinger et al. [197] and Acs et al. [23] showed that a cache can be exploited as an oracle to allow $\mathcal{A}$ to learn when popular content is requested by nearby consumers. This attack uses a timing side-channel based on router caches. It works by requesting popular content from the network via interest "probes" and trying to discern if they have been served from a nearby cache. Similarly, probing attacks that target content producers can be used to discover whether certain content was recently served. One mitigation strategy is to remove the timing channel by requiring caches to artificially delay responding to interests that are marked as "private".

In this section, we showed that caching can *help* privacy by hindering $\mathcal{A}$'s ability to conduct frequency analysis attacks. These attacks do not *directly* rely on the timing side-channel. Instead, they rely on the content popularity side-channel. The only step that relies on timing is when $\mathcal{A}$ estimates content popularity distributions. Without caches, interest frequency is not dampened before reaching $\mathcal{A}$. Therefore, caching is encouraged. Moreover, the timing side-channel mitigation from [23] does not make this attack any easier; it only increases latency for consumers.

An alternate attack mitigation strategy is to protect interest and content packets with semantically secure encryption, as described by Ghali et al. [146] and in Section 5.1. This would obviate any on-path caching and make each interest-content pair unique. Thus, impact on the network, especially in terms of congestion, would likely be substantial.

Namespace Enumeration

If $\mathcal{D}_A^{\mathcal{A}}(\mathbb{P})$ is unknown, feasibility of the frequency analysis attack relies on $\mathcal{A}$'s ability to enumerate the content namespace. This is possible if: $\mathcal{A}$ (a) knows all content names *a priori*, (b) can discover or learn them through external means, (c) can derive them by some namespace convention, e.g., if namespace structure is well-defined, or (d) can learn them from some search engine. Regardless of the method, public content is always enumerable. (If it were not, no one would be able to access it.) Therefore, restricting or controlling enumeration is only possible for restricted content, for which privacy is probably the more important. Thus, if there are access control mechanisms that require consumer authentication before requesting content, enumeration would be restricted to only authorized consumers. This might suffice for some applications. However, consider a CCN-based media distribution service similar to Netflix. Every authorized consumer has access to (essentially) the same content and discovery mechanisms. Since access is pervasive across the entire content collection, enumeration is not preventable. (Every client in the Netflix case can search for and discover the same protected content.)

Note that shortening the lifetime of a name-to-content binding does not help prevent enumeration. CCN requires every content object to have at least one name. Therefore, a producer could update the name-to-data bindings at regular intervals. However, if $\mathcal{A}$ can discover this binding in a given time epoch, it can also almost surely do the same for the next epoch. What matters for the attack considered in this work is how many times *a specific content* is requested, and not how many times a *specific name* is requested.

5.3 AC^{3}N: Efficient Anonymous Communication

ANDāNA (Anonymous Named-Data Networking Application) [105] was the first attempt to support anonymous communication in NDN and CCN. Inspired by Tor [300, 13], it uses onion-like concentric encryption to wrap interests for content that are successively decrypted and forwarded by participating anonymizing routers (ARs). Along the return path towards the consumer, content is wrapped in layers of encryption at each AR. Unlike Tor, which is a mature tool with well over a decade of deployment experience, ANDāNA was a proof-of-concept prototype application for NDN (and CCN). Its main purpose was to demonstrate the feasibility of anonymous content retrieval over NDN. Supporting high-throughput, low-latency, unidirectional, and bidirectional traffic for voice, video, and media streaming applications was not ANDāNA's goal.

Motivated by these shortcomings, we present an improved design for anonymous communication in CCN. Our approach, henceforth referred to as AC^{3}N (Anonymous Communication for Content-Centric Networking), addresses many performance and anonymity pitfalls of ANDāNA. The design of AC^{3}N relies only on the underlying network's ability to pull uniquely named content by name (via interests) and does not depend on any other design features, e.g., in-network caching. This makes AC^{3}N viable for similar ICN architectures.

The remainder of this section is outlined as follows. Section 5.3.1 describes the adversarial model against which AC^{3}N is designed. Section 5.3.2 presents the design of AC^{3}N and Section 5.3.3 reports on its implementation as an application over CCN. We test AC^{3}N in numerous environments with various types of uni- and bi-directional traffic. To illustrate its effectiveness, we compare these performance results to ANDāNA. Results indicate that our design yields noticeable performance gains without sacrificing consumer or producer anonymity. Finally, Section 5.3.4 asserts the anonymity claims of AC^{3}N. We conclude with a discussion of related work specific to anonymity networks and CCN.

5.3.1 Anonymity Overview and Threat Model

Anonymity is influenced by many features of CCN. For example, interest and content names may reveal information about the producer and, potentially, the consumer. Anonymity may also be compromised by the contents of router caches and content object digital signatures. In order to fully understand the extent of these anonymity shortcomings, we adopt the ANDāNA adversarial model and use it in the design and development of $\mathsf{AC}^3\mathsf{N}$. In this model, we assume an adversary who is capable of performing the following actions[14]:

- Deploy compromised routers,
- Compromise existing routers,
- Control content producers,
- Deploy compromised caches, and
- Observe and replay traffic

To keep this model realistic, we assume that the time to mount any one of these attacks is non-negligibly longer than the average RTT for an interest-content exchange. Formally, we define an adversary $\mathcal{A}$ as a 3-tuple: $(\mathsf{P}_{\mathcal{A}}, \mathsf{C}_{\mathcal{A}}, \mathsf{R}_{\mathcal{A}})$ where the components denote the set of compromised producers, consumers, and routers, respectively. See Table 5.4 for a complete list of notation used in this work. Following [105], if $\mathcal{A}$ controls a producer or a consumer then it is assumed to have complete and adaptive control over how they behave in an application session. In other words, $\mathcal{A}$ can control all of the timing, format, and actual information of each content through comprised nodes and links.

We define a *configuration* as a snapshot in time of the current activity associated with a consumer. In other words, each configuration is a relation that maps consumers to the state of a subset of the network. Let $Cr, R_1, \ldots, R_n, P$ be a consumer-to-producer path of length

[14]Any one of these actions can be performed adaptively, i.e., in response to status updates or based on observations.

$(n+1)$ from $Cr \in \mathsf{C}$ to $P \in \mathsf{P}$. Furthermore, let $\overline{\mathsf{int}}_1^n$ be an interest sent from Cr to P that traverses the route $R_1, \ldots, R_n$. A configuration CF is then defined as:

$$\mathsf{CF} : \mathsf{C} \to \{(R_1, \ldots, R_n, P, \overline{\mathsf{int}}_1^n)\}.$$

This relation can be viewed as a map from $C \in \mathsf{C}$ to a set of routers defining a path, or circuit, from C to all $P \in \mathsf{P}$ that interests $\overline{\mathsf{int}}_1^n$ traverse.

As in [105], we define anonymity in the context of indistinguishable consumer configurations. Specifically, two configurations CF and CF' are said to be *indistinguishable with respect to* $\mathcal{A}$, denoted $\mathsf{CF} \equiv_{\mathcal{A}} \mathsf{CF}'$, if, for all such polynomial-time adversaries $\mathcal{A}$ there exists a negligible function ϵ such that:

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{CF}) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{CF}') = 1]| \leq \epsilon(\kappa),$$

for global security parameter κ. This means the probability that, given two configurations, the likelihood that $\mathcal{A}$ can correctly differentiate one from the other is no better than a random guess. If $\mathcal{A}$ was able distinguish between two separate configurations, $\mathcal{A}$ would then also be able to determine, at a minimum, that either (a) *some* interest was sent by two different consumers, or (b) two different interests emanated from the *same* consumer. Since this is the minimum amount of information that can be revealed to $\mathcal{A}$, we use this as our basis for defining consumer, producer, and session anonymity, as well as producer and consumer linkability and interest linkability.

Definition 5.6. *[105] For $Cr \in (\mathsf{C} \setminus \mathsf{C}_{\mathcal{A}})$, Cr has* consumer anonymity *in* CF *with respect to* $\mathcal{A}$ *if* $\exists\ \mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$; *such that* $\mathsf{CF}'(Cr') = \mathsf{CF}(Cr)$ *and* $Cr' \neq Cr$.

Definition 5.7. *[105] Given* $\overline{\mathsf{int}}_1^n$ *and* $P \in \mathsf{P}$, $Cr \in \mathsf{C}$ *has* producer anonymity *in* CF *with respect to* P *and* $\mathcal{A}$ *if* $\exists\ \mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$*such that* $\overline{\mathsf{int}}_1^n$ *is sent by a non-compromised consumer to* $P' \neq P$.

Definition 5.8. *Two entities P and C serving as producer and consumer in an application session are said to have* $\mathsf{session\ anonymity}$ *in* CF *with respect to* $\mathcal{A}$ *if both C and P have producer and consumer anonymity in* CF *with respect to* $\mathcal{A}$.

There are two types of linkability that are important in this work: producer and consumer linkability, and interest linkability. Both of these are defined with respect to consumers, producers, interests, and content objects. Informally, two or more of these "entities" are *unlinkable* with respect to $\mathcal{A}$ if $\mathcal{A}$ cannot determine if they are related in any meaningful way. As an example, such a meaningful relation might be that content object $C(N)$ corresponds to the interest $I(N)$.

Since packet (message) arrivals are discrete events observed at consumers, routers, and producers, we refer to distinct messages based on the order in which they arrive. Specifically, let $\mathsf{int}{:}i^e$ be the i-th interest received or processed by entity e in the network, e.g., a consumer, router, or producer. With this notation in place, we formally define interest unlinkability below. Content object unlinkability has an analogous definition.

Definition 5.9. *Two interests* $\mathsf{int}{:}i^e$ *and* $\mathsf{int}{:}j^e$ *that arrive at entity e are* $\mathsf{unlinkable}$ *with respect to* $\mathcal{A}$ *in configuration* CF *if*

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{CF}, \mathsf{int}{:}i^e) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{CF}, \mathsf{int}{:}j^e) = 1]| \leq \ \epsilon(\kappa).$$

Producers and consumers may also be linkable with respect to a particular configuration CF and $\mathcal{A}$. Intuitively, this means that interests issued by a consumer and those received by a producer can be *paired*. We formally define the inverse of this idea below.

Definition 5.10. *A producer $P \in \mathsf{P}$ and consumer $Cr \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are* $\mathsf{unlinkable}$ *in* CF *with respect to* $\mathcal{A}$ *if there exists* $\mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$ *where interests generated from Cr are sent to a producer $P' \neq P$.*

It may be easier to consider the notion of linkability instead. Specifically, a producer $P \in \mathsf{P}$ and consumer $Cr \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are linkable if in CF with respect to $\mathcal{A}$ if *all interests* sent from Cr are sent to P.

Linkability and anonymity are closely related. Consider the following corollaries, which are proved in [105].

Corrolary 5.3. *[105] Producer $P \in \mathsf{P}$ and consumer $Cr \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are unlinkable in configuration CF with respect to $\mathcal{A}$ if P has producer anonymity with respect to Cr's interests or Cr has consumer anonymity and $\exists\ \mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$ where $\mathsf{CF}'(Cr') = \mathsf{CF}(Cr) = Cr$ with $Cr' \neq Cr$ and Cr''s interests are routed to a producer $P' \neq P$.*

Corrolary 5.4. *[105] Producer $P \in \mathsf{P}$ and consumer $Cr \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are unlinkable in configuration CF with respect to $\mathcal{A}$ if both P and Cr have producer and consumer anonymity, respectively.*

Our primary goal is to achieve consumer and producer anonymity and unlinkability with minimal overhead. We describe key design elements and show that $\mathsf{AC^3N}$ achieves this goal in Section 5.3.2. For brevity, we analyze claims of anonymity and unlinkability in Section 5.3.4.

ANDāNA Highlights

To motivate $\mathsf{AC^3N}$, we first re-examine ANDāNA. In ANDāNA, circuit ARs are chosen by consumers. Before interests are issued by a consumer, names are first *wrapped* in concentric layers of encryption. Each "layer" contains a routable name prefix for the next hop (AR) in the circuit and the underlying encrypted layers. Each AR decrypts their layer of the name to obtain the next routable prefix in the circuit and corresponding layer, i.e., it "unwraps" its layer of encryption, and then forwards the interest with the new name accordingly. Upon

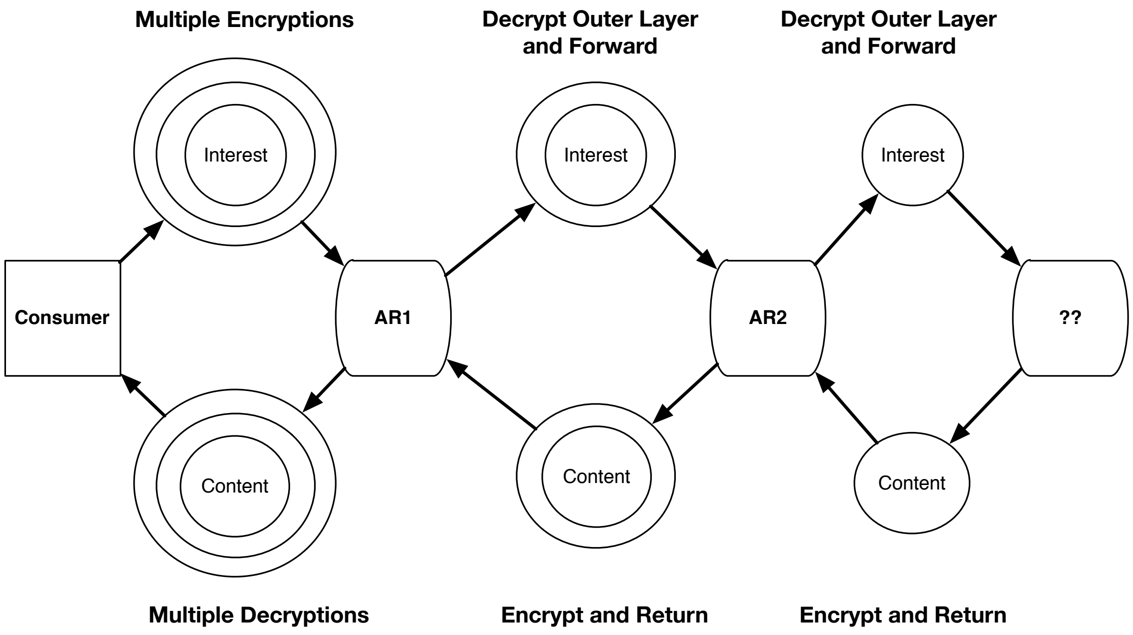


Figure 5.9: ANDāNA concentric encryption and decryption

the receipt of content objects in the reverse path, each AR will encrypt the entire content object and forward the "wrapped" result to the next downstream hop. The consumer then recovers the content object by iteratively decrypting each layer of encryption surrounding the content object. This linear wrapping and unwrapping behavior is illustrated in Figure 5.9. The right-most entity is labeled with "??" because the plaintext interest may traverse through more than a single hop before reaching the final producer.

Unlike Tor, ANDāNA does not support persistent anonymous circuits between consumers and producers. Rather, ephemeral (one-time) circuits are created as the interest is sequentially decrypted and forwarded. State information in each AR is only maintained in the symmetric (session-based) variant ANDāNA.

In the symmetric variant of ANDāNA, state information, consisting of a unique session identifier and symmetric key used for interest and content decryption and encryption, respectively, is established in each anonymizing router using a standard key exchange (handshake) protocol. The use of symmetric encryption removes the computational burden of public key encryption. However, ANDāNA requires that the session identifier be sent in the clear for every interest, which allows $\mathcal{A}$ to link interests and content packets, thus enabling deanonymization attacks against consumers (see below for details). Furthermore, the handshake procedure wastes consumer bandwidth and time, especially in the case of short-term communication.

Identified Issues

The primary motivation for $\mathsf{AC}^3\mathsf{N}$ is to attain the same anonymity guarantees as the public key variant of ANDāNA with *better* versatility and performance. Although ANDāNA includes a symmetric (session-based) variant as a more efficient alternative, it does not provide unlinkability. Generally, unlinkability is a sufficient, rather than a necessary, condition for anonymity. However, in ANDāNA interest-content correlation can lead to consumer and producer linkability, which can immediately violate anonymity.

For example, suppose that $\mathcal{A}$ eavesdrops on incoming and outgoing interests for a particular AR. By analyzing traffic patterns, $\mathcal{A}$ can link incoming and outgoing session IDs. In fact, a variant of this adversary was studied in the context of Tor by Murdoch and Danezis in [232] and was shown to be quite successful. We believe that the same attack could be augmented for ANDāNA. In particular, repeating this attack at each AR in a circuit can result in deanonymization of both the consumer and producer.

The use of application and environment contextual information has been investigated by Franz et al. [127], where side-channel and environment information (e.g., deterministic behavior of an AR always forwarding a packet after unwrapping an interest received from a downstream neighbor) is used to quantify the *degree of unlinkability.* Furthermore, regardless of how linkability information is acquired, it has been shown that it can degrade consumer and producer anonymity beyond that attainable by general traffic analysis [276].

Since most relevant literature focuses on mix-based anonymizing services akin to Tor, upon which ANDāNA was designed, it is clear that all linkability problems studied in the context of Tor are also applicable to symmetric variant of ANDāNA. This is why one of the key goals of AC^{3}N is to attain the same anonymity guarantees as the public key variant of ANDāNA, which has no linkability issues, while still providing more efficient support for low-latency, high-throughput and bidirectional traffic, as compared to the symmetric variant of ANDāNA.

5.3.2 System Design

This section describes the design of AC^{3}N. All relevant notation is presented in Table 5.4.

Table 5.4: Relevant AC^{3}N notation.

Notation	Description
C	Set of all consumers
P	Set of all producers
R	Set of all routers
κ	Global security parameter
Cr, P	Consumer and producer, respectively
$R_i \in \mathsf{R}$	The i-th anonymizing router (AR) in an AC^{3}N circuit
Circ_i	A set of session information corresponding to the i-th circuit at an AC^{3}N consumer
$\overline{\mathsf{int}}_i^j$	Encrypted interest wrapped from R_i to R_j $(i \leq j)$
(pk_i, sk_i)	Public and private key pair of router R_i
$\mathcal{E}_{pk_i}(\cdot)$	Public key encryption using pk_i
$\mathcal{D}_{pk_i}(\cdot)$	Public key decryption using pk_i
$\mathsf{Encrypt}_{k_i}(\cdot)$	XOR-based symmetric key encryption using key k_i
$\mathsf{Decrypt}_{k_i}(\cdot)$	XOR-based symmetric key decryption using key k_i
ST_i	AR R_i session table used to store session ID and digest tuples
E_{k_i}	Interest and content encryption key for AR R_i
M_{k_i}	Shared MAC key for AR R_i and R_{i+1}
$\mathsf{EncryptionIV}_i$	Encryption initialization vector used for R_i
$\mathsf{SessionIV}_i^j$	j-th value of the dynamic session initialization vector for R_i
$\mathsf{Session}_i$	Session ID for r_i
$\mathsf{SessionIndex}_i^j$	j-th dynamic session index (identifier) shared between the consumer and AR R_i
$H(\cdot)$	Collision-resistant hash function with domain $\{0,1\}^*$ and range $\{0,1\}^\kappa$

Circuit and Session Establishment

Similar to Tor [300], anonymizing routers (ARs) and circuits are at the core of $\mathsf{AC^3N}$. As previously mentioned, a *circuit* is a sequence of ARs through which upstream interests and downstream content objects flow. Circuits are established for long-term sessions, i.e., they are not ephemeral. ARs in a circuit serve two purposes: (1) decrypt and forward encrypted interests, and (2) encapsulate (encrypt) content objects using previously acquired or agreed upon keys and forward them downstream.

Consumers generate interests wrapped in several layers of encryption and receive content objects also wrapped in several layers of encryption that it can decrypt. Each AR is an *application* running on router, and therefore technically serves as the producer for each downstream AR in the circuit. Unlike standard CCN content that carries a digital signature, $\mathsf{AC^3N}$ uses MACs for more efficient authenticity checks.

To increase interest and content throughput, circuit sessions are established and initialized with long-term symmetric keys used for both content encryption and MAC generation and verification. The complete set of session state information, which is established for n ARs $R_1, \ldots, R_n$ in a circuit, is as follows:

- Session IDs, $\mathsf{Session}_i$, and session initialization vectors (IVs), $\mathsf{SessionIV}_i^0$,
- Content encryption keys, E_{k_i}, and initial counter values, $\mathsf{EncryptionIV}_i$, and
- Pairwise MAC keys M_{k_i} between adjacent ARs and the consumer (used to tag and verify content).

Algorithm 10 describes the circuit establishment procedure in more detail. Consumers execute the $\mathsf{EstablishCircuit}$ function, which invokes the Init function to establish state with AR R_i, $i = 1, \ldots, n$. The ARs accept session establishment via the $\mathsf{InitHandler}$ functions, which store session information in local memory and respond with an appropriate acknowledgment. Note that no two routers will share the same session identifier (with non-negligible probability) even though they are part of the same circuit, since consumers generate session identifiers independently and uniformly at random from $\{0,1\}^\kappa$.

After a circuit and its session information have been established, all subsequent traffic is protected via a CCA-secure symmetric scheme [180]. The encryption and MAC key for router R_i are indexed via $\mathsf{SessionIndex}_i^j$, the dynamic session index (identifier) sent in the clear along with the encrypted interest. To provide unlinkability, the session index is "rotated" from $\mathsf{SessionIndex}_i^j$ to $\mathsf{SessionIndex}_i^{j+1}$, after each new interest is received and forwarded, using a one-way and strongly collision-resistant hash function $H(\cdot)$. Specifically, the transfer functions are:

$$\begin{aligned}\mathsf{SessionIndex}_i^{j+1} &= H(\mathsf{SessionIV}_i^j + \mathsf{Session}_i)\\ \mathsf{SessionIV}_i^{j+1} &= 1 + \mathsf{SessionIV}_i^j \mod (2^\kappa).\end{aligned}$$

Note that $\mathsf{SessionIV}_i^j$ is kept private.

$\mathsf{AC^3N}$ sessions are unidirectional. Thus, bidirectional traffic requires two sessions. This allows each party to choose its own set of ARs. Besides promoting better privacy, this can improve QoS by distributing computational load among multiple, and possibly distinct, ARs.

State initialization in $\mathsf{AC^3N}$ is separate from circuit usage, i.e., it uses a handshake routine to initialize state. However, our design does not preclude on-line state establishment. For example, the first wrapped interest issued by a consumer for a new circuit could be overloaded to include all of the state establishment information in addition to the associated interest information.

Algorithm 10 Circuit session establishment

```
Require: Anonymous routers $R_1, \ldots, R_n$ ($n \geq 1$) with public keys $pk_1, pk_2, \ldots, pk_n$.
1: function InitHandler(int)
2:     $(E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i^1, \mathsf{Session}_i) := \mathcal{D}_{sk_i}(\mathsf{int})$
3:     $\mathsf{SessionIndex}_i^1 := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^1)$
4:     Store $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i)$
5:     Insert $(\mathsf{SessionIndex}_i^1, \mathsf{Session}_i, \mathsf{SessionIV}_i^1)$ into the session table $\mathsf{ST}_i$
6:     $\mathsf{resp} \leftarrow \mathsf{Encrypt}_{E_{k_i}}(\mathsf{SessionIndex}_i^1)$
7:     return resp
8: function Init($r_i$, $M_{k_{i+1}}$)
9:     $E_{k_i} \leftarrow \{0,1\}^\kappa, M_{k_i} \leftarrow \{0,1\}^\kappa$
10:    $\mathsf{EncryptionIV}_i \leftarrow \{0,1\}^\kappa, \mathsf{SessionIV}_i^1 \leftarrow \{0,1\}^\kappa$
11:    $x_i \leftarrow \{0,1\}^\kappa, \mathsf{Session}_i := H(x_i)$
12:    $\mathsf{SessionIndex}_i^1 := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^1)$
13:    $\mathsf{Payload} := \mathcal{E}_{pk_i}(E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i, \mathsf{Session}_i)$
14:    $\mathsf{int} := \mathsf{namespace}_i/\mathsf{CREATESESSION}/\mathsf{Payload}$
15:    $\mathsf{resp} := \mathsf{GetContent}(\mathsf{int})$
16:    $(\mathsf{AckSessionIndex}_i^1) := \mathsf{Decrypt}_{E_{k_i}}(resp)$
17:    if $\mathsf{SessionIndex}_i^1 = \mathsf{AckSessionIndex}_i^1$ then
18:        return $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, x_i, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1)$
19:    else
20:        return Error
21: function EstablishCircuit($j$, $R_1, \ldots, R_n$)
22:    $(\mathsf{Session}_n, E_{k_n}, M_{k_n}, \mathsf{EncryptionIV}_n^1, \mathsf{SessionIV}_n^1) := \mathsf{Init}(r_n)$
23:    $\mathsf{Circ}_j = \{\}$
24:    $\mathsf{Circ}_j[n] = [(\mathsf{Session}_n, E_{k_n}, M_{k_n}, \mathsf{EncryptionIV}_n^1, \mathsf{SessionIV}_n^1)]$
25:    for $i = n-1$ downto 1 do
26:        if $i = n-1$ then
27:            $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1) := \mathsf{Init}(R_i, \bot)$
28:        else
29:            $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1) := \mathsf{Init}(R_i, M_{k_{i+1}})$
30:        $\mathsf{Circ}_j[i] = (\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1)$
```

AC^{3}N Circuit Usage

Generally, AC^{3}N circuits are used the same way as in ANDāNA. The encrypted interest generation procedure is shown in Algorithm 11. In it, a consumer wraps an interest for a sequence of ARs and forwards it towards the first AR.

Note that each encrypted interest also includes a timestamp to mitigate replay attacks. The interest and content forwarding procedures are shown in Algorithms 12 and 13, respectively. Superscripts for session IVs and indexes are omitted for presentation clarity.

Content encryption at R_i uses a stream cipher whose key stream is initialized by $\mathsf{EncryptionIV}_i$. As presented in the content forwarding routine, $\mathsf{EncryptionIV}_i$ is advanced similarly to the $\mathsf{SessionIV}_i$ so that the key stream is a fresh pseudorandom bit string for each content object. One additional benefit of this form of encryption is that it permits the key stream to be precomputed offline. It does, however, introduce the probability of improperly computed key streams, which will result in corrupt ciphertext. Section 5.3.4 discusses this issue in more detail.

After issuing an interest using the encrypted interest generation procedure, an encrypted content object and MAC tag tuple $\overline{data_i^n} = (data_i^n, \sigma_1)$ is returned. The consumer then verifies the MAC tag σ_1 and then decrypts $data_i^n$. The commutative property of XOR allows the consumer to decrypt each layer of the content in any arbitrary order.

Algorithm 11 Interest onion encryption

```
Require: Interest $\mathsf{int}$, $R_1, \ldots, R_n$ circuit length $n$
Ensure: Encrypted interest $\overline{\mathsf{int}}_1^n$
1: $\overline{\mathsf{int}} = \mathsf{int}$
2: for $i = n$ downto 1 do
3:     $\{\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^k, \mathsf{SessionIV}_i^k\} := \mathsf{Circ}_j[i]$
4:     $\mathsf{SessionIndex}_i^k := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^k)$
5:     $\mathsf{SessionIV}_i^{k+1} = \mathsf{SessionIV}_i^k + 1 \pmod{2^\kappa}$
6:     $\overline{\mathsf{int}}_i^n = R_i/\mathsf{SessionIndex}_i^k/\mathsf{Encrypt}_{E_{k_i}}(\overline{\mathsf{int}}, \mathsf{timestamp})$
7: return $\overline{\mathsf{int}}_1^n$
```

5.3.3 Performance Assessment

In this section we assess the performance of $\mathsf{AC^3N}$ against $\mathsf{AND\bar{a}NA}$. $\mathsf{AND\bar{a}NA}$ was originally implemented in C using the CCNx 0.8x library [10]. To bring the evaluation up to speed

Algorithm 12 Interest decryption and forwarding

```
Require: $\overline{\mathsf{int}}_i^n := R_i/\mathsf{SessionIndex}_i^k/\mathsf{Encrypt}_{E_{k_i}}(\overline{\mathsf{int}}, \mathsf{timestamp})$
Ensure: $(\overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$ or discarded packet
 1: if $\mathsf{SessionIndex}_i^k \in \mathsf{ST}_i$ then
 2:     $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i^j) := \mathsf{Lookup}(ST_i, \mathsf{SessionIndex}_i)$
 3:     $\mathsf{SessionIV}_i^{k+1} := \mathsf{SessionIV}_i^k + 1 \pmod{2^\kappa}$
 4:     $\mathsf{SessionIndex}_i^{k+1} := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^{k+1})$
 5:     Update $(\mathsf{SessionIndex}_i^{k+1}, \mathsf{Session}_i, \mathsf{SessionIV}_i^{k+1})$ in $\mathsf{ST}_i$
 6:     $(\overline{\mathsf{int}}_{i+1}^n, timestamp) := \mathsf{Decrypt}_{E_{k_i}}(\overline{\mathsf{int}}_i^n)$
 7:     if decryption fails or $\mathsf{timestamp}$ is not stale then
 8:         Discard $\overline{\mathsf{int}}_i^j$
 9:     else
10:         Persist tuple $T_i = (\overline{\mathsf{int}}_i^n, \overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$ to pending interest table $\mathsf{PT}_i$
11:         return $(\overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$
12: else
13:     Discard $\overline{\mathsf{int}}_i^n$
```

with existing technology, both ANDāNA and AC^{3}N were implemented using the CCNx 1.0 library from PARC. All experiments were conducted on VMs running Ubuntu 14.04 LTS. Each host was equipped with an Intel(R) Core(TM) i5-3427U CPU at 1.80GHz with 8GB of main memory. Also, we use the public key variant of ANDāNA, since it provides the same anonymity guarantees as AC^{3}N.

Note that we do not include Tor in our performance comparison. We argue that such a comparison would be ineffective and misleading. Tor is designed to run over TCP/IP network architectures, whereas AC^{3}N is designed for CCN architectures. Current CCN implementations run as overlays upon TCP/IP. Thus, there is an unavoidable amount of overhead incurred by running AC^{3}N. Put another way, TCP/IP and the CCN architectures differ fundamentally in that there is virtually no transport and network layer in the latter. Thus, AC^{3}N performance results would need to take this overhead into consideration when compared against Tor over TCP/IP.

Algorithm 13 Content encryption and forwarding

```
Require: Content $\overline{data^n_{i+1}}$ in response to interest $\overline{\mathsf{int}}^n_{i+1}$
Ensure: Encrypted data packet $\overline{data^n_i}$
 1: Recover tuple $T_i = (\overline{\mathsf{int}}^n_i, \overline{\mathsf{int}}^n_{i+1}, \mathsf{Session}_i)$ based on $data^n_{i+1}$
 2: Parse $\overline{data^n_{i+1}}$ as a tuple $(data^n_{i+1}, \sigma_{i+1})$
 3: if $\sigma_{i+1} = \perp$ and $M_{k_{i+1}} = \perp$ then
 4:     Verify the signature of $\overline{\mathsf{int}}^n_{i+1}$.
 5:     if The signature passed verification then
 6:         Pass
 7:     else
 8:         return $\mathsf{Error}$
 9: else if $\sigma_{i+1} \neq \perp$ and $M_{k_{i+1}} \neq \perp$ then
10:     if $\sigma_{i+1} = \mathsf{Verify}_{M_{k_{i+1}}}(data^j_{i+1})$ then
11:         Pass
12:     else
13:         return $\mathsf{Error}$
14: else
15:     return $\mathsf{Error}$
16: Remove signature or MAC tag and name from $data^n_{i+1}$
17: Create new empty data packet $data^n_i$
18: Set name for $data^n_i$ as the name for $\overline{\mathsf{int}}^n_i$
19: $data^n_i := \mathsf{Encrypt}_{E_{k_i}}(\mathsf{EncryptionIV}^k_i, data^n_{i+1})$
20: $\mathsf{EncryptionIV}^{k+1}_i := \mathsf{EncryptionIV}^k_i + 1 \pmod{2^\kappa}$
21: $\sigma_i := \mathsf{MAC}(data^n_i)$
22: $\overline{data^n_i} = \overline{(data^n_i, \sigma_i)}$
23: return $\overline{data^n_i}$
```

Unidirectional Assessment

Perhaps the most standard use case for both ANDāNA and AC^{3}N is as a unidirectional anonymizing circuit. Thus, it is important to assess the performance of both tools for this particular use case. In this work, we consider the most important metrics for performance to be (a) interest-content latency L and (b) and throughput S. To assess these metrics, we consider the following simple experiment. Let $\mathsf{Circ} = R_1, \ldots, R_n$ be a circuit of length $n-1$ with n AR nodes. A client Cr is connected to R_1, and Cr wishes to retrieve content from producer P connected to R_n. Thus, the complete path is $Cr, R_1, \ldots, R_n, P$. To obtain content, Cr issues a random interest that can be satisfied by P through Circ. Such an interest

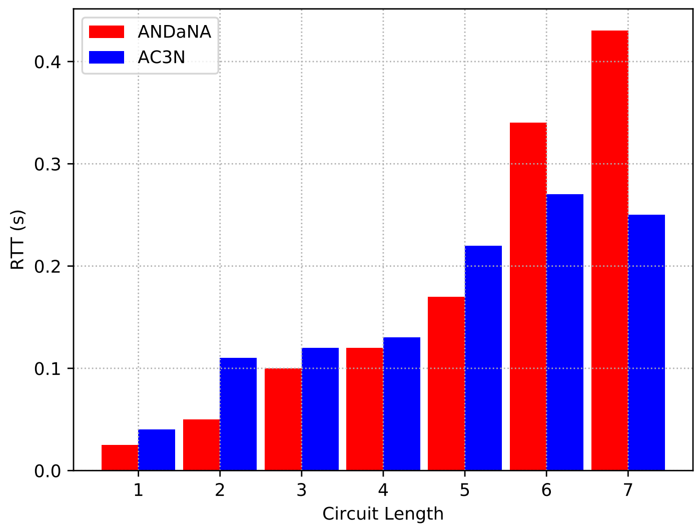


Figure 5.10: Unidirectional AC^{3}N RTT measurements

is issued once every t seconds and the RTT is recorded. To compute the average latency, the average of all observed RTTs is computed. To compute the maximum throughput, the total number of content bytes received is divided by the total time to send a large amount of back-to-back interests without delay. The RTT and throughput measurements as a function of the circuit length are shown in Figures 5.10 and 5.11, respectively.

Bidirectional Comparison

In order to justify AC^{3}N as a viable choice for anonymous communication suitable for low-latency, bidirectional traffic over CCN, we first establish a baseline of performance measurements. Since ANDāNA targeted circuits of length 2,i.e., two AR hopes, we consider circuits of the same length. To set a baseline of performance measurements, we instantiated two entities Cr_1 and Cr_2 (both acting as a producer and consumer) that interact by requesting moderate-size content in short, frequent intervals. In order ensure that such content is never satisfied by the cache of any intervening AR, which is a reasonable assumption for real-time

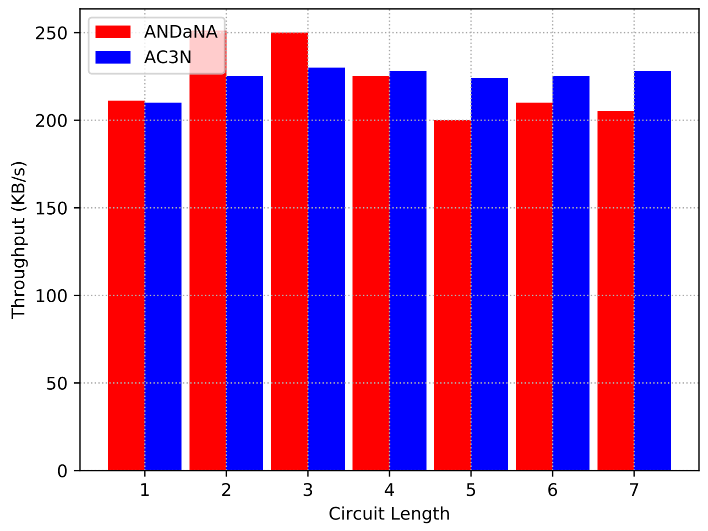


Figure 5.11: Unidirectional AC^{3}N throughput measurements

voice and video applications that always want fresh content, instead of stale cached content, each content is requested from an anonymized namespace and indexed by a sequence number. For example, to request the latest content from Cr_2, Cr_1 issues an (encrypted) interest for `/Cr2/S`, where S is the sequence number, incremented as soon as the interest is issued. This ensures that such interests are never satisfied by any router-cached content for the duration of the session.

With this experimental setup, we tested AC^{3}N under the following scenarios:

1. Cr_1 and Cr_2 connected point-to-point (i.e., no intermediate hops).
2. Cr_1 and Cr_2 connected via two "insecure" ARs that perform no interest or content encryption (i.e., each AR just serves as an application-level proxy to forward interests and content along through the circuit).
3. Cr_1 and Cr_2 connected via two "secure" ARs that perform interest and content encryption.

Table 5.5: ANDāNA baseline bidirectional performance metrics.

Test Scenario	L_1 (s)	σ_1 (s)	L_2 (s)	σ_2 (s)
1	0.00735	0.00798	0.00340	0.00053
2	0.01321	0.00257	0.01359	0.00567
3	0.03905	0.13791	0.04248	0.13923

Table 5.6: AC^{3}N bidirectional performance metrics.

Test Scenario	L_1 (s)	σ_1 (s)	L_2 (s)	σ_2 (s)
1	0.00517	0.00813	0.00541	0.00049
2	0.01176	0.00371	0.01521	0.00611
3	0.07805	0.12865	0.05321	0.11691

Table 5.5 shows performance results from scenarios 1, 2, and 3 for ANDāNA. We characterize performance with respect to the user-perceived values of these latency L. We denote L_i as the perceived (RTT) latency of party $Cr_i \in \{1, 2\}$. We also assess the standard deviation for each of these measurements. All experiments were performed using the following procedure:

- Each party generates $1,000$ sequential messages, at a rate chosen uniformly from the range $[1, 100]$s.
- Each party responds to all interests with a random content object of 150B.

According to our preliminary experimental evaluation, with results shown in Table 5.6, low-latency, bidirectional communication is feasible using ANDāNA.

5.3.4 Security and Correctness Analysis

To assess the anonymity claims of AC^{3}N we start by adopting the adversarial model of ANDāNA [105] and previous described in Section 5.3.1. The fundamental differences between ANDāNA and AC^{3}N are that, in the latter, (1) each pair of adjacent routers share a distinct MAC key used for efficient content authenticity checks and (2) sessions are identified by the output of $H(\cdot)$, rather than encrypting and decrypting interests using expensive asymmetric procedures. Accordingly, proofs of anonymity need to be augmented to take this into account.

The remainder of the design is syntactically equivalent to that of $\mathsf{AND\bar{a}NA}$. Thus, we simply re-state relevant theorems without proof. In doing so, however, we generalize them to circuits of length $n \geq 2$.

Theorem 5.7. *[105] Consumer $c \in (\mathsf{C} \setminus \mathsf{C}_{\mathcal{A}})$ has consumer anonymity in configuration* CF *with respect to adversary $\mathcal{A}$ if there exists $c \neq c'$ such that any of the following conditions hold:*

1. *c, c' are in the same anonymity set with respect to the adversary $\mathcal{A}$ (see [105]).*
2. *There exist ARs r_i and r'_i such that $r_i, r'_i \notin \mathsf{R}_{\mathcal{A}}$, both r_i and r'_i are on the circuit traversed by $\overline{\mathsf{int}}_1^n$.*

Theorem 5.8. *[105] Consumer c has producer anonymity in configuration* CF *with respect to producer $p \in \mathsf{P}$ and adversary $\mathcal{A}$ if there exists a pair of ARs r_i and r'_i such that r_i and r'_i (for some uncompromised entity $c \notin \mathsf{C}_{\mathcal{A}}$) are on the path traversed by $\overline{\mathsf{int}}_1^n$, $p \neq p'$, and all routers $r_1, \ldots, r_n$ are equal in $\mathsf{CF}(c)$ and $\mathsf{CF}(c')$.*

Unlike the $\mathsf{AND\bar{a}NA}$ session-based design, $\mathsf{AC^3N}$ does not suffer from interest linkability. This result is captured in the following theorem.

Theorem 5.9. *Independent interests corresponding to the same circuit session are not linkable.*

Proof. Let $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+1)$ be two subsequent interests issued by a consumer using the same circuit and received at router r_j. By Algorithm 11, after $\mathsf{int} : i$ is issued, $\mathsf{SessionIndex}_j^{i+1}$becomes $H(\mathsf{Session}_j^i + (\mathsf{SessionIV}_j^i + 1))$ and $\mathsf{SessionIV}_j^{i+1}$ becomes $\mathsf{SessionIV}_j^i + 1 (\bmod 2^{\kappa})$. Thus, the session index included in $\mathsf{int}{:}(i+1)$ is $\mathsf{SessionIndex}_j^{i+1}$. By the properties of H, $\mathsf{SessionIndex}_j^{i+1}$ is indistinguishable from $\mathsf{SessionIndex}_j^i$ since the inputs are different. Therefore, without knowledge of $\mathsf{SessionIV}_j^i$, an adversary cannot link $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+1)$ to the same session. This property thus holds recursively for any two interests $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+k)$, $(k > 2)$. Therefore any two independent interests corresponding to the same circuit session are unlinkable. $\square$

In addition to anonymity properties, we are also concerned with the correct operation of each AR supporting a session between two parties. In this context, we define session correctness as the ability of a consumer to correctly decrypt content that is generated *in response to* its original interest. That is, if a consumer issues an interest, it should be able to correctly decrypt the content that it receives. The following factors impact the correctness of the session:

1. Each AR $r_1, \ldots, r_n$ on the consumer-to-producer circuit should correctly recover the session identifier associated with the current session.
2. The session key streams should only be advanced upon the receipt of an interest corresponding to the consumer who initiated the session or content that is generated from the upstream router (potentially the producer) in the circuit.

The first item is necessary in order for each AR to correctly decrypt interests, encrypt content, and perform content signature generation and verification. The second item is necessary so that all content can be correctly decrypted by the consumer. We claim that, given a CCA-secure public key encryption scheme, the probability that either one of these factors being violated by an adversary $\mathcal{A}$ is negligible. Let $\mathsf{ForgeSession}$ and $\mathsf{KeyJump}$ denote the events corresponding to instances where an adversary creates a ciphertext that maps to a valid session identifier for *some* session currently supported by an AR, i.e., the forged session belongs to the routers session table ST, and the event that an adversary causes the key stream for *some* AR in a consumer-to-producer circuit to fall out of sync with the consumer.

By the design of $\mathsf{AC^3N}$, it should be clear that $\mathsf{KeyJump}$ occurs when $\mathsf{ForgeSession}$ occurs, since the key stream is only advanced upon receipt of an interest, but may also occur when an adversary successfully forges a MAC tag corresponding to the signature of a piece of content from the upstream router (or producer). We denote this latter event as $\mathsf{ContentMacForge}$. With the motivation in place, we now formally analyze the probabilities of these events occurring below. For notational convenience, we assume that each event only occurs as a result of some adversarial action, so we omit this relation in what follows.

Lemma 5.1. *For all PPT adversaries $\mathcal{A}$, there exists some negligible function* negl *such that*

$$\Pr[\mathsf{ForgeSession}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, we know that session identifiers are computed as the output of a collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^m$, where $m = \mathsf{poly}(\kappa)$ (i.e., polynomial in the global security parameter). Consequently, forging a session identifier *without* the input to H implies that a collision was found, thus violating collision resistance of H. Thus, forging a session is equally hard as finding a collision in H, or more formally, $\Pr[\mathsf{Collision}(H) = 1] = \Pr[\mathsf{ForgeSession}]$. By the properties of collision resistance of H which states that $\Pr[\mathsf{Collision}(H) = 1] \leq \mathsf{negl}(\kappa)$ for some negligible function negl, it follows that $\Pr[\mathsf{ForgeSession}] \leq \mathsf{negl}(\kappa)$.

□

Lemma 5.2. *For all PPT adversaries $\mathcal{A}$, there exists some negligible function* negl *such that*

$$\Pr[\mathsf{ContentMacForge}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, the MAC scheme used for content symmetric content signature generation and verification is defined as the algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$, where Gen generates the secret key k used in the scheme, $\mathsf{Mac}_k(m)$ outputs the MAC tag $t := F_k(m)$ for some PRF F, and $\mathsf{Verify}_k(m, t)$ outputs 1 if $t = \mathsf{Mac}_k(m)$ and 0 otherwise. This is known and proven to be a secure MAC scheme, meaning that for all PPT adversaries

$\mathcal{A}$ there exists a negligible function negl such that $\Pr[\mathsf{MacForge}_{\mathcal{A},\Pi}(1^\kappa) = 1] \leq \mathsf{negl}(\kappa)$, and since $\mathsf{ContentMacForge}$ occurs exactly when the even $\mathsf{MacForge}$ occurs, we have that $\Pr[\mathsf{ContentMacForge}] \leq \mathsf{negl}(\kappa)$. □

Lemma 5.3. *For all PPT adversaries $\mathcal{A}$, there exists some negligible function* negl *such that:*

$$\Pr[\mathsf{KeyJump}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, it follows that $\Pr[\mathsf{KeyJump}] = \Pr[\mathsf{ForgeSession}]+\Pr[\mathsf{ContentMacForge}]$, and since the sum of two negligible functions is also negligible, it follows that there exists some negligible function negl such that $\Pr[\mathsf{KeyJump}] \leq \mathsf{negl}(\kappa)$. □

Theorem 5.10. *Session correctness of* $\mathsf{AC^3N}$ *is only violated with negligible probability.*

Proof. This follows immediately from Lemmas 1, 2, and 3 and the fact that the sum of two negligible functions is also negligible.[15] □

5.4 CCVPN: Namespace Tunnels

Neither ANDāNA nor $\mathsf{AC^3N}$ were designed for the simple VPN use-case of protecting traffic contents without hiding identities of communicating parties. Anonymizing circuits, as used in ANDāNA and $\mathsf{AC^3N}$, are unnecessary when the goal is communication privacy instead of complete anonymity. Furthermore, similar to Tor, ANDāNA is an application-layer tool, which means that it is not suitable for high-volume low-latency communication. In contrast, a single network-layer VPN tunnel can serve many consumers within a trusted domain.

[15]This sum comes from the fact that probability of "failure" events must be taken into account in both directions of the session.

Judging from the popularity and utility of VPNs in today's Internet, we conclude that a VPN-like technology is a much-needed tool for CCN.

In this section, we present CCVPN, the first CCN-based network-layer tunnel. Similar to ANDāNA and AC^{3}N, CCVPN encrypts interest and content packets between two end-points. However, these end-points are network gateways instead of ARs. In the standard configuration, both tunnel end-points are gateways between trusted domains. Tunnels may also be nested, similar to IPsec [185]. One of the end-points (the source) can be an individual consumer. In fact, the standard two-hop AR circuit is identical to a nested tunnel with the same source. Though designed to use efficient symmetric-key cryptographic primitives, CCVPN also works with public-key cryptography. This makes it easy to deploy in real-world CCN networks.

We implemented CCVPN and experimentally assessed its performance. Our results indicate that collective throughput across multiple consumers sharing a tunnel remains stable, up to a modest bound of 60 consumers, each requesting content at the rate of 1 mbps. Moreover, as expected, the average RTT of consumer requests decreases proportionally to collective throughput and request rate. Further improvements in both throughput and perceived RTT can be made with an implementation in a faster CCN router, such as that in the CICN project [115]. We leave this for future work.

5.4.1 System Design

Virtual Private Networks (VPNs) support secure communication between networks on the Internet. They allow users to send and receive data across insecure public networks as if their computing devices were directly connected to the same private network [187]. The goal of CCVPN is to provide users with analogous functionality within CCN.

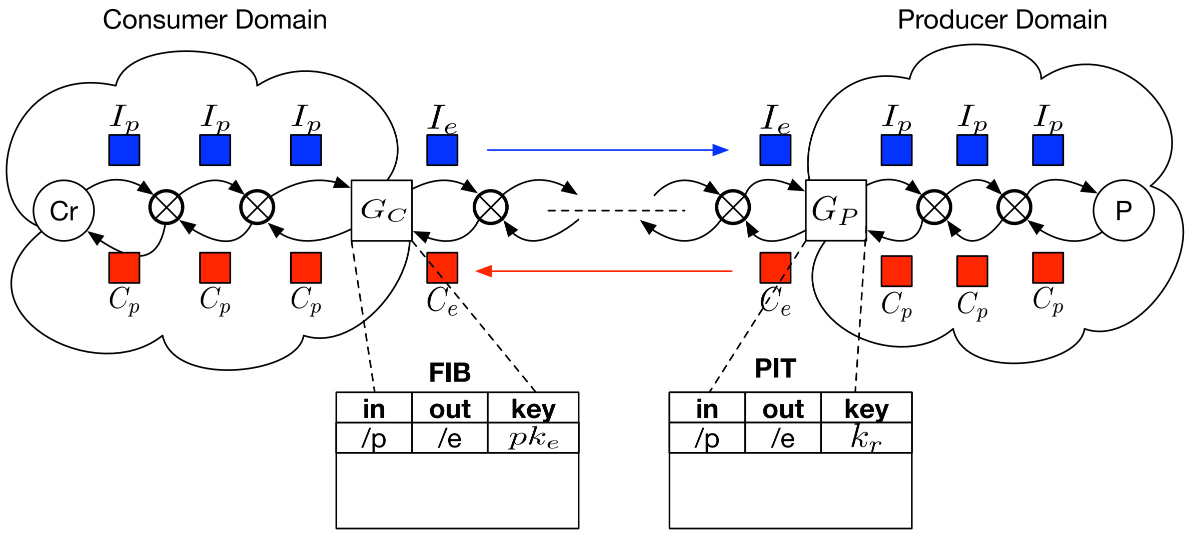


Figure 5.12: CCVPN connectivity architecture

CCVPN involves four types of entities:

- *Consumer:* issues an interest for content it wants to retrieve.
- *Producer:* entity that creates and publishes content.
- *Consumer-Side Gateway* (G_c)*:* end-point of the secure tunnel on the consumer side.
- *Producer-Side Gateway* (G_p)*:* end-point of the secure tunnel on the producer side.

A secure CCVPN tunnel is uni-directional, i.e., for a given tunnel, the roles of Consumer and Producer are fixed. (We use the term *"uni-directional"* to mean that all interests flow in one direction, while all content flows in the opposite direction.) As discussed later, G_c and G_p can run on the same platform to enable bi-directional communication composed of two uni-directional tunnels. However, for clarity's sake, we present them as separate logical entities.

Figure 5.12 shows the flow of a single interest and content exchange over CCVPN. Entities inside *Consumer* (*Producer*) domain form a physically interconnected private network. The goal is to create an overlay network that joins *Consumer* and *Producer* networks such that interest and content packets are only visible to entities within the respective domains.

G_c is an entity in *Consumer* domain that encrypts outgoing interests. Similarly, G_p decrypts and forwards incoming interests. When a content packet is forwarded in response to an interest, G_p decrypts incoming interests from G_c and forwards them in the *Producer* domain. G_p effectively serves as a proxy for the original consumer. When the intended content is returned to G_p, the latter encrypts (or encapsulates) the data before it exits the *Producer* domain and is forwarded to G_c. Finally, G_c decrypts the content packet and forwards it towards *Consumer*. We describe these steps in more detail below.

NOTE: We use the term "encryption" to denote *authenticated encryption*, i.e., encryption and decryption algorithms that ensure plaintext integrity.

Upon arrival of an interest I_p, G_c performs a FIB lookup to check whether the interest's name prefix is among those used for VPN communication. (We assume G_c's FIB is pre-configured with the list of prefixes that require VPN tunneling, i.e, prefixes associated with the producers in *Producer* domain in Figure 5.12.) If the FIB lookup succeeds, G_c obtains G_p's name (prefix) and public key (pk_e). G_c then runs Algorithm 14 to generate a new interest I_e that encapsulates the original interest I_p.

First, Algorithm 14 generates a fresh and random symmetric key (k_r), used later for *Content* encryption and decryption. Next, it retrieves G_p's name and public key from the FIB. It uses the public key to encrypt[16] the symmetric key k_r and the original interest I_p.

It then creates the new interest I_e referring to G_p's name with encrypted I_p as the payload.[17] I_e is routed towards G_p. Since the payload is encrypted with G_p's public key, only G_p can decrypt it to obtain I_p and k_r.

[16]In fact, for the sake of efficiency, hybrid encryption is used. For details on the hybrid encryption implementation refer to [291].

[17]Recall that, since I_e carries an encrypted form of I_p in its payload, the name of I_e has a unique identifier appended to it so as to prevent cache hits between G_c and G_p.

Algorithm 14 Consumer gateway interest encapsulation

```
Require: Original interest $I_p(N)$
Ensure: Encapsulated interest $I_e(N_{G_p})$
 1: $k_r \leftarrow_{\$} \{0,1\}^{\kappa}$
 2: $N_{G_p}$ = retrieveNameFromFIB($N$)
 3: $pk_e$ = retrievePKFromFIB($N$)
 4: $payload = Enc_{pk_e}(I_p(N)||k_r)$
 5: $I_e(N_{G_p})$ = createNewInterest($N_{G_p}$, $payload$)
 6: storeToPIT($I_e(N_{G_p})$,$k_r$)
 7: return $I_e(N_{G_p})$
```

Upon receipt of I_e, G_p verifies whether the interest name prefix matches one of its own. If so, it runs Algorithm 15, using its private key (sk_e) to decrypt I_e, which yields I_p and k_r. G_p then stores I_e and k_r in its PIT, as part of the entry for the pending interest I_p. In doing so, G_p acts as a proxy consumer.

Algorithm 15 Producer gateway interest decryption

```
Require: Encapsulated interest $I_e(N_{G_p})$, Private key $sk_e$
Ensure: Original interest $I_p(N)$
 1: $payload$ = getPayload($I_e(N_{G_p})$)
 2: $I_p(N)||k_r = Dec_{sk_e}(payload)$
 3: storeToPIT($I_p(N)$,$k_r$,$N_{G_p}$)
 4: return $I_p(N)$
```

I_p is forwarded inside the *Producer* domain until it reaches *Producer* or a cached copy of the target content at some router within *Producer* domain. In either case, C_p is forwarded to G_p. Upon receiving C_p, G_p does a PIT lookup using C_p's name, i.e, name referenced in I_p, to retrieve k_r and I_e. Next, G_p encrypts C_p with k_r, yielding encrypted content C_e with the name I_e, as shown in Algorithm 16.

Next, C_e is forwarded back to G_c using router state previously established by I_e. Upon receiving C_e, G_c invokes Algorithm 17: G_c performs a PIT lookup which matches C_e's name to the pending interest for I_e, and retrieves k_r. G_c then decrypts C_e with k_r and obtains C_p. Finally, C_p is forwarded to *Consumer* using router state set up by I_p.

Algorithm 16 Producer gateway content encryption

```
Require: Original content $C_p(N)$
Ensure: Encrypted content $C_e(N_{G_p})$
1: $k_r$ = retrieveKeyFromPIT($N$)
2: $N_{G_p}$ = retrieveNameFromPIT($N$)
3: $payload = Enc_{k_r}(C_p(N))$
4: $C_e(N_{G_p})$ = createNewContent($N_{G_p}$,$payload$)
5: return $C_e(N_{G_p})$
```

Algorithm 17 Consumer gateway content decryption

```
Require: Encrypted content $C_e(N_{G_p})$
Ensure: Original content $C_p(N)$, or $\perp$ on failure
1: $k_r$ = retrieveKeyFromPIT($N_{G_p}$)
2: $payload$ = getPayload($C_e(N_{G_p})$)
3: $C_p(N) = Dec_{k_r}(payload)$
4: if $C_p(N) == \perp$ then                          ▷ Decryption failed
5:     return $\perp$
6: else
7:     return $C_p(N)$
```

As mentioned earlier, in the context of bi-directional communication, the same device can run both G_p and G_c. The same holds for actual end-points, i.e., the same "box" can act as Consumer in one uni-directional VPN tunnel, and as a Producer in the opposite-direction uni-directional VPN tunnel. This applies to interactive-type communication, such as conferencing or remote login where both end-points of the tunnel act as a producer and a consumer.

We also note that CCVPN does not inhibit in-network content caching within domains. Outside of the domains, i.e., between G_c and G_p, interest names are sufficiently random and prevent cache hits. This is a side effect of the authenticated encryption mechanism used to encapsulate I_p and form I_e. Finally, the roles of G_c and Consumer (or G_p and Producer) can be collocated, as is the case with IP-based VPNs. This enables private one-to-one, many-to-one, and one-to-many communication.

A trivial variation of CCVPN is the case when G_c and G_p have a shared pre-installed symmetric key. The only difference is that Algorithms 14 and 15 use symmetric key encryption instead of PKE. We refer to this variation as *symmetric-key CCVPN*. This variant can be configured if gateway keys are manually installed. Alternatively, a key exchange protocol such as CCNxKE [228] can establish the gateway shared secret.

5.4.2 Threat Model and Analysis

This section discusses security considerations and properties of CCVPN.

Threat Model

We consider the worst-case scenario, where Consumer issues an interest that has not been cached in any router in either Consumer or Producer domains. Therefore, the interest travels all the way to Producer.

Adversary Goals and Capabilities. The goal of the adversary $\mathcal{A}$ is either: (1) to learn some information about the original interest I_p or original content C_p, or (2) to learn the identities of Consumer or Producer.

$\mathcal{A}$ is also considered successful if it impersonates Producer by faking a content packet. We assume that $\mathcal{A}$'s presence and activities are confined to the public networks, i.e., the Internet. In other words, $\mathcal{A}$ has no presence in either *Consumer* or *Producer* domain. Also, since they are part of *Producer* and *Consumer* domain, respectively, we assume that G_c and G_p are not compromised.

We allow $\mathcal{A}$ to perform the following actions:

- **Eavesdrop on traffic:** $\mathcal{A}$ can eavesdrop on any link (outside Producer and Consumer domains), thus learning packet contents and other characteristics, such as timing and sizes.
- **Compromise existing, or introduce new compromised, routers:** this means that $\mathcal{A}$ can inject, delay, and discard interest or content packets at will. If $\mathcal{A}$ compromises an existing router, it learns all private information, including private keys and cached content.

Security Analysis

In this section we analyze security of CCVPN. We aim to prevent $\mathcal{A}$ from achieving goals outlined in Section 5.4.2. Formally, this translates into semantic security of all traffic, except for whatever can be inferred by traffic analysis. Consequently, an off-path adversary can not forge content packets with non-negligible probability. Our analysis relies on arguments in the standard security model. It consists of assessing the security of the interest and content encapsulation algorithms. Moreover, we assume that all interest and content packets are *padded* to the standard MTU size between the gateways. (This is done in order to make packets indistinguishable.)

Definition 5.1. *An interest encapsulation algorithm Encapsulate*(I_p) *is indistinguishable iff, given any two interests* I_p^1 *and* I_p^2*, chosen by* $\mathcal{A}$*, and a randomly selected bit* b*,* $\mathcal{A}$ *has* $1/2 + \epsilon(\kappa)$ *probability of guessing the value of the bit* b *when given* $I_e^b = \text{Encapsulate}(I_p^b)$*, where* ϵ *is a negligible function in security parameter* κ*.*

Theorem 5.1. *Let* $\mathsf{Encapsulate}_{pk}(I_p)$ *denote the interest encapsulation routine described in Algorithm 14. If* Enc_{pk} *is a CPA-secure public-key encryption scheme, then* $Encapsulate_{pk}(I_p)$ *is indistinguishable.*

Proof; Suppose that Claim 5.1 is false. Then there is a polynomial adversary $\mathcal{A}$ capable of guessing b in Definition 5.1 with non-negligible advantage, when given $I_e^b = \mathsf{Encapsulate}_{pk}(I_p^b)$ with $b \leftarrow \{0,1\}$ chosen at random. We show that if $\mathcal{A}$ exists, it can be used to construct another polynomial adversary $\mathcal{A}^{\mathsf{CPA}}$ which breaks CPA-security of Enc_{pk}. $\mathcal{A}^{\mathsf{CPA}}$ plays the CPA-security game with a challenger sending it two messages: m^0 and m^1. Following the CPA-security game, the challenger randomly chooses a value for the bit $b' \leftarrow \{0,1\}$ and gives back $C = Enc_{pk}(m^{b'})$ to $\mathcal{A}^{\mathsf{CPA}}$. To break CPA-security $\mathcal{A}^{\mathsf{CPA}}$ must guess the value of the bit b' with non-negligible advantage. For that purpose $\mathcal{A}^{\mathsf{CPA}}$ queries the challenger for encryptions of m^0 and m^1 ($c^0 = Enc_{pk}(m^0)$ and $c^1 = Enc_{pk}(m^1)$) and construct two interests $I_e^0 = \mathsf{createNewInterest}(Gp_{name}, c^0)$ and $I_e^1 = \mathsf{createNewInterest}(Gp_{name}, c^1)$, using the same $\mathsf{createNewInterest}$ function used by algorithm 14, which is public (note that Gp_{name} is also public). Finally, $\mathcal{A}^{\mathsf{CPA}}$ gives I_e^0 and I_e^1 as input to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Since under our assumption $\mathcal{A}$ guesses the bit b with non-negligible advantage, then $\mathcal{A}^{\mathsf{CPA}}$ breaks the CPA-security of Enc_{pk}. Since this violates the hypothesis of Claim 5.1, $\mathcal{A}$ cannot exist.

Definition 5.2. *A content encapsulation algorithm $Encapsulate(C_p)$ is indistinguishable iff, given any two content packets: C_p^1 and C_p^2, chosen by $\mathcal{A}$, and random bit b, $\mathcal{A}$ has $1/2 + \epsilon(\kappa)$ probability of correctly guessing b when given $C_e^b = Encapsulate(C_p^b)$, where ϵ is negligible function in security parameter κ.*

Theorem 5.2. *Let $ContentEnc_{k_r}(C_p)$ denote the content encapsulation routine described in Algorithm 16. If $EncryptThenMAC_{k_r}$ is an authenticated encryption (i.e., CCA-secure) symmetric-key scheme used to construct $ContentEnc_{sk}$, then:*

1. *$ContentEnc_{k_r}(C_p)$ is an indistinguishable content encapsulation algorithm;*
2. *$\mathcal{A}$ negligible probability of generating a valid fake encapsulated content I_c'*

Proof Follows directly from the definition of CCA-security for authenticated encryption and from the same argument as in the previous proof. We therefore claim that nothing is leaked in encapsulated interest or content packets as they are forwarded between gateways. Since the only information in these packets is G_p and a random nonce derived from the encrypted payload of I_e, $\mathcal{A}$ learns nothing about I_p or C_p, or the identities of Consumer or Producer.

Additional Considerations

Unlinkability between Consumer domain and encapsulated packets: An argument similar to those in Sec. 5.4.2 can be used to show that, in a setting where multiple consumer domains establish tunnels to a given producer domain gateway Gp, an outside observer can not correlate (with non-negligible advantage) a given encapsulated packet (interest or content) to the domain where the original interest was issued.

In fact, this is an advantage of CCVPN when compared to VPNs over IP. In IPsec, for the sake of forwarding, each encapsulated packet carries source and destination addresses of correspondent tunnel end-points. Thus, an outside observer can easily determine the two domains that communicate encapsulated data. In CCVPN, encapsulated content packets are forwarded to the consumer domain according to routers' PITs, while interests carry no source addresses. Thus, observing encapsulated packets gives no information about the correspondent consumer domain.

Gateway-to-Gateway Authentication: In CCVPN, any host that has G_p's public key can initiate a tunnel with G_p. In other words, our design does not include any authentication between tunnel end-points. We claim that $G_p - G_c$ authentication is not required since it not all application scenarios need it. For example, a producer offers its content to any consumer while requiring that the latter request and receive such content *privately.* In this case, there is no need for G_c to authenticate itself to G_p.

Another CCVPN use-case is where two physically separated private networks, e.g., offices of the same company in different countries, need to behave as a unified network. In that case, it is necessary to prevent an extraneous G_c, from connecting to G_p. Standard host-to-host CCN security mechanisms can be used for mutual authentication between G_p and G_c, prior to VPN communication. We leave the evaluation and specification of gateway-to-gateway authentication protocols for future work. However, we note that CCNxKE [228], a CCNx-compliant key exchange protocol, supports mutual authentication and could be used for this purpose.

Denial of Service: Since CCVPN gateways face the public network they are clearly exposed to DoS attacks. A DoS attack on G_p might involve flooding it with fake encapsulated interests, while a DoS attack on G_c would consist of flooding it with an enormous amount of encrypted content packets. The former is more dangerous, since interest decapsulation involves a public-key decryption operation. If symmetric-key algorithms were used to encrypt interest and content packets, efficacy of DoS attacks would be reduced, though not negligible. We defer DoS counter-measures to future work.

5.4.3 Performance Assessment

We now discuss performance aspects of CCVPN.

State Consumption

CCVPN has an immediate impact on a gateway's FIB and PIT sizes. (Cache size remains unaffected since only decapsulated content objects are ever cached.) Let F_S be the total size of a standard CCN router FIB in bytes, and N_F – number of FIB entries. For simplicity, we assume that each name prefix in the FIB has a constant size of 64B. (We expect this to be a reasonable upper bound in practice). Thus, $F_S = N_F \times\ s$, where s is the size of each FIB entry; s includes a name prefix (64B) and a bit-vector that identifies matching links for the interface. We assume that a gateway has 128 links, which is a safe upper bound. Therefore, $s = 80\text{B}$ $(= 16 + 64\text{B})$. Now consider FIB size F_G for a CCVPN gateway. Some entries of a FIB will point to "private" prefixes, i.e., other domains, and therefore would be of larger size to account for the corresponding prefix and key material. For both public- and symmetric-key encryption, key size is the same: 32B [19]. Therefore, taking into account FIB entry prefix key, target domain prefix, e.g., G_p, encryption key, and corresponding bit-vector, the total size of one "private" FIB entry is 176B, meaning that $F_G = 176N_F\text{B}$. Thus, in the worst case, CCVPN FIB is at most $F_G/F_S = 176/80 = 2.2$ times larger than the standard FIB. In practice, however, we expect growth factor to be much smaller, since the fraction of public-to-private FIB entries would be non-zero.

We apply the same analysis to PIT size. A standard PIT entry includes a complete name and ingress bit-vector.[18] A gateway PIT entry would contain the same elements as a standard PIT entry, plus a symmetric key (32B), a nonce (12B), and an encapsulation name (64B + 32B) – the name of an encapsulated interest that includes an additional 32B payload ID segment to identify the encapsulated value in the payload. Let P_S and P_G be the sizes of the standard and CCVPN gateway PITs, respectively, and let N_P be the number of PIT entries in each individual table. Based on the above discussion, and assuming that a name is at most 64B, a standard PIT entry is 80B, while a gateway PIT entry is 204B. Therefore, in the worst case, CCVPN PIT would be at most $P_G/P_S = 204/80 = 2.55\text{B}$ larger than the standard PIT. Assuming a steady state size of approximately $1e^5$ entries [75], the PIT would be 20.4MB – well within the capacity of modern routers.

[18]They may also include optional KeyId and ContentId. However, since they are included in the gateway PIT as well, we omit them from this analysis.

Processing Overhead

A CCVPN gateway adds some new steps to the data path of a packet. The main computational burdens are packet encapsulation and decapsulation. In the public-key variant of CCVPN, gateways process interests using public-key encryption, while content – using symmetric-key encryption. Let $T^P_E(n)$ and $T^P_D(n)$ be respective times to encrypt and decrypt nB of data using a public-key encryption scheme. Similarly, let $T^S_E(n)$ and $T^S_D(n)$ be respective times to encrypt and decrypt the same amount of data with a symmetric-key encryption scheme. Then, the latency of a single interest-content exchange is increased by:

$$T = T^P_E(n_I) + T^P_D(n_I) + T^S_E(n_C) + T^S_D(n_C)$$

where n_I and n_C are original interest and content sizes, respectively. As a rough estimate, [4] lists the cost of AES-GCM to be $2.946\mu s$ for setup followed by 102MiB/second, on an Intel Core 2 1.83 GHz processor running Windows Vista in 32-bit mode (with AES ISA support). For packets that are at most $1,500$B, total processing time is $\approx\ 17\mu s$. Moreover, public-key encryption and decryption operations are always at least as expensive; thus, total latency is increased by at least $T = 4 \times 17\mu s = 68\mu s$. In comparison to network latency for a single packet, this might be unnoticeable, though for a steady arrival state of $\approx\ 1e^5$, it would lead to an unstable system that would quickly overflow. (This is because $65\mu s \times 1e^5 = 6.8s$.) Therefore, there is an upper bound on the number of private packets a gateway can process per second. This bound is entirely dependent on system configuration and network conditions.

Another performance issue stems from gateways not being able to process packets without allocating memory. Specifically, each packet requires either an encryption or decryption. However, since this cannot be done entirely in-place, the gateway must allocate some memory for every packet, e.g., to store the MAC tag, to account for ciphertext expansion, or to apply

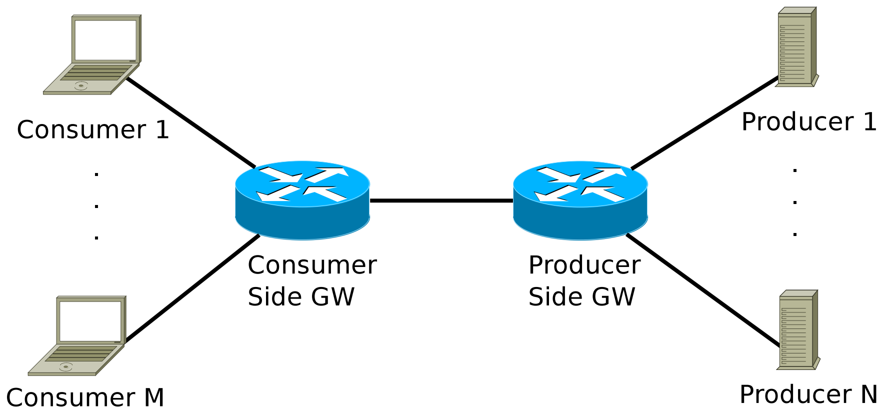
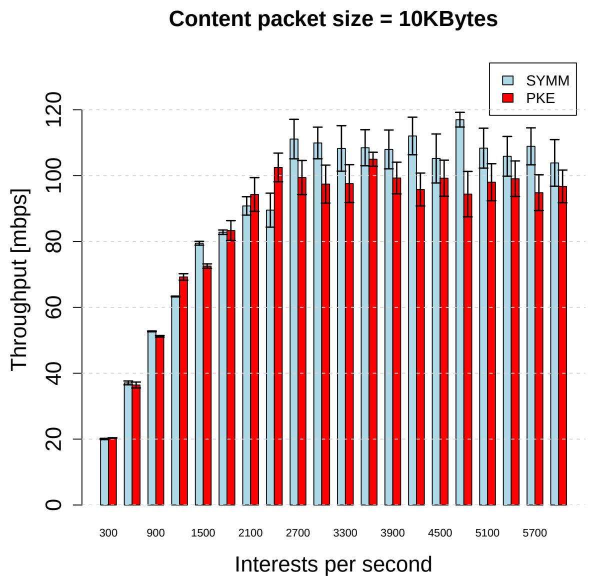


Figure 5.13: Testbed network topology with M consumers and N producers

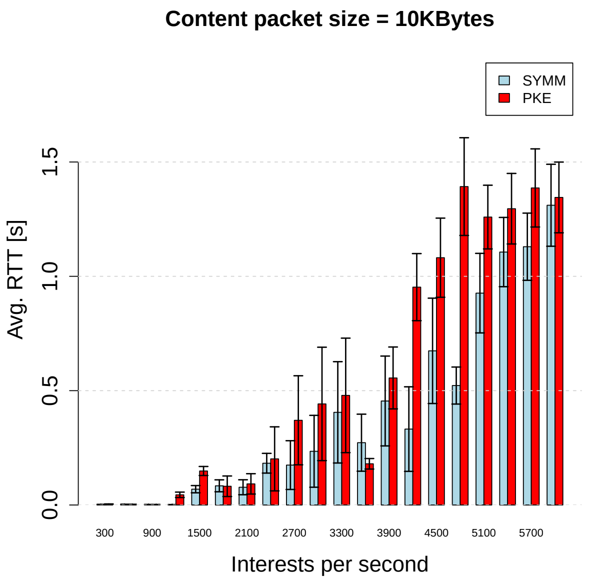
padding. This overhead can outweigh that of cryptographic computations if packet arrival rate is high enough. Therefore, when implementing CCVPN, special care must be taken to ensure that memory allocation is minimized or avoided.

5.4.4 Implementation and Performance Assessment

We implemented CCVPN as a network-layer service running on the gateways of private networks that compose the VPN (see Figure 5.12). The implementation uses the CCNx software stack [1] and libsodium cryptographic library [19]. Both are publicly available and written in C. For the public key version of CCVPN, we use the libsodium public-key authenticated encryption API in the interest encapsulation and decapsulation routines (Algorithm14, and Algorithm15 of Section 5.4.1). Internally, these perform a x25519 [55] key exchange to derive a symmetric key that is then used to encrypt and authenticate the interests in transit. AES256-GCM [108] is used to encrypt and authenticate content packets (Algorithm 16, and Algorithm 17 of Section 5.4.1). Recall that symmetric keys used to encrypt and authenticate content packets are generated and sent together with the encapsulated interest in Algorithm 14. In the symmetric key version of CCVPN, both



(a) Throughput



(b) Avgerage RTT

Figure 5.14: CCVPN performance with one consumer and one producer

interests and content objects are encapsulated with AES256-GCM under the assumption that gateways already share a symmetric key.

Experiments presented in this section were performed on an Intel Core i7-3770 octa-core CPU @3.40GHz, with 16GB of RAM, running Ubuntu 14.04 LTS. Content payload size was fixed to 10KB. In each experiment, gateway processes, i.e., G_c and G_p processes, were assigned high priority and ran on a single core. Our results indicate that interest encapsulation (Algorithm 14) with public and symmetric key encryption cost approximately 700μs and 520μs, respectively. Decapsulation via Algorithm 15 with public and symmetric key decryption cost approximately 640μs and 400μs, respectively. Content encapsulation and decapsulation with Algorithms 16 and 17, respectively, cost approximately 550μs and 460μs.

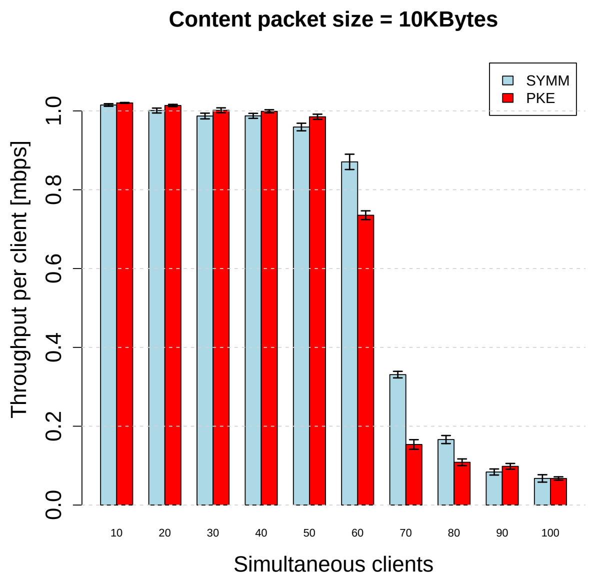
To evaluate the impact of CCVPN's cryptographic overhead on overall network performance, we measured network throughput and request-response RTT under various topology settings. In our testbed, G_p and G_c are directly connected. N producers are connected to the former, and M consumers – to the latter, as illustrated in Figure 5.13. We consider three variations for values of $[M, N]$:

- **One consumer and one producer** $[1, 1]$: We slowly increase interest issuance rate until we can to determine maximum network throughput and impact on RTT, as congestion increases.
- **Multiple consumers and one producer** $[M, 1]$: We fix interest issuance rate so that each consumer requests $\approx$ 1 mbps of content, and gradually increase the number of consumers, until throughput per consumer starts to decrease, i.e., until congestion starts to occur. All interests coming from all consumers are served by a single producer.
- **Multiple consumers and multiple producers** $[M, N]$: We gradually increase the number of consumers. However, we also increase the number of producers by the same amount in each round, i.e., $M = N$. The number consumers and producers is increased until congestion is detected.

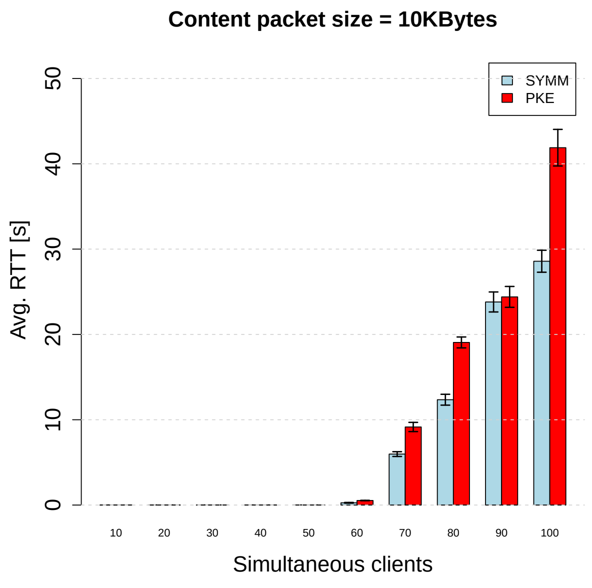
In all of the these settings, every interest is a unique request for a unique content. Therefore, experimental results reflect throughput and RTT in the worst-case scenario, i.e., no content caching at the gateway. The results are presented with 95% confidence intervals.

Figure 5.14 shows network performance when $[M, N] = [1, 1]$ as consumer's request rate increases. The network achieves maximum throughput of 100 mbps in the public key version and slightly higher throughput of 110 mbps in the symmetric key version. Average RTT per message starts to increase as interest issuance rate approaches maximum network throughput – a sign of congestion.

Results for multiple consumers requesting content from a single producer ($[M, 1]$) are shown in Figure 5.15. When requesting at rate of 1 mbps, each consumer achieves near-optimal throughput (1 mbps) when less than 50 clients request content simultaneously. With at least

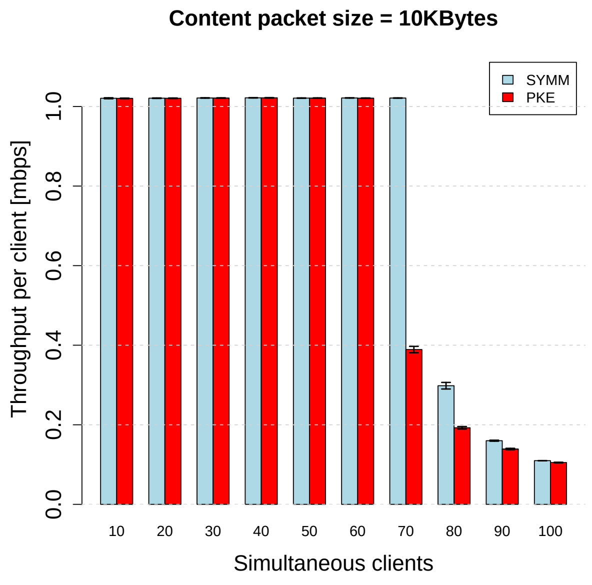


(a) Throughput

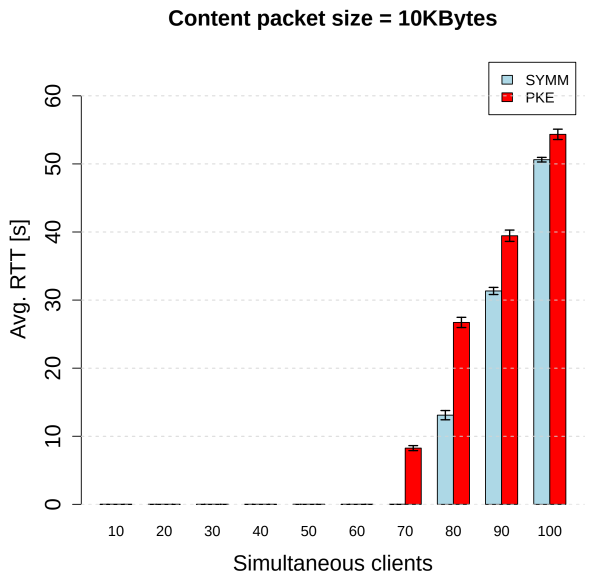


(b) Average RTT

Figure 5.15: CCVPN performance with multiple consumers and one producer



(a) Throughput



(b) Average RTT

Figure 5.16: CCVPN performance with multiple consumers and multiple producers

60 clients average RTT starts to increase due to congestion, and average throughput for each client gradually goes down.

Since a CCN producer must sign every content[19], congestion observed in Figure 5.15 might be influenced by the overhead of having a single producer signing a large number of interests, in addition to gateway cryptographic overhead. To evaluate this effect in Figure 5.16, we show average throughput and RTT in the $[M, N]$ scenario, where $M = N$ and each consumer requests from a fixed producer at a rate of 1 mbps. Here, results are slightly better. The network offers requested throughput (1 mbps per client) with at most 60 nodes, in the public key version, and at most 70 nodes, in the symmetric key version.

Discussion

CCVPN exhibits moderately good results with respect to network load capacity, considering overhead incurred by deploying secure tunnels over the CCN architecture. With gateway processes running each on a single core of a single processor, the VPN can provide reasonable throughput to at most 70 consumers. These performance results represent a lower bound that can be improved in several ways, such as:

1. **Implementation optimization:** CCNx software stack is an on-going research project and it prioritizes functionality over performance. We believe that CCVPN performance can be significantly improved by optimizations that do not rely exclusively on CCVPN design, but also in CCNx software.
2. **Distributed and parallel processing:** In a real deployment scenario, a large organization that wants to use CCVPN would have dedicated network devices running the gateway service, possibly in multiple cores. Also, multiple VPN gateways can share the load in a large organization.

[19]Unless the content can be requested by hash.

3. **Caching:** Content caching is a major advantage of CCN, as compared to IP. Our experiments evaluated worst-case scenarios, i.e., consumers always request distinct content packets and caching does not occur. In a real-world deployment, popular content (inside the VPN) would be cached, thus increasing throughput and reducing RTT.

5.5 Transparent Packet Security

Confidentiality is an application-layer function since it requires some form of authorization. The same is *not true* for privacy. As of late, privacy is considered a necessary feature for emerging Internet protocols due to growing evidence of large-scale network packet interception and eavesdropping by unauthorized entities [82]. Specifically, pervasive eavesdropping and monitoring is now considered an attack on privacy [114]. To combat these attacks, ubiquitous and opportunistic encryption protocols are being standardized for IP-based protocols such as TCP and DNS, e.g., [60, 61, 347]. Consequently, any viable IP alternative, especially CCN, should deal with privacy issues an equally application-agnostic, end-to-end manner. To the best of our knowledge, there has been no concrete work towards this goal in the CCN community.

In this section we present TRAPS, a mechanism that enables transparent packet security for CCN that, unlike traditional end-to-end encryption mechanisms, does not prohibit packet caching in the network. TRAPS is built on the premise that if one knows an *application* name of content, then it can obtain and decrypt data. Otherwise, without knowledge of the application name, data remains encrypted and secure. TRAPS uses application names to create obfuscated network counterparts and encrypt the corresponding content. Thus, TRAPS can be implemented entirely at end hosts, i.e., consumers and producers, as is done for protocols such as tcpcrypt [61]. Moreover, TRAPS can be extended with stronger end-to-end encryption that makes knowledge of an application name insufficient to decrypt content.

Our contributions are three-fold:

- The first lightweight, application-transparent "transport" security protocol for CCN.
- End-host network stack modifications necessary to support TRAPS.
- Analysis of TRAPS security subject to passive eavesdroppers and the performance overhead incurred by end-hosts.

5.5.1 Separating Privacy and Confidentiality

Privacy and confidentiality are pervasive problems in CCN. While they may seem to be orthogonal issues, they stem from the fact that some packet fields are not encrypted unless explicitly done so by applications. Cleartext packet data (and metadata) can reveal details about producers and intent of the data contained in content objects. It can also leak information about content requested by particular consumers. One goal of TRAPS is to provide a transparent mechanism to deter or prevent such types of inference. In this section, we review previous proposals of content confidentiality and problems of name privacy to motivate the need for TRAPS.

Confidentiality Conundrum

Content confidentiality prevents unauthorized parties from accessing protected content. This can be achieved with some form of CBAC or IBAC. (See Chapter 4 for examples.) Other (albeit futile) possibilities include specifying little or no cache time for content objects. Assuming routers are honest, this means that all interests would be routed to producers, who could then determine access rights to content on a per-interest basis. This would require consumers to provide some form of unforgeable identity or authentication token that could be used by the producer to make this authorization decision. The effect of caching on limited availability can therefore be perceived as a very weak network-layer confidentiality enforcement mechanism.

CBAC is the most popular access control design in CCN. However, content encryption need not necessarily be done by applications. CCN lacks an encryption mechanism that (a) does not involve or require any application input and (b) is implemented above the network layer. To the best of our knowledge, there is no protocol for enabling such *transparent* encryption between consumers and producers. We claim that such a protocol is both necessary and timely given that CCN and related architectures are maturing and privacy (via encryption) is paramount.

Pitiful Privacy

Privacy is often an overlooked property in CCN. Many applications rely on well-formed, deterministically generated, and meaningful names to ease developer burden. For example, in NDN-RTC [162], audio segments are assigned names with the following format: `<prefix>/ndnrtc/user/<username>/streams/audio0/<bitrate>/...`. This leaks an unnecessary amount of information about the contents and subjects of a conversation. Overcoming this privacy challenge has deep implications on how names are conveyed to the network. Ghali et al. [146] confirmed (often unstated) intuition that names must be indistinguishable from random strings in order to guarantee some measure of privacy. Their analysis shows that any technique which can decouple application names, such as the example above, from those which are carried in packets will improve privacy. This separation is another motivating principle for TRAPS.

5.5.2 Threat Model

As the name suggests, the goal of TRAPS is to transparently encrypt packets to improve data privacy – *not confidentiality*. Our adversary $\mathcal{A}$ is one which attempts to learn the identity (application name) of encrypted content. $\mathcal{A}$ is active and has the ability to compromise any router in the network. $\mathcal{A}$ can perform any action on compromised routers.

Moreover, we assume $\mathcal{A}$ can compromise any producer at will. In doing so, $\mathcal{A}$ may view all data under control of such a producer. Once a producer is compromised, all of its data is said to be *exposed*. $\mathcal{A}$ may also compromise consumers to control their traffic and view past data that was received. We call data visible to $\mathcal{A}$ after a consumer has been compromised *exposed data*. An important criteria for TRAPS is that *exposed data does not cause immediate harm for non-exposed data*, i.e., data which has not yet been requested by the compromised consumer or data that is owned by a non-compromised producer.

Unlike standard encryption protocols, such as TLS [107], wherein there is assumed to be a global PKI or possibly shared secrets between trusting parties, we choose to restrict our notion of "transparent" to one in which we rely on neither. As such, security of TRAPS cannot depend on pre-configured certificates or previously exchanged secrets. (We explicitly omit a design requiring key exchange protocols for TRAPS since it implies an association. As we will discuss later, a goal of TRAPS is to not break the data-oriented nature of CCN.) Rather, TRAPS depends on implicitly shared knowledge between consumers and producers. Since there are no cryptographic secrets shared a priori, TRAPS security is a function work expended by $\mathcal{A}$ to learn this implicit information. By default, TRAPS is not intended to be computationally or information-theoretically secure such as is the case with TLS [107]. A powerful enough adversary could break it, but at a cost that is a *design parameter* for the protocol.

5.5.3 Transparent Packet Security

In this section we describe TRAPS and how it sets out to achieve security goals outlined above. First, we clearly state TRAPS requirements:

R-1. TRAPS should be completely transparent to applications. Neither consumers nor producers should be required to opt in to the protocol. However, TRAPS may be exposed to applications to enable stronger security properties.

R-2. TRAPS should not require consumers or producers to share any cryptographic secrets or perform any sort of key establishment.

R-3. TRAPS should not break the data-centric and request-based model of CCN. As a result, features such as caching should still work with TRAPS.

R-4. TRAPS security should be a tunable parameter that has a reasonable default and, if desired, can be decided upon by producers and conveyable to consumers. Moreover, consumers and producers should be able to opt-out of TRAPS if desired.

The core idea of TRAPS is that implicit knowledge of a name is considered a shared secret between a consumer and producer. Consumers know a priori (or learn through some out of band mechanism) content names, whereas producers know what (static) content they provide and are willing to publish. In TRAPS, knowing a name is sufficient to decrypt a packet. Without a name, data remains encrypted. Thus, names can be viewed as a type of password needed to access content. Cryptographic secrets can be derived from names, and later used to protect both the wire-encoded name and data. An eavesdropper who sees a protected request and response learns very little. Moreover, they would have to expend a non-negligible amount of effort, in terms of computation and memory resources, to learn the underlying data.

Protocol Overview

TRAPS builds on a couple of new of cryptographic primitives. The first of which are memory-hard functions (MHFs) [249]. A MHF is a function which, on a random access machine, requires $S(\lambda)$ space and $T(\lambda)$ operations to compute, where $S(\lambda) \cdot T(\lambda) \in \Omega(\lambda^2)$ and λ is the security parameter. Optimally, a MHF requires just as much space as it does operations. MHFs are intentionally expensive to compute since they are meant to deter one from computing massive numbers in parallel with custom hardware.

Another primitive we rely upon is so-called convergent or message-locked encryption (MLE) [52]. A MLE scheme is one where the (symmetric) key used to encrypt and decrypt a message is derived from the message itself. In [52], the symmetric key k for message M is derived as $k = H(M)$, where H is a suitable cryptographic hash function, such as SHA-256. To encrypt a message M, one then computes a tag $T = H(k)$ and ciphertext $C = \mathsf{Enc}_k(M)$, and outputs (C, T). Decrypting and verifying (C, T) requires one to decrypt C, derive tag T', and check if $T' == T$. MLE schemes are deterministic and therefore enable secure de-duplication; identical messages will be encrypted to identical ciphertexts. TRAPS exploits this property for a large class of traffic – specifically, *static* data.

At a high level, TRAPS can be viewed as a composition of a password-hashing and message-locked encryption. It uses name obfuscation (via hashing) and (message-locked) content encryption to protect names and data, respectively. Name obfuscation uses a cryptographic (or memory-hard) hash function F to map meaningful names to random correspondents as described by Ghali et al. [145]. (The need for memory-hard variants of F is discussed in Section 5.5.4.) Content encryption uses secret-key cryptographic algorithms ($Enc_k(\cdot)$ and $Dec_k(\cdot)$) for efficiency. We also make use of a key derivation function (KDF), e.g., HKDF [194].

TRAPS is configurable and accepts the following inputs: Security parameter λ, obfuscation function $\mathsf{F} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, and a salt generation and rotation functions, s_G and s_R, respectively. We denote a specific configuration of TRAPS as $\mathsf{TRAPS}(\lambda, \mathsf{F}, s_G, s_R)$.[20]

The complete end-to-end operation of $\mathsf{TRAPS}(\lambda, \mathsf{F}(\cdot), \perp, \perp)$ – the default configuration – is shown in Figure 5.17. (If not specified, $\lambda = 256$, F = SHA-256, and $s_G = s_R = \perp$.) Recall that, to obtain data D with name N, denoted $D(N)$, consumers issue a request (interest) for $D(N)$, denoted $I(N)$. We refer to the i-th segment of N as N_i. If $D(N)$ is chunked into n pieces, we denote $D_i(N)$ as the i-th piece. The unique identifier for $D(N)$ is its cryptographic hash digest, denoted $D_{ID}(N)$. Detailed descriptions of each step in TRAPS are given in Algorithms 18, 19, and 20, and a complete depiction of TRAPS encryption is shown in Figure 5.18. Observe that if a different consumer Cr' application issues an interest for the same name N, and this interest is routed along a path containing a router which has cached the obfuscated content object $C(\bar{N})$, the content object will be returned to Cr' as expected. Since $C(\bar{N})$ contains the name $\bar{N}$ and nonce r, Cr' can decrypt $C(\bar{N})$ before passing up $C(N)$ to the application. Thus, TRAPS can still exploit caches. An alternative strategy would have been for consumers to provide their public key in each interest, similar to DNSCurve [56]. Producers could encrypt the random content encryption key using this public key and return it in response. This, however, would invalidate caches.

TRAPS is transparent to the network. Only the producer and consumer applications see application names; all network entities, such as routers, deal only with obfuscated network names and encrypted content objects. Moreover, since the translation is deterministic, router processing is unaffected. (Equality on N is identical to equality on $\bar{N}$.) Applications may configure TRAPS by choosing λ, F, s_G, and s_R as needed.

[20]When not needed, we omit these parameters for presentation clarity. Also, by default, $s_G = s_R = \perp$, meaning that the salt is the empty string and never changes. If $s_G = \perp$, then it simply returns an empty string upon any input.

1. Consumer Cr's application issues interest $I(N)$ to its network stack.
2. Cr's network stack computes obfuscated interest $I(\bar{N}) = \mathsf{Obfuscate}(\mathsf{F}, s_G, I(N))$. ($\mathsf{Obfuscate}$ is detailed in Algorithm 18.) Cr issues $I(\bar{N})$ to the network.
3. Routers forward $I(\bar{N})$ to P.
4. P's network stack recovers N from $I(\bar{N})$ by looking up N in a table that maps all obfuscated names to their application representations.[21] P's stack then forwards N to the application.
5. P's application returns $C(N)$ to its network stack.
6. P's network stack computes encrypted content object $C(\bar{N}) = \mathsf{EncryptContent}(C(N))$. If required, $C(\bar{N})$ is then signed. P issues $C(\bar{N})$ to the network.
7. Routers forward $C(\bar{N})$ to Cr.
8. Cr's network stack computes $C(N) = \mathsf{DecryptContent}(C(\bar{N}))$ and passes it up to Cr's application.

Figure 5.17: TRAPS protocol summary

Based on these algorithm descriptions, the only state needed for TRAPS is a table that maps obfuscated names to their original counterparts. For consumers, this table size is of order equal to the number of outstanding interests. An entry is created for every issued interest and then subsequently removed when the corresponding content object is returned and consumed. Insertion, search, and deletion in this table are all $\mathcal{O}(1)$ operations if a hash table implementation is used. In addition to this state, the component needs a suitable source of entropy to derive content encryption keys. For producers, this table is of size proportional to the total number of published content objects.

Algorithm 18 $\mathsf{Obfuscate}(\mathsf{F}(\cdot), s_G, I(N))$

```
1: $s = s_G(\mathsf{now}())$
2: $\bar{N} = []$
3: for $i = 1 \to |N|$ do
4:     $\bar{N} = \mathsf{Append}(\bar{N}, \mathsf{F}(N_1 || \cdots || N_i || s))$
5: $\mathsf{InterestMap}[\bar{N}] = I(N)$
6: return $I(\bar{N})$
```

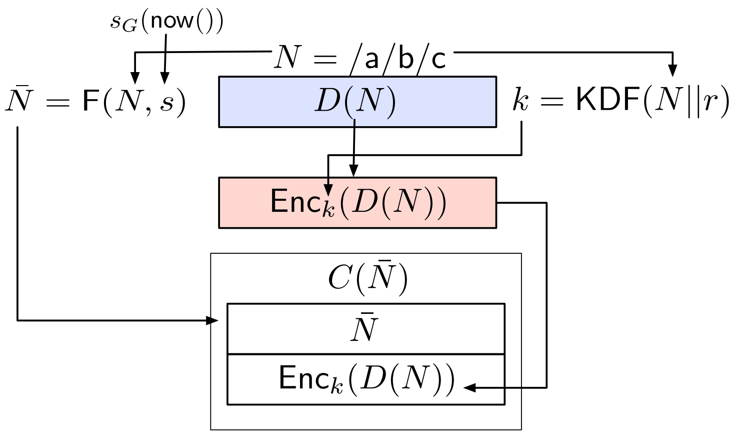


Figure 5.18: TRAPS translation and construction procedure

Algorithm 19 EncryptContent($C(N), \lambda$)

```
1: $r \leftarrow \{0,1\}^\lambda$
2: $k \leftarrow \mathsf{KDF}(N||r)$
3: $\bar{\mathsf{payload}} = Enc_k(D(N))$
4: $C(\bar{N}) = (\bar{N}, \bar{\mathsf{payload}}, r)$
5: return $C(\bar{N})$
```

Algorithm 20 DecryptContent($C(\bar{N})$)

```
1: $I(N) = \mathsf{InterestMap}[\bar{N}]$
2: $k \leftarrow \mathsf{KDF}(N||r)$
3: $\mathsf{payload} = Dec_k(C(\bar{N}).\bar{\mathsf{payload}})$
4: $C(N) = (N, \mathsf{payload})$
5: return $C(N)$
```

Static Content

For static content, it is common for a consumer to request N with $D_{ID}(N)$. Doing so requires knowledge of $D_{ID}(N)$ a priori. This is typically obtained in a Manifest, denoted Manifest($D(N)$). As described in Chapter 3, consumers first fetch the manifest pertaining to a collection of static content objects and then use its contents to subsequently request constituent chunks by their unique identifier. TRAPS can be also applied to each content object chunk in a manifest. However, there is a subtle detail to consider here. Since chunks potentially share locators (names), they will also use identical encryption keys. Thus, we must convey a proper IV or nonce for each one (depending on the encryption mode of operation) such that encryption remains secure.[22] Also, if an encryption key is derived from N, then each encrypted chunk becomes forever bound to N. This prevents de-duplication of nameless chunks that move around under different locators.

Therefore, to support de-duplication even when N (the locator) changes, encryption keys must be based on something else. To address this problem, we turn to MLE. In particular, we let the encryption key for a chunk be derived from its application data contents and, optionally, a name as well. This name can be N or $\bar{N}$, depending on how restrictive are the requirements for access to $D(N)$. Let $\mathsf{KeyGen}(\cdot, \cdot)$ be a function that takes as input $D(N)$ and (optional) name $N \in \{\perp, \mathbb{N}\}$ to compute an encryption key k. A key $k = \mathsf{KeyGen}(D(N), \perp)$ encrypts content such that it can be accessed with any locator. A key $k = \mathsf{KeyGen}(D(N), N)$ will bind k to locator N. Lastly, a key $k = \mathsf{KeyGen}(D(N), \bar{N})$ binds k to an ephemeral, obfuscated locator. The exact derivation mechanism to be used is a design choice for producers, provided they convey this decision to consumers. By default, the $k = \mathsf{KeyGen}(D(N), \perp)$. With k, the producer can then encrypt each chunk $D_i(N) \in D(N)$ in such a mode that permits random access decryption. (This is necessary so that chunks may be received out of order or in partial without preventing their use.) For integrity reasons, the producer should also generate a tag as outlined in Section 5.5.3. This tag can be included in content objects in place of a signature.

The final part of this approach is to convey $D_{ID}(N)$ to consumers, since a consumer cannot derive a hash for content to which it does not have access. For this, we use $\mathsf{Manifest}(D(N))$. FLIC manifests already carry $D_{ID}(N)$ as additional metadata. Thus, after retrieving a FLIC manifest, a consumer can derive decryption key(s) for constituent chunks. Figure 5.19 shows this manifest-based construction visually.

[22]Reusing a key and nonce pair with AES-GCM has disastrous consequences, see e.g., [64].

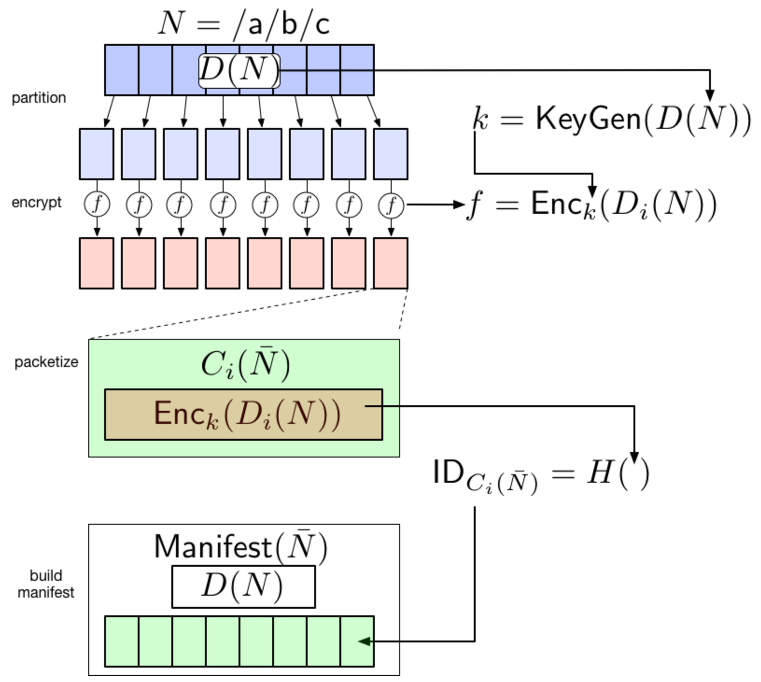

$N = /a/b/c$
$D(N)$
partition
$k = \mathsf{KeyGen}(D(N))$
encrypt
$f = \mathsf{Enc}_k(D_i(N))$
packetize
$C_i(\bar{N})$
$\mathsf{Enc}_k(D_i(N))$
$\mathsf{ID}_{C_i(\bar{N})} = H()$
build manifest
$\mathsf{Manifest}(\bar{N})$
$D(N)$

Figure 5.19: TRAPS hash-based construction procedure

Dynamic Content

The protocol described in Sections 5.5.3 and 5.5.3 assumes that all content is static. This enables producers to precompute data to map network names to their application counterparts. This is not possible for dynamic content due to preimage-resistant nature of F.

There are (at least) two ways to support dynamic content with TRAPS. The first is to additionally encrypt N, with application-specific segments, under the producer's public key pk_P. This encrypted name could be included in the payload of the interest so that routing still occurs on $\bar{N}$. When an interest $I(\bar{N})$ of this form arrives at a producer P, the latter can decrypt $I(\bar{N})$'s payload using sk_P to recover N. To acquire pk_P without first communicating with P, consumers could use a key-resolution service such as CCN-KRS [209]. A second alternative is for consumers to establish a session using, e.g., the CCNxKE protocol, over which to transfer encrypted dynamic content. Note that this approach requires consumers and producers to opt-out of TRAPS so that name transformations are not inadvertently applied.

Discovery and Interoperability

TRAPS relies on consumers knowing the name of content they wish to access. Encrypted nameless objects are protected from eavesdroppers if their parent packet plaintext hash digests are unpredictable. In section 5.5.3, we assumed that manifests carrying these hash digests were themselves encrypted using TRAPS. Therefore, if manifest names are predictable, then hashes can be discovered by anyone. To improve this, manifests could be protected in a number of different ways. First, a secure session established via CCNxKE could be used to transfer manifests. This would prevent passive eavesdroppers between consumers and producers from learning manifest contents. Alternatively, if access control schemes such as IBAC are used, wherein consumers and producers share keying material that can be used to encrypt names, then manifests could be fetched with encrypted names. While using a session or IBAC to fetch manifests would work, both introduce extra complexity into applications that is otherwise not needed by TRAPS.

Dictionary Attacks

When $s_G = \perp$, TRAPS is susceptible to dictionary attacks [254] since the only secret information in the protocol is N, the application name, which is known by all consumers who request content.[23] A dictionary attack is where an adversary precomputes hash digests from a dictionary or list of popular inputs (names) so as to easily reverse these values. The

[23]Using deterministic public-key encryption for name obfuscation would serve no better than the hash function. Since adversaries would also have access to the producer's public key, and could therefore compute the obfuscated names as easily as regular consumers.

first dictionary attack deterrence built into TRAPS is the use of salts, generated via s_G, to obfuscate names.

There are several possible options for s_G. The first is for s_G to return $s \leftarrow_{\$} \{0,1\}^{\lambda}$, and to rotate (update) this value every $s_R()$ seconds. One problem with salts is consumer synchronization: how does a consumer get a salt before fetching $I(N)$? If salts are *explicit*, then producers must generate and publish them for consumers to obtain. Specifically, let N_s be a salt name published with obfuscated name $\mathsf{Obfuscate}(\mathsf{F}, s_G, N_s)$. Cr could issue an interest for N_s to obtain the salt s, and then use s to subsequently derive names of desired content.

We recommend a simpler input for the salt: time. Here, s_G is equal to the current time epoch divided by s_R. For example, if the current time epoch (UNIX time) is 1483921358 (01/09/2017 @ 12:22am UTC), and s_R is 10000, then $s = 1483921$. Consumers and producers share knowledge of time and can be assumed to be in sync within some loose margin of error. As the granularity of time (s_R) decreases, the probability that it is shared between consumers and producers increases. Of course, since time is predictable, it is possible for a powerful attacker to pre-compute obfuscated names with future versions of time. This could be mitigated by mixing both time and a producer-provided salt on a regular frequency. For example, producers could publish a new salt every day or week, which would then be required when computing F.

Another deterrence built into TRAPS is to use MHFs as F. These dampen the efficacy of offline dictionary attacks while adding more online computational and memory overhead to consumers. This performance tradeoff is assessed in Section 5.5.5.

5.5.4 Security Analysis

Recall that one TRAPS goal is to allow anyone with knowledge of or access to N to request and decrypt it accordingly. Conversely, it should be hard for anyone without N to request and decrypt content. In our threat model, this is captured by an eavesdropping $\mathcal{A}$ whose goal is to decrypt a packet it observes in transit. There are two types of packets considered in this model: those with and without a content identifier (or named and nameless content objects, respectively). By design, security of each is based on unpredictability of either a name or plaintext content, as we describe below.

Following Bellare et al. [52], we define an MLE encryption scheme to be a tuple of algorithms $\mathsf{MLE} = (P, K, E, D, T)$, responsible for parameter initialization, key generation, encryption, decryption, and tag generation. In [52], MLE schemes were proven secure in a so-called chosen distribution attack (CNDA) scenario. The main idea is that if each message that can be drawn from the *source* of messages is unpredictable, then the scheme is secure with overwhelming probability. Formally, a source is a polynomial-time algorithm $\mathsf{NG}_{D(\mathcal{N})}$ that, on input 1^λ, returns a list of names $N_0, \ldots, N_{\gamma-1}$, where $N_i \in \mathcal{N}$, and some auxiliary information Z about those names. Each name is sampled from the distribution $\mathcal{D}(\mathcal{N})$ and has the same length $m(\lambda)$. (In our scenario, we let $\gamma = 2$ without loss of generality.) Using this information, we now define the main privacy game as follows. Let $\mathcal{A}_{t,s}$ be $\mathcal{A}$ parameterized by computation time t and space parameters s. When not needed, we omit these parameters.

Game $\mathsf{PRV\text{-}CNDA}^{\mathcal{A}}_{MLE,\mathsf{NG}_{\mathsf{D}(\mathcal{N})}}$

```
$P \leftarrow_\$ \mathcal{P}(1^\lambda)$
$b \leftarrow_\$ \{0,1\}$
$(N_0, N_1, Z) \leftarrow_\$ \mathsf{NG}_{D(\mathcal{N})}(1^\lambda)$
$C_b \leftarrow_\$ \mathcal{E}_P(\mathcal{K}_P(D(N_b)))$
$b' \leftarrow \mathcal{A}(P, C_b, Z)$
return $b = b'$
```

For this game, $\mathcal{A}$'s advantage is defined as

$$\mathcal{A}^{\mathsf{prv-cda}}_{MLE,\mathsf{NG}_{D(\mathcal{N})},\mathcal{A}} = 2 \cdot \Pr[\mathsf{PRV\text{-}CNDA}^{\mathcal{A}}_{MLE,\mathsf{NG}_{D(\mathcal{N})}}] - 1.$$

TRAPS is $\mathsf{PRV\text{-}CNDA}$ secure if $\mathcal{A}^{\mathsf{prv-cda}}_{MLE,\mathsf{NG}_{D(\mathcal{N})},\mathcal{A}}$ is a negligible function in λ. To be precise, we require that the symmetric-key encryption scheme has both key recovery (KR) security and one-time real-or-random (ROR) security [52].[24] With this, we are now prepared to define the security of TRAPS.

Theorem 5.3. *Let* KDF *be modeled as a random oracle and let* $\mathsf{SE} = (\mathsf{SK}, \mathsf{SE}, \mathsf{SD})$ *be a one-time symmetric encryption scheme with key length* $k(\cdot)$ *is KR- and ROR-secure. Let* $\mathsf{MLE}(\mathsf{SE})$ *be a MLE-scheme instantiated using* SE*. Then, all* non-exposed data *protected by* TRAPS *are PRV-CNDA-secure.*

Proof. First, observe that $\mathcal{A}$ can effortlessly see N for any data generated or requested by an exposed producer or consumer, respectively. Moreover, any exposed data item $D(N)$ is inherently insecure since $\mathcal{A}$ knows N and can therefore decrypt the packet. Now, by the name-to-data binding property in CCN, there is a one-to-one correspondence between names and data. Therefore, it follows that the distribution of names is equivalent to the distribution of data. Thus, by the proof given in [52], it follows that non-exposed named packets protected by TRAPS are PRV-CNDA-secure. Since TRAPS is used to protect named content objects, which in turn protects nameless content objects, PRV-CNDA-security follows for them as well. $\square$

If manifest(s) used to convey data digests is (are) protected using a CCA-secure encryption scheme [180], then security can be improved to that of PRV-CDA [52], which is similar to PRV-CNDA security except that it depends on content object unpredictability rather than name unpredictability. (Typically, data have more entropy than names.)

[24]The reader is referred to [52] for formal definitions of these schemes.

Theorem 5.4. *Let* KDF *be modeled as a a random oracle and let* $\mathsf{SE} = (\mathsf{SK}, \mathsf{SE}, \mathsf{SD})$ *be a one-time symmetric encryption scheme with key length* $k(\cdot)$ *is KR- and ROR-secure. If the Manifest(s) carrying data digests are protected with CCA-secure encryption, then the corresponding nameless packets which are protected by* TRAPS *and not exposed are PRV-CDA-secure.*

Proof. CCA-security implies semantic security, and therefore no information is leaked from an encrypted Manifest. Moreover, since data items $D_i(N)$ are not exposed, $\mathcal{A}$ has no information about their contents. Thus, PRV-CDA security follows immediately by the proof given in [52]. □

The Long Tail

TRAPS security vanishes as information (names and data) become predictable. This is because trial decryption driven by dictionary attacks are possible. However, there is one crucial element of TRAPS that limits trial decryption attacks: Since each content key is derived from a fresh and random nonce, $\mathcal{A}$ cannot attempt trial decryption until *after* it observes an encrypted content object. This means dictionary attacks targeting the content must be *online*. Given realistic traffic volumes on modern networks, this means $\mathcal{A}$ has to either have a tremendous amount of computational resources available or must be selective in which content it attempts to decrypt. Both conditions make less popular or predictable content, i.e., content in the "long tail" of the popularity distribution, intuitively safer.

Another avenue for attack is through the obfuscated name via an offline dictionary attack. The cost of the dictionary attack is controlled by F. To illustrate this effect, let the security of F be defined by space and time parameters s and t. Let $\mathbb{N}$ be a set of names from which the attacker will sample. Moreover, let $\mathcal{D}(\mathbb{N})$ be the distribution of these names such that $\mathcal{D}(N)$ for $N \in \mathbb{N}$ is the probability that N is selected when sampled from $\mathbb{N}$. An optimal dictionary attack is one where $\mathcal{A}$ proceeds as follows. $\mathcal{A}$ traverses $\mathcal{D}(\mathbb{N})$ in descending order by probability and computes $\mathsf{Obfuscate}(\mathsf{F}, s_G, I(N))$ for each N until the target value is found. We may estimate this attack complexity as follows. Let N^* be the actual name represented by a packet with the obfuscated name $\bar{N}$. Let $C(s,t)$ be the cost of computing F given parameters s and t. Assuming N^* is the k-th most popular name in $\mathcal{D}(N)$, this attack will require (at most) k computations of $\mathsf{Obfuscate}$ and therefore cost $k \times C(s,t)$. This leads to the following minimum work bound for these offline dictionary attacks.

Definition 5.3. *Given a space and time parameters s and t, as well as a set of names $\mathbb{N}$ with distribution $\mathcal{D}(\mathbb{N})$ whose expectation is $E(\mathbb{N})$ and PMF is $f(\cdot)$, the average amount of work required to conduct a dictionary attack on one name $N \leftarrow \mathbb{N}$ is $(1 - f(E(\mathbb{N}))) \times C(s,t)$.*

If $\mathcal{D}(\mathbb{N})$ is the uniform distribution with 10 elements, then the expected cost is $5 \times C(s,t)$. Similarly, if $\mathcal{D}(\mathbb{N})$ is the Zipf distribution with 10 elements, whose expected value is $\frac{H_{10,\alpha-1}}{H_{N,\alpha}}$, then the expected cost is $(1 - \frac{H_{10,\alpha-1}}{H_{N,\alpha}}) \times C(s,t)$. The cost scales linearly as the set of names increases.

5.5.5 Performance Assessment

In this section we assess the performance of TRAPS. All experiments were performed on a workstation with a 2.8 GHz Intel Core i7 CPU and 16GB of 1600 MHz DDR3 RAM running Ubuntu 14.04.

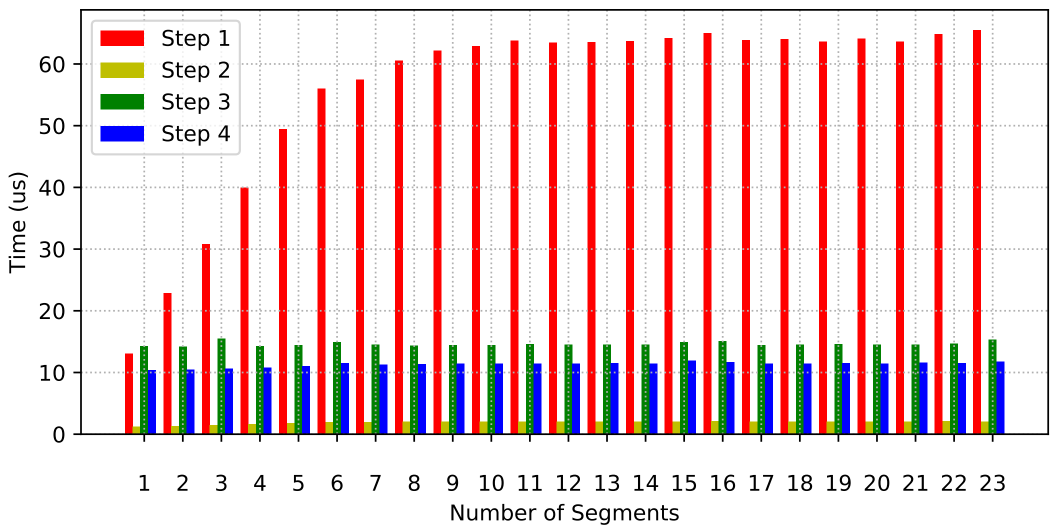
End Host Overhead

Computationally, TRAPS only introduces overhead at consumers and producers. Routers and other network entities are unaffected by interests and content object messages using TRAPS name obfuscation and content encryption. Therefore, to assess TRAPS performance, we quantify overhead incurred by consumers and producers. This overhead is divided into four parts (steps): (1) interest obfuscation, (2) interest de-obfuscation, (3) content encryption, and (4) content decryption. Each of these procedures adds processing overhead to every interest and content object pair.

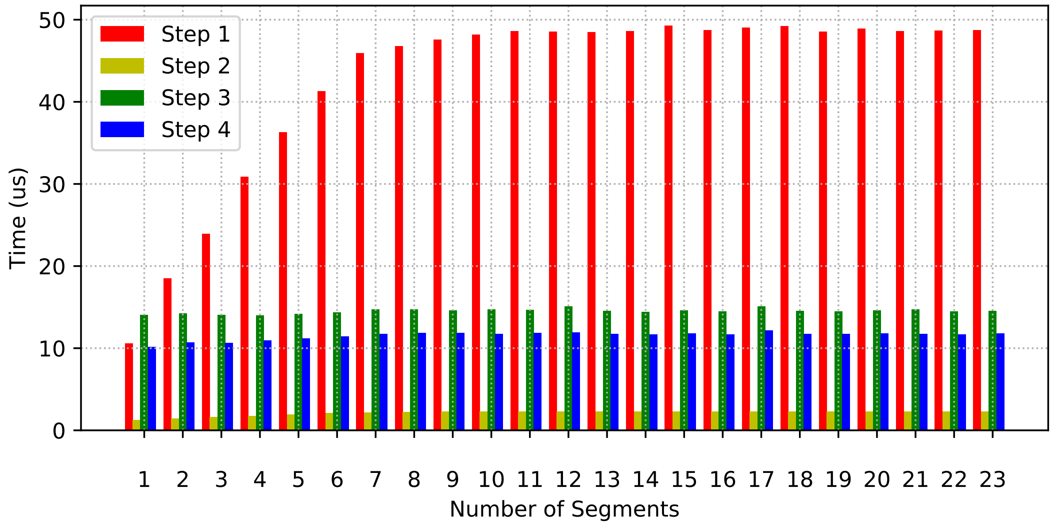
To quantify this processing overhead, we implemented each of operation in C on top of the Libccnx and Libparc [1] libraries. For input, we used the Unibas dataset from the The Content Name Collection [11], which contains *unique* URLs submitted by users to URL shortener websites. We converted these URLs into a CCN-compatible name format. For example, the URL `http://www.domain.com/file.html` was converted into the CCN name `/com/domain/file.html`.

This dataset does not provide any information about the corresponding content object sizes, so we randomly generate the sizes between the range of 1.5KB and 9KB. For efficiency, we use ChaCha20+Poly1305 [244] content object encryption. Randomness for nonce generation is drawn from `/dev/urandom` since it does not block when insufficient entropy is available. For comparison we use both SHA-256, which is *not* a MHF, and Argon2 [59], which is a recent MHF, as the name obfuscation function F. Our experimental software is available online at [14].

Figure 5.20 shows results from this experiment. The left plot uses Argon2d with parameters $t = 6$ and $m = 12$ (2^{12} KiB) and the right plot uses SHA-256. The name de-obfuscation procedure (Step 2), which is simply a hash table lookup, is negligible compared to other steps. Request obfuscation is most expensive since it requires computation that is linear in the length of each input name. Content encryption and decryption perform moderately better; encryption involves more work since it must first sample randomness necessary to generate each random nonce. Overall, the worst-case time for a single step in TRAPS on our machine is approximately 60μs, which is well below average network I/O bottlenecks and therefore feasible.



(a) F =Argon2 (default parameters)



(b) F =SHA-256

Figure 5.20: TRAPS performance overhead

Consumer Throughput

Name obfuscation step is the most expensive step in TRAPS and thus a throughput bottleneck. Using MHFs instead of traditional hash functions lowers the throughput ceiling even further. Thus, applications must take care to not use Fs that disrupt normal QoS. This has two implications on F: (1) computation overhead must never exceed network overhead and (2) the number of *possible* hash functions per second should always exceed the number of packets *sent* per second. This suggests two strategies in selecting F and its parameters. Let R be the maximum number of bytes per second sent by a client Cr. Let L be the minimum link MTU from Cr to each producer $P_1, \ldots, P_n$. Let T_{min} be the minimum RTT between Cr and any P_i. Finally, let T_{obf} be the (worst-case) time required to obfuscate a name in a single packet.

Based on the previous two conditions for throughput, it must be the case that

$$\frac{PT_{obf}}{L} < 1 \tag{5.1}$$

and that $T_{obf} \leq T_{min}$. Equation 5.1 states that the total time consumed by hashing all required packets in an epoch does not exceed one second. Otherwise, the system would be unstable since the client could not keep up with the desired packet transmission rate.

Equation 5.1 also places an upper bound on the number of packets that can be sent every second. Given T_{obf} and assuming an approximate value of $L = 1500$B, we can compute an upper bound on P to satisfy this inequality. That is, we can find the largest P such that $P < \frac{L}{T_{obf}}$. To estimate this capacity, we profiled SHA-256 and Argon2 to measure the expected throughput under a variety of configurations. Figure 5.21 plots the results using Argon2 with $t = 4$ and $m \in \{21, 25\}$ and SHA-256 as F. As expected, we can achieve packet throughput rates on the order of 10^8 interests/second with SHA-256, but with a $m = 25$ (2^{25} KiB) memory cost for Argon2 this drops down to 10^5 interests/second.

We may also write Equation 5.1 as

$$T_{obf} < \frac{L}{P} \tag{5.2}$$

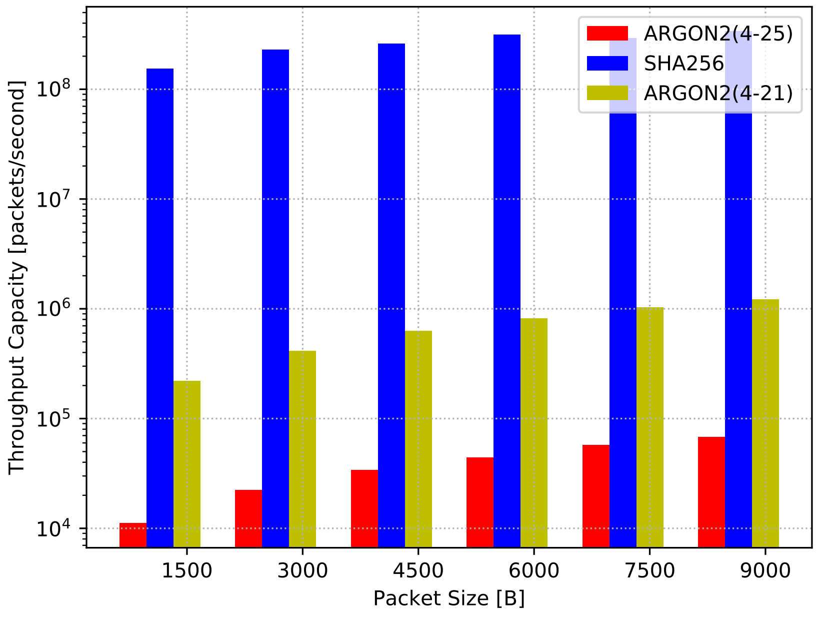


Figure 5.21: TRAPS throughput capacity

and, for reasonable values of P, find F parameters that bring it close to this upper bound. For example, assume $P = 100$Mbps and $L = 1500$B. Then, $T_{obf} < 15\mu$s. This is well within the bounds when using SHA-256 as F. However, this is not the case for Argon2. For P values of 2Mbps and 128Mbps, Argon2 parameters $t = 8$ and $M = 26$ can be used to meet throughput criteria. Larger values for m indicate that more memory is used for the function, which is the ultimate limiting factor in its performance as well as the primary factor in its security (see [59] for more details). For comparison, Netflix Ultra-HD quality streaming requires a throughput of 25 Mbps [241]. Based on this assessment, TRAPS can certainly meet this requirement.

Salt Rotation Overhead

TRAPS security depends in large part on (a) name unpredictability and (b) name salt rotation frequency. Let $P_{\mathbb{N}}$ be the set of names owned or otherwise published by producer P. Then, for each name $N \in P_{\mathbb{N}}$, P must compute $F(N)$. If T is the average cost of computing $F(N)$ for any name $N \in P_{\mathbb{N}}$, the fastest rate at which P could update is once every $T \cdot |\mathbb{N}|$. Assuming P wanted to rotate its salt every day and that it was using an obfuscation function with $T \approx 0.5$s, the maximum size of $\mathbb{N}$ is roughly $172,000$. (This assumes P spends the entire day pre-computing the next day's names with on a freshly chosen salt, which is clearly undesirable.)

Chapter 6

Availability

Denial of Service (DoS) attacks are a major threat to any network architecture. They range in severity from slowing down a single node in the network to taking an entire subnet offline. DoS attacks on today's IP-based networks include bandwidth depletion via floods [18], DNS cache poisoning [292], black-holing and prefix hijacking [208], and reflection attacks [299], among others. As [136] suggests, CCN features curtail many of these attacks by design:

- Flow symmetry prevents reflection attacks [136].
- Interest aggregation and content caching mitigate bandwidth depletion attacks [318].
- Verified content authenticity *mitigates* prefix hijacking attacks [150].

This resilience comes at significant cost, as described at length by Aamir et al. [21]. In standard CCN, flow symmetry, specifically, requires per-packet state stored at routers. Consequently, this state can be easily abused via so-called Interest Flooding (IF) attacks [24, 136, 85, 30]. In one IF attack, a malicious consumer (or a distributed botnet) issues nonsensical interests[1] so as to overwhelm targeted routers and saturate their PITs. Since

[1]For example, an interest with a name reflecting a valid producer's prefix, with a random number as its last segment.

such interests will not be satisfied and yield a NACK, PIT resources remain occupied longer than those for legitimate traffic.

According to Carofiglio et al. [75], PIT entry counts can range from fewer than 100 for edge routers with a small number of per-namespace flows, to well over 10^6 in network cores as upstream paths become congested.[2] Guimarães et al. [159] suggested that even small ratios of malicious interests may result in a non-negligible number of legitimate interests being dropped. PIT timeout values play a clear role in this packet loss. Moreover, malicious consumers and producers can cooperate to target specific routers and exploit timeout values.

Numerous attempts to address IF attacks are proposed in the literature. Gasti et al. [136] surveyed a number of possible approaches, including local router statistic collection to drive pushback and per-interface rate limiting. Compagno et al. [85] extended these ideas with Poseidon, a router-local mechanism to implement rate limiting based on per-face fairness metrics. The approach targets interests for existing and non-existing content, the latter of which being particularly challenging to suppress. Similar techniques were explored by Wang et al. [322]. Rather than use fairness metrics, they use fuzzy logic filters based on PIT occupancy and entry expiration rates to trigger downstream pushback mechanisms. Routers are assumed to cooperate. Another attempt to classify interest flooding beyond fairness metrics was studied by Nguyen et al. [242]. They developed a flooding detector that exploits the divergence of upstream interest and downstream content flows when under attack. The threshold for their detector factors in a desired false probability value.

Kondo et al. [191] scanned URLs in use on today's Internet and use the results to develop a set of heuristics and statistics to detect anomalous names that do not resemble legitimate URLs. Firewalls used these rules to filter malicious or fake interests. This technique assumes names will have human-readable segments and patterns, which is not true of encrypted or otherwise obfuscated names that are not necessarily malicious. Salah et al. [275, 274] propose

[2]A similar model was studied by Wang et al. [319].

an IF countermeasure that relies on router cooperation. Special routers, called *monitors*, signal to a central domain (or AS) controller the state of their PIT entries. Collected state is used to determine if any router is under attack. This design depends on monitor distribution and router-to-controller signaling rates.

Dai et al. [93] proposed an interest traceback mechanism to link floods to their sources. Equipped with this knowledge, routers can take preventative steps to sever links to the source. Li et al. [202] proposed using puzzles to throttle consumer interests. In their design, routers may force downstream hops to solve puzzles (with proofs-of-work) as a form of payment before forwarding packets.

Alternative PIT implementations were studied by Virgilio et al. [314]. Experiments suggest that PITs which store full names, hashes of names, BF-based sets of names are all vulnerable to DoS attacks and perform equally well under normal operating conditions. (One additional problem is that BF-based PITs necessarily introduce false positives.) Almirshari et al. [31] proposed a technique to "piggyback" interest and content objects to enable high throughput bidirectional communication in NDN and remove unneeded PIT state for half of the traffic. However, as it does not remove PITs entirely, it is still susceptible to IF attacks. Dai et al. [92] studied PIT extensions to support modern applications such as streaming services and online gaming. They proposed creating long-lived PIT entries for bidirectional communication between clients and servers. This long-lived state makes IF attacks easier to conduct.

Techniques to outright replace the PIT have also been proposed. Tsilopoulos et al. [308] devised a "semi-stateful" solution wherein packets are marked (with BFs) with forwarding information. This approach shifts PIT state to packets themselves and creates additional network communication and control overhead. Wang et al. [321] proposed related architecture changes wherein routers selectively move per-router state to per-packet state for suspicious interests. They introduced a secondary data structure that stores malicious prefixes under attack. If an interest arrives with a matching malicious prefix, reverse-path forwarding state is stored in the packet, not the PIT, before forwarding. Upstream routers must honor this state and use it to forward responses downstream. Mirzazad-Barijough et al. [132] proposed a MPLS-like stateless variant for CCN. Their approach addresses IF though still enforces path symmetry and does not generally aid applications that require bidirectional communication. Garcia et al. [131, 132] introduced CCN-GRAM (Gathering of Routes for Anonymous Messenger), which is a type of semi-stateless CCN architecture that uses "anonymous datagrams" in lieu of normal CCN packets. CCN-GRAM replaces per-interest state with source and destination "addresses," where a source address is an anonymous identifier with local meaning and a destination address is a content name as per usual. CCN-GRAM achieves similar functionality as stateful CCN without router per-packet state. This means it inherits problems of forced path symmetry, which can be problematic for mobility (a growing use case). In Section 6.1, we describe a similar stateless (or PIT-less) architecture in which PIT state is replaced by *backwards-routable names*, or RBNs. These RBNs permit routers to forward the resulting content based on FIB entries, rather than PIT entries. The technique melds with standard CCN at network edges.

PIT are not the only vulnerable router data structures. Yuan et al. [340] studied forwarding engine performance to determine pain points for forwarding based on CCN names at line rate. They concluded that the primary problems were (1) exact string matching with fast updates, (2) longest prefix matching for variable-length and unbounded names, and (3) large-scale flow maintenance. Thus, adversaries can exploit variable length names to attack FIBs and caches.

Design and implementation of high-speed CCN forwarders are well studied in the literature [336, 289, 290, 260, 129]. For example, Quan et al. [260] proposed an efficient name lookup engine for CCN forwarders that exploits name variability. Specifically, Tree-Bitmaps [109] and BFs are used together, in parallel, to look up a name using LPM. The proposed parallel lookup algorithm computes BF hashes on one half of a name and performs exact match in a Tree-Bitmap on the other half. So et al. [289**?**] also proposed an efficient forwarding engine that uses a hash-based data structure for name LPM. The design uses SipHash for its low cycles/byte performance and collision properties.

Perino et al. [250] proposed an alternative LPM implementation that does not use successive lookups. Their approach decomposes the LPM problem into two subproblems: (1) finding the longest length of a matching prefix using a BF and (2) a hash table lookup. High-quality pre-computed hashes improve the efficacy (2) with negligible effect on (1). (Other BF-based LPM solutions include [324] and [104, 293] for IP networks.) Fukushima et al. [129] built on [250] and used BFs composed with a hash table lookup to reduce search space and hash table size.

Song et al. [295] proposed a novel FIB design that uses name compression and Patricia tries [301] to reduce FIBs footprints. Wang et al. [326] proposed a LPM implementation that relies on a specific sorted order of names in FIBs. It builds on work of Wang et al. [323], which proposes a special name segment encoding scheme.

All aforementioned designs focus on router implementation assuming a fixed packet format that carries application names. Ali et al. [151] examined requirements for application naming schemes in CCN without discussion of how they can be efficiently conveyed in the network layer. In Section 6.2, we propose the concept of *network names* that change PIT and FIB inputs, rather than their implementation, to improve performance and harden against dataplane attacks.

IF and more general DoS attacks are not unique to CCN. Alzahrani et al. [32, 34] showed that DoS attacks are easy to implement in PURSUIT by adding noise to content zFilters. (Recall that zFilters determine to which link(s) published content is sent, as determined by Topology Managers.) Alzahrani et al. [32, 34] built on Rothenberg et al. [271] and proposed to mitigate these problems by binding zFilters to ephemeral identifiers derived from link interfaces, content itself, and key shared with topology managers. Each hop re-computes expected zFilter values using content and a current shared key before performing per-interface matches and making forwarding decisions. Alzahrani et al. [33] later specified a key management scheme that allows each forwarder to verify zFilter authenticity. However, per-hop verification can become another form of DoS.

IF and other attacks that focus on dataplane state exhaustion are only one of many network-layer attacks. Wählisch et al. [315] surveyed a variety of attacks that exist on general ICN architectures, including router-targeted computational and state exhaustion, targeted depletion of shared bandwidth via interest exhaustion, as well as state decorrelation attacks, e.g., via well-timed cache manipulation. (Technically, state decorrelation is more of a concern to publish-subscribe networks, rather than request-based architectures such as CCN.) One attack not considered centered on prefix hijacking. This attack injects fake content to limit availability and induce DoS through expensive content verification.

Ghali et al. [142] proposed preliminary attempts to isolate and identity fake content in CCN using popularity metrics. However, their technique is imperfect since it relies on honest consumer feedback. Later, Ghali et al. [150] higlighted that content poisoning is reducible to network-layer trust and, consequently, laid out mechanics that soon became KeyId and ContentId in CCN. Providing one or both of these restrictions is necessary, though not always sufficient, for preventing content poisoning attacks.[3] Wong et al. [328] proposed a different variant for publish-subscribe networks such as PURSUIT by publishing verification data, such as certificates and public keys, separately from content in the data plane. Subscribers can securely fetch and use verification data separately from actual content.

[3]Cache pollution [87, 176, 216, 248, 335, 177] is a related attack on availability wherein cooperating consumers and producers populate caches with valid but useless (unwanted) content. The goal is to invalidate the utility of network caches. However, as this fake content will never be requested by legitimate consumers, it is only an attack on content availability and network performance. Mobility-First and XIA are also susceptible to this attack.

In CCN and related architectures, content verification *is not* a mandatory precondition for content forwarding. A router must only verify content before serving it from a cache, as described in Section 3.5.1. A router which does not cache content therefore has no incentive to verify content signatures. Moreover, a caching router may not verify and cache all content if the content arrival rate is too high, even if in some cases it is feasible [58]. Even with optimizations such as segmented caches and verification-on-cache-hit, as described by Kim et al. [188], per-packet public key cryptography can be prohibitively expensive at high line rates.

As an added complication, packet verification requires each byte to be processed by a router. If a content object (or interest) is fragmented, routers may have to allocate a non-negligible amount of memory to store and reassemble constituent fragments. NDN uses this hop-by-hop fragmentation scheme by default [25]. Routers reassemble and verify content objects before forwarding any individual fragments. Fake content can easily exploit this behavior as an attack on routers. Ghali et al. [139, 140] proposed a secure cut-through fragmentation scheme for CCN called FIGOA that allows routers to quickly forward individual fragments while incrementally computing the parent packet's cryptographic hash digest. Verification can then be quickly checked once the final fragment is received, without reassembly. Mosko et al. [230] slightly modified FIGOA to move content signatures and hash digests to leading packet fragments. This allows routers to verify signatures immediately and continue forwarding fragments, thereby amortizing the verification cost with fragment forwarding overhead.

Optional and inconsistent content verification permits on-path attackers to silently hijack name prefixes in CCN. One way to avoid this is to proactively try and avoid malicious routers during normal operation. Wu et al. [334] proposed using per-router reputations to rank forwarding interfaces. Router reputation decreases exponentially as poisoned to non-poisoned content ratios increase. Routers are punished, i.e., not forwarded interests, as they forward more poisoned content. Determining upstream router reputation requires mandatory content verification, which may be infeasible. DeBenedetto et al. [106] proposed an alternative mechanism wherein consumers alert upstream routers to fake content, allowing them to adjust their forwarding tables if necessary. However, as we discuss in Section 6.3, this reactive approach is cumbersome and costly. In Section 6.3, we propose a lightweight network-layer integrity checking mechanism that helps routers verify the integrity of a packet as it flows through the network.

6.1 A Stateless Data Plane

In this section, we comprehensively assess the stateful forwarding plane of CCN with respect to many of its claimed benefits, including support for reverse-path routing, infrastructure security, flow and congestion control, and interest aggregation. We show that many benefits are: (1) either unrealistic or infeasible in practice, (2) can be achieved by means other than stateful forwarding, or (3) so marginal that their value simply does not justify their overhead. We then present, in Sections 6.1.2 and 6.1.3, a new stateless architecture for CCN based on Routable Backward Names (RBNs). This design can co-exist with the current CCN architecture (with PITs) or replace it entirely. Our proposed stateless architecture is different from that of Mirzazad-Barijough, et al. [132], where content is forwarded using MPLS-like labels and not per-packet state. As we show in Section 6.1.1, [132] still assumes pull-based communication as the preferred mechanism for all applications and enforces path symmetry for interests and content objects. After discussing the design, Section 6.1.4 concludes with experimental results which indicate that stateless CCN retains the essence and performance characteristics of standard CCN while successfully avoiding pitfalls of per-interest packet state.

6.1.1 Assessing the PIT

Currently, the PIT is a fundamental and mandatory component of the CCN forwarding plane. As shown by Carofiglio et al. [75] and others, PIT load is dependent on router location, namespace properties, and adversary displacement in the network. A PIT implementation can be optimized, yet it will always be subject to unpredictable and untrustworthy network conditions. Thus, aside from being unnecessary to support CCN-like communication, the PIT raises more (serious) problems than it solves. We support this claim by systematically analyzing the following alleged PIT benefits:

1. Reverse-path forwarding
2. Infrastructure security
3. Flow and congestion control
4. Interest collapsing

We then show that all these benefits are either false, unnecessary, or very meager at best.

Reverse-Path Forwarding

A key tenet of CCN is that content is never sent to a consumer who previously did not issue an interest, i.e., does not have a pending interest, for this content. According to [172], since interests contain no source addresses, PITs are needed:

> "...to forward Content Objects from producers to consumers along the interest reverse path by leaving per-hop state in each router..."

We disagree with this statement for two reasons. First, network path symmetry is not guaranteed and should not be assumed. [174] demonstrated that route symmetry between the same flow on the Internet is lower in the core than at the edges. Several tier-1 and tier-2 networks were studied and it was shown that, due to "hot-potato-routing," flow asymmetry exceeds 90% in the core. Thus, symmetric path routing in the core appears to directly contradict today's practices that promote and exploit path asymmetry for better traffic distribution. Attempting to enforce symmetric data traversal appears to be a challenge from an economic perspective.

Second, pull-based communication with symmetric paths is not well-suited for *all* applications. While appropriate for scalable content distribution applications[4], it is substantially different from modern TCP/IP applications and protocols which rely on interactive sessions and bidirectional streams between endpoints. For instance, the WebSocket [118] protocol uses full-duplex TCP streams for clients and servers that engage in real-time, bidirectional communication. It is used by many popular interactive applications, such as multimedia chat and multiplayer video games. Two-way communication is not limited to Web protocols. Voice applications such as Skype [286] and peer-to-peer systems such as BitTorrent [83] rely on two endpoints which both produce and consume data as part of the application.

Given the relative infancy of CCN and abundance of real-world applications that currently do not fit CCN's mold, it is difficult to argue that pull-based communication can satisfy all application needs. Moreover, even today, some existing CCN applications abuse interest messages to carry information from consumers to producers [71]. Other applications rely on consumers and producers to send interests to each other as data mules. NDN-RTC, a recently developed NDN video teleconference application, is one such example that supports such bidirectional communication between peers [162]. (We use NDN and CCN interchangeably here since both are equivalent in this context.)

Another emerging application design pattern is data transport via set synchronization. The NDN ChronoSync protocol is a prime example of this pattern [348]. Each ChronoSync user acts as *both* a producer and consumer. Consumers (members) issue *long-standing* interests

[4]Which some believe to be already well-served by today's CDNs.

to a group (common namespace) about specific data to be synchronized; These interests are routed to all members. When target data is changed by someone, this member satisfies previous interest(s) with a fingerprint of the data in a content object. Each member is then responsible for requesting updated content to synchronize with the others. This protocol is built on the fundamental assumption that pull-based data transmission is the only communication pattern.

Based on the trends of current TCP/IP applications and proposed design strategies for CCN-based protocols and applications, it seems clear that bidirectional communication is here to stay. For it to work, router FIBs need to contain prefixes for all end-points – not just producers. Therefore, all communicating parties need to obtain and use a routable prefix, which effectively serves as an address. As a consequence, *forwarding information stored in a PIT becomes redundant and unnecessary.*

Infrastructure Security

Gasti et al. [136] showed how CCN (in the context of NDN) prevents many modern DoS attacks, including: bandwidth depletion, DNS cache poisoning, black-holing and prefix hijacking, as well as reflection attacks. Out of all attacks considered, the PIT is needed only to prevent reflection attacks [299]. Since content is forwarded based on PIT entries, such attacks are impossible in CCN. However, forwarding content via the PIT is not the only way to prevent reflection attacks. If packets have a source address, the ingress filtering technique in [117], whereby ISPs filter packets based on source addresses, would work equally well. Moreover, despite its resilience to reflection attacks, CCN is susceptible to IF attacks [136]. Although several attempts to detect, mitigate, and prevent these attacks have been made [314, 85, 320, 24, 243, 242, 302], each of them is effective against only a very naïve or weak attacker. Thus, IF attacks remain a daunting open problem with no solution in sight barring network architecture changes.

Flow and Congestion Control

[336] presented the first thorough argument in support of a stateful forwarding plane in the context of NDN. (Due to their near-identical features, the same applies to CCN.) The PIT can be used to record RTTs for interest and content exchanges, which, in turn, is useful for making dynamic forwarding decisions. For instance, if the RTT for content in a given namespace on a particular link becomes too high, that link might be congested and alternatives should be explored. This type of in-network congestion and flow control has been studied further in [210, 337, 272, 273, 73]. For instance, [73] propose a joint hop-by-hop (i.e., in-network) and receiver-based control protocol that relies on PIT-based RTT measurements for flows. In-network flow control allows routers to control flow closer to congested links, as opposed to performing the same by receivers.

However, according to [325], flow differentiation is a difficult challenge. One approach to "interest shaping" is by controlling the flow of data on upstream and downstream links independently of flows. This does not require any information from the PIT. Instead, it relies on knowledge of average interest and content size, link bandwidth, and interest arrival rates (or demand). Similar to [273], it also relies on receiver-driven flow control via an Additive-Increase-Multiplicative-Decrease window. Ren et al. [264] proposed another example of a receiver-driven flow control protocol for CCN. In contrast, Mahdian et al. [210] proposed a rate-based congestion control protocol that exploits the multi-path and stateful nature of CCN. Given these results, it is not clear where congestion control logic is most useful. Nevertheless, recent trends in the CCN research community show that pushing stateful control protocols towards receivers, rather than to network nodes, is a viable and attractive approach.

Utility of Interest Collapsing

Dabirmoghaddam et al. [91] were the first to accurately model interest collapsing in CCN and NDN. The results indicate that collapsing occurs very little, i.e., with probability rarely exceeding 0.15, at the edge of the network (where content will be cached) for popular content classes. In this section, we perform our own independent analysis and confirm these results.

Interest collapsing is only performed on interests arriving at routers during a small window of time Δ. Due to the increase of network data rates and the decrease of end-to-end delays provided by in-network caching, the value of Δ is small, e.g., on order of tens of milliseconds. Therefore, we believe that the effect of interest collapsing does not play a crucial role in the performance of CCN.

To support this claim, we model the probability of interest collapsing occurring in the first hop router R. The reason for choosing the first router is that the benefits of interest collapsing are closest to the consumer(s). This is because collapsing two similar interests at the consumer-facing router reduces bandwidth usage more than if the collapsing happened closer to the producer. We assume that content popularity follows a Zipf distribution with classes $k = 1, \ldots, K$ and average number of segments σ_k.[5] Let each class arrival rate λ_k at R be modeled as a Poisson process. The event of interest collapsing at R for content class k is denoted as $\mathsf{Coll}^{R}_{int}(k)$. The probability of this event occurring is given as [74]:

$$\Pr\left[\mathsf{Coll}^{R}_{int}(k)\right] = \frac{1 - e^{-\Delta\lambda_k}}{1 - (1 - 1/\sigma_k)e^{-\Delta\lambda_k}} \tag{6.1}$$

[5]Large content is typically split into smaller segments.

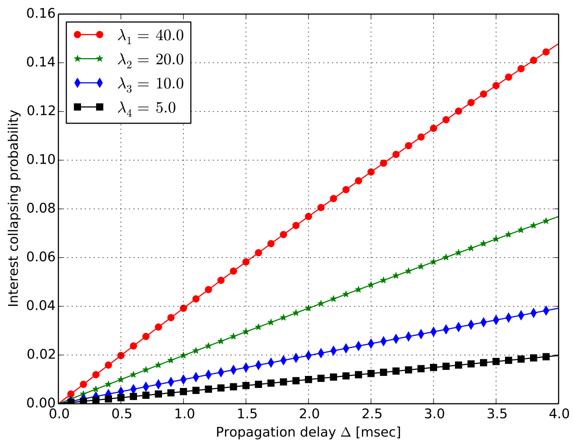
Theorem 6.1. *Assuming in-network routing is only enabled at edge routers [133], the interest collapsing probability at consumer-facing router R is*

$$\Pr\left[\mathit{Coll}^R_{int}(k)\right] = \left(1 - p^R_k\right)\left(1 - \left(\prod_{i=1}^{L} e^{-\frac{l_i}{\alpha_i}}\right)^{\frac{2\lambda_k}{c}}\right) \tag{6.2}$$

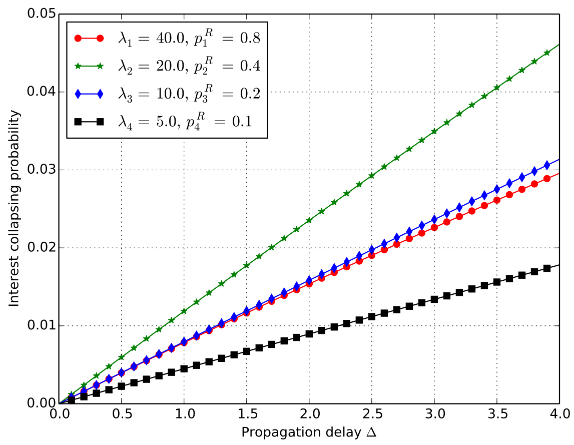
for L links between R and producer P, $c = 3\times10^8 m/s$ the speed of light, l_i the length of link i, constant α_i that depends on the characteristic of the material of which link i is manufactured, and p^R_k the cache hit probability of a class k content at R.

Proof. We are only interested in modeling interest collapsing for individual content objects, thus we set content size $\sigma_k = 1$ segment. Therefore, Equation 6.1 can be written as follows.

$$\Pr\left[\mathsf{Coll}^R_{int}(k)\right] = 1 - e^{-\Delta\lambda_k}$$



(a) Cache hit rates for all classes of content is 0 (R's cache is disabled).



(b) Cache hit rates differ for different classes of content.

Figure 6.1: Interest collapsing probability at R

However, taking content caching at R into consideration, the previous equation can be rewritten as:

$$\Pr\left[\mathsf{Coll}_{int}^{R}(k)\right] = \left(1 - p_k^R\right)\left(1 - e^{-\Delta\lambda_k}\right) \tag{6.3}$$

where $\left(1 - p_k^R\right)$ represents cache miss probability. In other words, if the requested content is cached at R, the latter satisfies corresponding interests from its local cache without creating PIT entries. Therefore, interest collapsing does not occur.

We now redefine Δ as a function of the propagation delays incurred by each link on the path $R \leftrightarrow P \leftrightarrow R$ (i.e., the RTT between R and P.) Note that δ_i is the propagation delay of the link connecting r_{i-1} and r_i, where $r_0 = R$ and $r_L = P$. Moreover, p_k^i represents cache hit probability at router r_i and if interests generate a cache hit at r_i, then they are not propagated further.

$$\begin{aligned}\Delta &= 2 \cdot \sum_{i=1}^{i^*}\left(\delta_i\left(1 - p_k^i\right)\right)\\ &= 2 \cdot \sum_{i=1}^{i^*}\left(\frac{l_i}{\alpha_i c}\left(1 - p_k^i\right)\right)\end{aligned}$$

where $\alpha_i c$ represents the propagation speed of link i, and $1 < i^* < L$ is the index of router r_{i^*} where a cache hit first occurs.

However, assuming that in-network caching only happens at the network edge and that δ_L is negligible compared to $\delta_1 + \delta_2 + \cdots + \delta_{L-1}$ (we neglect the effect of caching at r_{L-1}), the cache hit probability in all routers between R and P (not including R) is equal to 0, and

$$\Delta = 2 \cdot \sum_{i=1}^{L}\frac{l_i}{\alpha_i c}$$

Therefore,

$$\begin{aligned}\Pr\left[\mathsf{Coll}^{R}_{int}(k)\right] &= \left(1-p^{R}_{k}\right)\left(1-e^{-\left(2\cdot\sum_{i=1}^{L}\frac{l_i}{\alpha_i c}\right)\cdot\lambda_k}\right)\\ &= \left(1-p^{R}_{k}\right)\left(1-\left(e^{-\sum_{i=1}^{L}\frac{l_i}{\alpha_i}}\right)^{\frac{2\lambda_k}{c}}\right)\\ &= \left(1-p^{R}_{k}\right)\left(1-\left(\prod_{i=1}^{L}e^{-\frac{l_i}{\alpha_i}}\right)^{\frac{2\lambda_k}{c}}\right)\end{aligned}$$

This concludes the proof. □

We analytically analyze Theorem 6.1 in the following setup. For simplicity we use Equation 6.3 for various content arrival rates and propagation delay values between R and P. Since content popularity follows a Zipf distribution, content arrival rate for class $k+1$ is half of that for class k, i.e., $\lambda_{k+1} = \lambda_k/2$. To illustrate the largest possible interest collapsing probability, we assume that requested content (even if popular) is not cached at R.[6] Figure 6.1a shows the collapsing probability of four classes of content $k = [1, 4]$. The reason why the graph only considers propagation delay up to 4 milliseconds is because it is shown in [74] that the virtual RTT (VRTT), which is RTT taken into consideration the existence of caches, for content class $k = 4$ is around 4 milliseconds. We notice that $\Pr\left[\mathsf{Coll}^{R}_{int}(k)\right] \leq 0.15$ for the most popular content ($k = 1$). However, in a more realistic setup where R's cache is taken into consideration, the highest interest collapsing probability is less than 0.05 for content class $k = 2$, see Figure 6.1b. Based on these low probabilities, we conclude that interest collapsing becomes almost useless in practice when caching is present at the edge.

[6]Recall that caching content eliminate the possibility of interest collapsing.

6.1.2 Stateless CCN using Backwards Routable Names

As evidenced above, PITs are unnecessary to provide many of their offered services and simultaneously come at the price of serious infrastructure security problems that have not been addressed. To this end, we introduce a modified CCN architecture without PITs, called stateless CCN.

The main idea behind our stateless CCN design is simple: an interest now includes a new field called Backwards Routable Name (BRN): a routable prefix, similar to an IP source address. BRNs exist in a global namespace much like an IPv6 address. A BRN indicates *where* the corresponding content should be delivered. The corresponding content carries the BRN as its routable name towards the origin of the interest. Thus, with properly configured FIB entries, content is correctly delivered to the origin of the interest.[7] This modification to the CCN architecture is clearly inspired by IP – all packets (interest and content) are forwarded based on addresses they carry and not on network state. However, as we show below, this does not violate CCN's core value of named data being moved through, and stored in, the network.

To illustrate BRN-based forwarding, consider a scenario where a consumer Cr with *topological name* `/edu/uci/ics/gateway/bob` (N_{Cr}) requests content from a producer P with the name `/bbc/news/today` (N_{bbc}). In this case, Cr is the origin of the interest. (As we will show later, it is not mandatory for a consumer to be an origin.) Let $I(N, SN)$ be an interest with the routable name $N = N_{bbc}$ and Supporting Name $SN = N_{Cr}$. Also, let $C(N, SN)$ be the corresponding content object that matches $I(N, SN)$. In this example, assume that $C(N, SN)$ is not cached anywhere.

1. Cr advertises its name N_{Cr} and the routing protocol propagates this information accordingly.

[7] This requires origins to publicly advertise their BRN prefixes and participate in routing.

2. Cr issues $I(N_{bbc}, N_{Cr})$.

3. The network forwards $I(N_{bbc}, N_{Cr})$ towards P according to router FIB entries. At every hop, each router may optionally modify N_{Cr} if needed to preserve routing correctness and consumer privacy (see Section 6.1.3).

4. Once P receives $I(N_{bbc}, N_{Cr})$ it replies with $C(N_{bbc}, N_{Cr})$.

5. Similarly to Step 3, the network forwards $C(N_{bbc}, N_{Cr})$ back to Cr, based on N_{Cr}, using the same interest forwarding strategy.

Several modifications need to be made to the existing CCN architecture and protocol to enable this communication. At a minimum, interest and content object messages should carry two names: one of the requested content and the other of the origin. Contrary to IP, these two names *do not* correspond to a source and destination address. The origin's name serves as a topological address to which the content object should be sent, whereas the data's name serves as a topology-agnostic locator and identifier for the data. Therefore, the addition of this name does not violate the core CCN value that data names are distinct and independent of network locale.

We suggest modifying both interest and content top-level messages to include a new field called `SupportingName` (SN). This field contains the BRN of the interest origin. In the above example, interests and content objects would contain `/cnn/news/today` and `/edu/uci/ics/bob` as N and SN, respectively. Note that content object signatures can be generated in advance by omitting the content's SN field since this is only used for routing purposes. The resulting packet formats are shown in Figure 6.2 in ABNF format. `ValidationAlg` and `ValidationPayload` elements are defined in [225].

```
Message := MessageType PacketName [Payload] [Validation]
MessageType := Interest | ContentObject | ...
PacketName := Name SupportingName
Name := CCNx Name
SupportingName := CCNx Name
Payload := OCTET+
Validation := ValidationAlg ValidationPayload
```

Figure 6.2: Modified stateless packet format

We stress that a content might not follow the reverse path of the proceeding interest due to routing table configurations. In fact, we anticipate that origins might structure BRNs to control the degree of path asymmetry between interest and content messages.

Modified interest and content formats coupled with removing the PIT simplifies fast-path processing. Algorithms 21 and 22 show how a router would process interest and content messages. CS-Lookup represents a CS lookup operation based on N (content name). For clarity's sake, we omit content verification details in all algorithms. Interest forwarding involves a CS miss and FIB lookup whereas content object forwarding involves a CS update and FIB lookup. This is significantly simplified when compared to the traditional forwarding logic wherein interest forwarding requires a CS and PIT miss, PIT insertion, and FIB lookup whereas content object forwarding involves a CS miss, PIT hit and deletion, and CS update.

Algorithm 21 Stateless interest processing

```
1: Input: Interest $I(N, SN]$, arrival interface $F_i$, CS, FIB
2: $C$ = CS-Lookup(CS, $N$)
3: if $C \neq$ nil then
4:     (Optionally) Modify $SN$ to add privacy.
5:     Forward $C$ to $F_i$
6: else
7:     prefix, $F_o$ = FIB.Lookup($N$)
8:     Forward $I(N, SN]$ to $F_o$ based on local strategy
```

Algorithm 22 Stateless content object processing

```
1: Input: Content Object $C(N, SN]$, CS, FIB
2: Cache $C(N, SN]$ with $N$ as the key
3: $F_o$ = FIB.Lookup($SN$)
4: Forward $C(N, SN]$ to $F_o$ based on local strategy
```

6.1.3 Architecture Evaluation

Despite significant research progress over the past five years, the PIT no longer seems to be a practical solution for content object forwarding in CCN. As discussed earlier, router PITs are prone to DoS (specifically IF) attacks. They also store information already available from FIBs (consumer routable prefixes) and enforce unnatural path symmetry in an increasingly asymmetric Internet. (The latter problems remain for the stateless CCN variant of Mirzazad-Barijough et al. [132].) The proposed stateless CCN variant mitigates these problems by specifying the use of source and destination prefixes. To support our claims, we compare the stateful and stateless CCN architectures with respect to aforementioned features. We then discuss both advantages and disadvantages of stateless CCN.

Revisiting the PIT Benefits

Reverse-Path Routing. The proposed stateless CCN scheme requires FIBs to be updated to accommodate BRN prefixes advertised by consumers. It might seem, at first, that this would lead to a tremendous increase in FIB size. However, recall that CCN interest (and now, content) forwarding is based on LPM. In stateless CCN, consumers announce their BRNs only to their first-hop routers (e.g., an access point), which, in turn, combines all its consumers' BRNs and announces an aggregate prefix to neighboring routers, similar to the Border Gateway Protocol (BGP) route-aggregation feature [296]. We will revisit this aggregation feature later.

Also, path asymmetry between interest and content messages in stateless CCN is more compliant with networking and routing practices of today's Internet. As argued in Section 6.1.1, ISPs are likely to adopt an architecture that agrees with their present business model.

Forwarding Overhead. Stateful CCN dictates that, when processing an interest, a router should, in the worst case: (1) attempt to satisfy the interest from its CS, (2) create or modify a PIT entry for the interest, and (3) perform a FIB lookup. Meanwhile, stateless CCN eliminates (2), which reduces the number of operations needed to forward interests. To better understand this reduction, consider the operations needed to forward packets in stateful CCN. For interests, both the PIT and CS must be indexed (separately or together as in [289]) using full interest names. This costs a single lookup plus an additional write (to create a new, or update an existing, PIT entry) if the matching content is not cached. In stateless CCN, the PIT update procedure is removed, thereby improving the efficiency of the forwarding process.

To give an example of the overhead that is saved for this operation, we profiled the PIT lookup procedure for the PARC Metis forwarder [12]. Using a random set of URIs generated from the Cisco data set [11], we added and removed entries in the PIT at varying rates to match a desired steady state. We analyzed the PIT performance when its average number of entries is in the set $\{10, 100, 200, 300, 400\}$. The resulting lookup and insertion time is shown in Figure 6.3. For this implementation, running on a workstation with a 2.8 GHz Intel Core i7 CPU and 16GB of 1600 MHz DDR3 RAM running Ubuntu 14.04, we see that removing the PIT saves an average of approximately 4.5μs across all names in input data set.

Now consider content objects: forwarding requires a single PIT lookup, PIT deletion or eviction, and a CS write operation. In stateless CCN, the PIT index and update procedures are replaced with a FIB lookup procedure. Contrary to interest forwarding, stateless CCN content object forwarding should (in theory) be more expensive than that of stateful CCN. Using the data from [289], which presents a highly optimized software forwarder for NDN

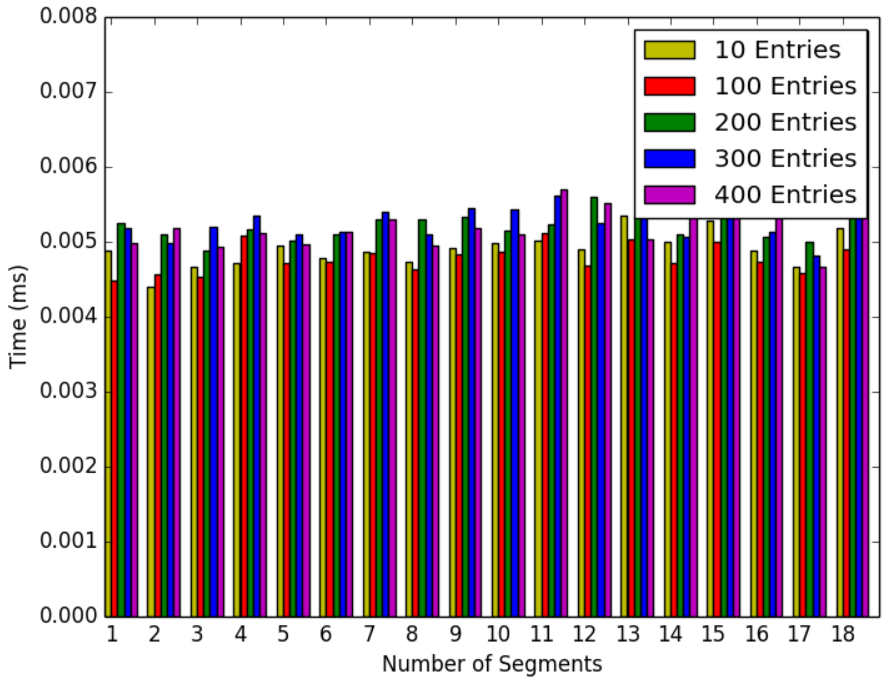


Figure 6.3: Average PIT lookup and insertion overhead

(with the same fundamental forwarding rules as stateful CCN), interests are forwarded at an average rate of 1500 cycles/packet whereas content objects are forwarded at an average rate of 550 cycles/packet. Interest processing requires: CS, PIT and FIB lookups as well as a PIT write operation to create or update an entry. Conversely, content processing requires PIT lookup and write (to remove an entry) operations.[8] The additional FIB lookup in interest processing is responsible for the extra overhead required for forwarding interests. Note that a FIB lookup is much slower than a PIT lookup. The reason is because the former is based on longest-prefix matching and actually consists of multiple lookups for different prefixes. This means that, in stateless CCN, replacing a PIT lookup while processing content with a FIB lookup should *increase* content forwarding overhead, but it would not exceed that of interest forwarding overhead in the standard stateful CCN design.

[8]CS processing also includes a CS write operation to cache the received content. However, it can be done in parallel and not on the fast path.

Flow and Congestion Control. Current receiver driven flow and congestion control algorithms are unaffected in stateless CCN. The only difference is that now routers are unable to compute the RTT for a given interest-content exchange. This prevents fine-grained flow control taking place close to congested links in the network. However, given that many flow control algorithms operate at the edge and do not rely on RTTs collected by routers, this is a tolerable loss.

Content Caching

As mentioned earlier, using BRNs for content routing does not preserve path symmetry. In fact, it encourages path asymmetry. Consequently, content might be cached along a different path than the interest originally traversed. It might seem that adjacent (or nearby) origins for the same content would therefore not benefit from in-network caching. We argue that this is not so. Due to their high processing rates, core routers will most likely not cache content. Meanwhile, consumer-facing routers would handle much less traffic and are thus more likely to cache content. In fact, caching has been shown to be most cost effective at the edges [133], e.g., at tier-3 ISP level. Since nearby consumers share the same edge router, they will all benefit from caching popular content in that router. This observation is supported by the results obtained in [174], wherein it is shown that path symmetry is highest at the edges of the network.

Figure 6.4 shows an example of caching in stateless CCN. The topology has 4 autonomous systems (ASes). AS1 and AS4 are stubs representing tier-3 ISPs, while AS2 and AS3 are transits representing tier-1 ISPs.[9] Interests issued by Cr are forwarded towards P along the dotted (red) path, and content is forwarded back to Cr along the dashed (blue) path. Assuming that caching only occurs near the edges, content sent from P to Cr gets cached in

[9]We ignore tier-2 ISPs for simplicity.

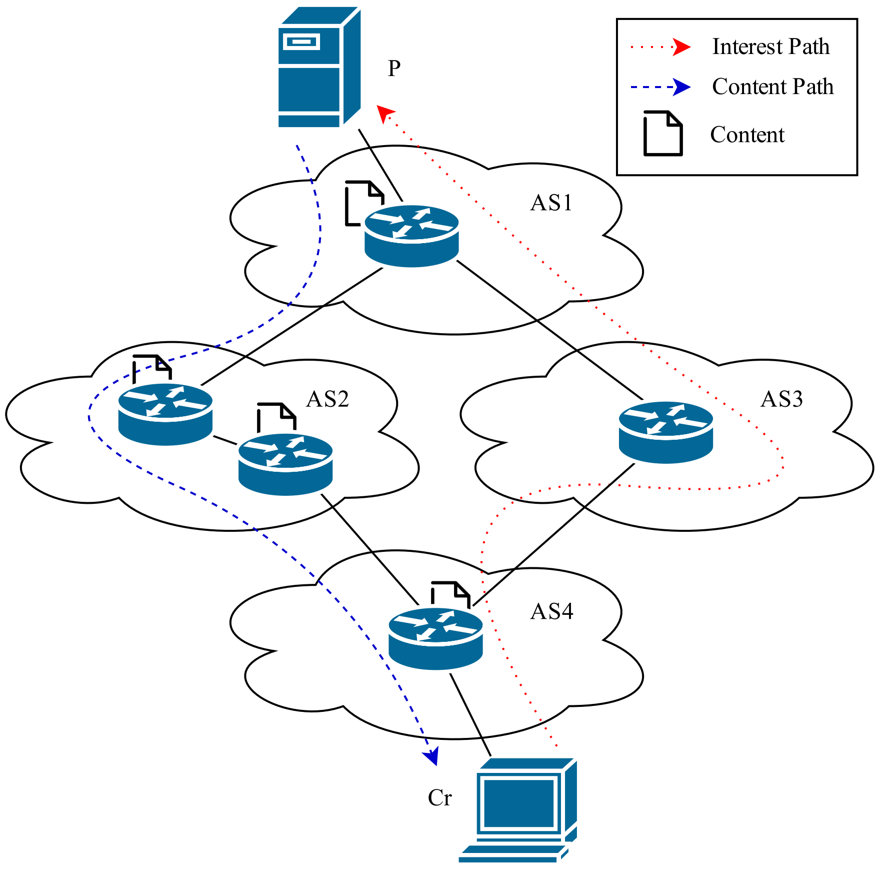


Figure 6.4: Caching in stateless CCN

AS4. Consequently, interests for the same content issued by other consumers in AS4 would be satisfied from AS4 cache(s).

Infrastructure Security

We now discuss both beneficial and problematic infrastructure security issues in stateless CCN.

FIB Explosion. Stateless CCN necessitates that FIBs contain entries for origin and producer prefixes. Scalable name-based routing is still a topic of research for CCN and related architectures. Aggregation (as described later in this Section) helps reduce the number of entries in a FIB if those entries are topological; it does not offer much help for producer prefixes which are, in theory, agnostic to topological information [277]. Fortunately, since BRNs are necessarily topological, they can be aggregated similar to the way in which IPv4 addresses are aggregated behind a NAT.

Interest Flooding. Stateless CCN mitigates this attack by eliminating its root cause – the PIT. Without per-request state in routers, this attack vector is removed. By and large, this is the primary benefit of stateless CCN.

Reflection Attacks. Interest and content path symmetry in CCN prevents reflection attacks. However, in stateless CCN, BRNs serve as a *de facto* source address in interest, and destination in content, messages. Thus, reflection attacks re-appear. Fortunately, the ingress filtering technique described in [117] can mitigate them.

Cache and Content Poisoning. Content authentication in stateless CCN is identical to that in the stateful CCN architecture. It is done by producers signing content objects or using Self-Cerftifying Names (SCNs) [48]. Regardless of the method, all content *must* be verified by consumers. However, verification is not mandatory for routers, for several reasons; see [150] for more details. Lack of in-network content verification opens the door for content poisoning attacks [142]. Moreover, due to possible path asymmetry in BRN-based content forwarding, content poisoning countermeasures that work in the current CCN architecture do not apply anymore.

The PIT enables a router to apply the so-called Interest-Key Binding (IKB) rule [150], whereby consumers and producers collaborate to provide routers with enough (minimal) trust information to perform content verification. This information is currently stored in the PIT. However, as mentioned above, path asymmetry renders the IKB impractical *for the initial data request*. In stateless CCN, a router might receive (unsolicited) content without prior interest traversing the same path. If such content is returned on a path different from the original interest, routers cannot trust any information it carries. However, this does not prevent a router from opportunistically caching content it forwards. In doing so, the router can apply the IKB rule to *subsequent* requests for the *same cached data* without forwarding the interest upstream. (The difference here is that, in stateful CCN, the IKB rule can be applied to verify content before it is inserted into the cache, whereas now it must be applied once, and only once, the first cache hit occurs.)

Origin Privacy. Lack of source addresses in stateful CCN enables a degree of consumer privacy. If origins are consumers, then BRNs in stateless CCN negate this benefit much in the same way that global IPv6 addresses harm user privacy [236]. (However, as we will discuss, in an ideal deployment of stateless CCN, origins would *not* be consumers.) To mitigate this problem, a router R can assign a random identifier to each of its downstream consumers to be used as part of their BRN and could overwrite the BRN in all ingress interests based on this pseudonym (in line 4 of Algorithm 21). For example, instead of including an BRN as `/edu/uci/ics/consmerA`, the gateway could set the BRN as `/edu/uci/ics/$r$`, where r is a random string that is rotated on a regular basis. The procedure to modify a BRN based on the arrival interface at a router is detailed in Algorithm 23. One important benefit of this strategy is that r can be rotated at random and independent of other routers so that consumers BRNs do not appear fixed upstream, thus mitigating interest linkability [149].

Algorithm 23 Stateless BRN obfuscation

```
1: Input: $SN$, arrival interface $F_i$, FIB, $r$
2: $(\mathsf{prefix}, \mathsf{F_d}) := \mathsf{FIB.Lookup}(SN)$
3: if $F_d = F_i$ then
4:     $\mathsf{index} := |\mathsf{prefix}|$
5:     $SN_{\mathsf{index}} = H(SN_{\mathsf{index}}||r)$
6: Return $SN_{\mathsf{index}}$
```

Deployment Issues

Stateless CCN is an alternative to the current stateful CCN. They need not replace one another. In fact, as we have designed it, they can co-exist. Consider the following scenarios:

1. Cr includes a BRN (SN) in an interest and upstream routers forward it as necessary. Stateful routers create PIT entries and stateless routers do not. In both cases, the interest is forwarded according to the FIB using content name N. Upon receipt of a content message, a stateful router uses its PIT to forward the content downstream, while a stateless router does that using the FIB and SN. In this case, stateful forwarders simply ignore the SN fields in both interests and content objects. This makes the proposed stateless CCN backwards compatible with the current CCN architecture.
2. Cr issues an interest as per current CCN rules. If a stateless router receives such an interest, it generates a NACK indicating that the interest cannot be forwarded further. To handle this NACK, some downstream node must provide a BRN for the interest and re-forward it as needed. This node can be the consumer or an AS gateway, i.e., a router that can forward packets to and from other ASes, acting as the origin.

Any node that satisfies an interest must honor its version (stateless or stateful) when producing a response. For example, if a producer (or a caching router) receives an interest with an BRN, it must reply according to stateless CCN by keeping both N and SN in the corresponding content.

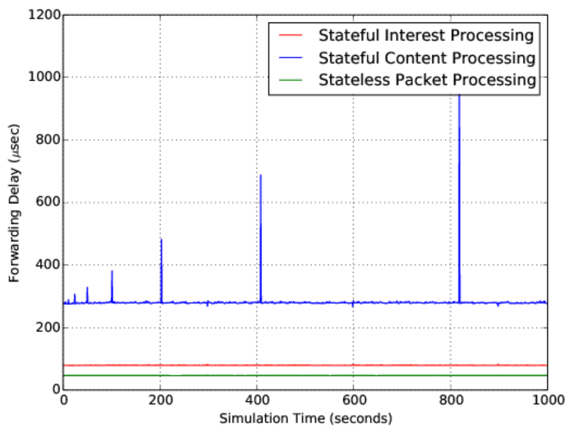
We envision a hybrid approach where stateless CCN is deployed at the network core and stateful CCN at the edge. This aligns well with the CCN edge-caching strategy [133] and current path asymmetry in the Internet's core [174]. Edge routers in consumer-facing ASes will possess both caches and PITs to aid with content verification. When consumers issue interests, they first traverse through stateful CCN routers in a consumer-facing AS. When they leave this AS, the gateway, acting as the interest origin, supplies a BRN before forwarding upstream. Such interests will not induce any PIT state at the network core.

This hybrid approach has several powerful advantages. **First**, consider the benefits of the hybrid deployment with respect to congestion control and mobility. If stateful CCN is deployed near the edge, then fine-grained congestion information can be collected and conveyed to consumers to adjust their transport protocol state accordingly. Moreover, as PITS are deployed in stateful CCN near the edge, where mobility events take place, existing proposals to handle mobility such as the trace-in-PIT proposal of Zhang et al. [345] can be used. **Second**, it provides a native IF attack recovery mechanism. If R implements a PIT but does not have enough resources to create a new entry for an interest, R can respond with a NACK similar to what is described above. Consumers, then, issue interests according to stateless CCN guidelines. The disadvantage of this approach as an effective IF attack countermeasure is that (1) it is reactive, so it can only be used after the attack occurs, and (2) it incurs an additional end-to-end latency since consumers (or downstream routers) need to reissue stateless CCN interests. **Third**, it allows forwarder state to scale where it scales best: *at the edge*. IF attacks are a problem specifically because the state does not scale well throughout the entire network. However, in smaller subnets, this state can be much better managed without falling victim to a DoS attack.

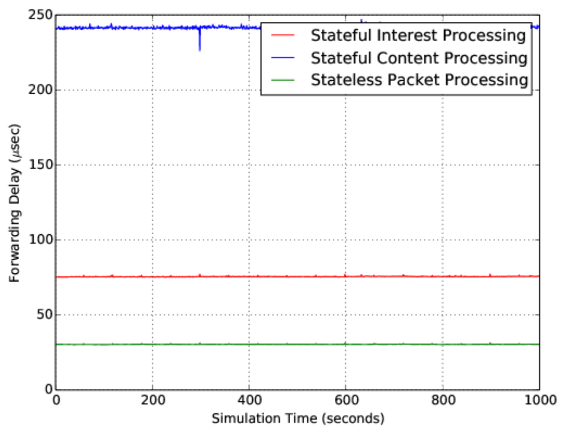
We also note that interests can cross stateless and stateful network boundaries with ease. If an interest travels from a stateful to a stateless network, the gateway must supply a BRN before forwarding the interest. The gateway is then considered the origin of the interest. Similarly, if an stateless interest arrives at a stateful gateway, the latter must store the BRN (in the SN field) in the corresponding PIT entry and *subsequently remove it from the interest*. This is necessary if the interest will cross, multiple times, across a stateful and stateless boundary.

6.1.4 Performance Assessment

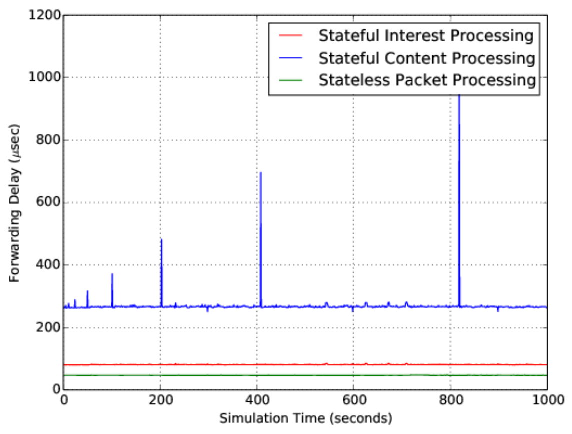
We now evaluate performance of the stateless CCN in relation to stateful CCN. The key metric we use is the degree to which forwarding overhead is affected by stateless routing. To do this, we modified the ndnSIM 2.1 simulator [215] to support the stateless CCN architecture



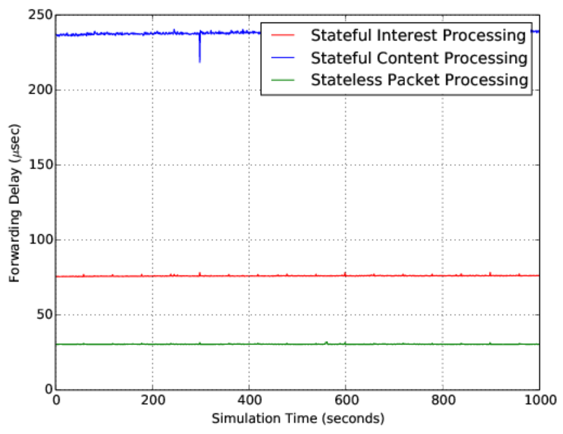
(a) Processing overhead for DFN topology with 160 consumers and routers with caches.



(b) Forwarding overhead for DFN topology with 160 consumers and routers without caches.



(c) Processing overhead for AT&T topology with 160 consumers and routers with caches.



(d) Forwarding overhead for AT&T topology with 160 consumers and routers without caches.

Figure 6.5: Forwarding overhead in stateful (red, blue) and stateless (green) CCN variants

proposed in Section 6.1.2. Specifically, we modified the NDN Forwarding Daemon (NFD) [26] to forward interests and content objects based on names and BRNs.

We then simulated topologies based on Deutsches ForschungsNetz (DFN), the German Research Network [5, 6] and AT&T core network (selected due to the size and diverse node distribution). Each topology consists of 160 consumers[10], a single producer connected to

[10]Each consumer node in the figures consists of 10 actual consumers.

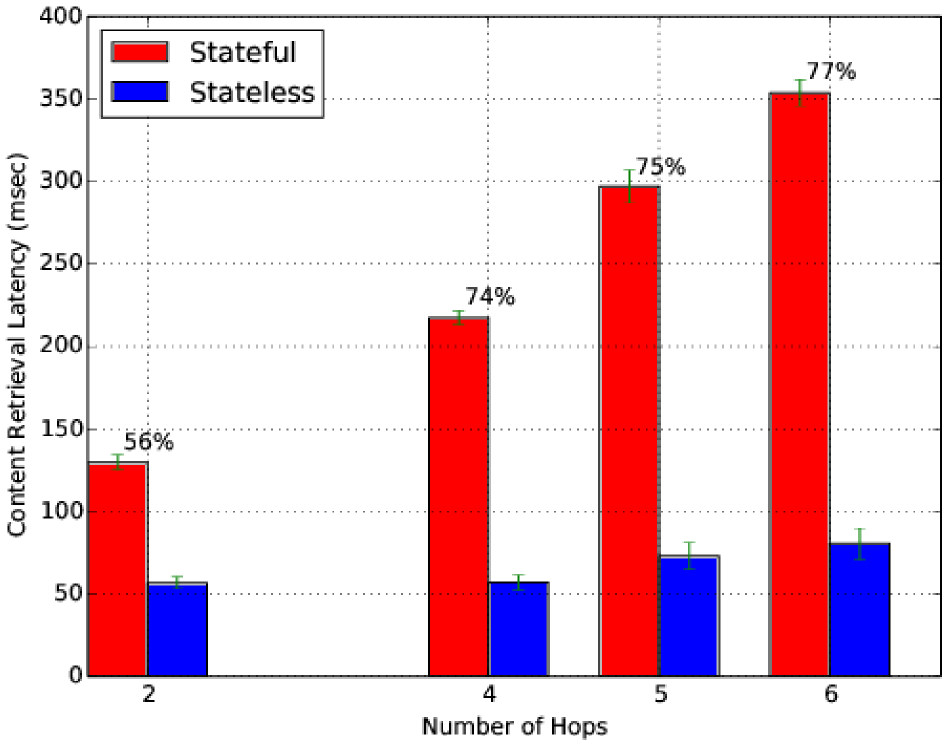


Figure 6.6: Stateful and stateless content retrieval latency

one of the edge routers, and multiple routers (more than 30). Each consumer generates 10 interests per second, with a random suffix so as to avoid cache hits. This is done to force interests to traverse the complete path to the producer and therefore maximize the amount of processing that takes place in forwarders in the upstream and downstream paths. This captures the worst-case scenario. In our experiments, neither the consumers nor the producer are equipped with a cache. We do, however, assess the forwarding overhead differences in the presence and absence of router caches.

The results of both experiments are shown in Figure 6.5. Figures 6.5b and 6.5d capture the overhead imposed by packet forwarding regardless of caching effects. They show that caching adds an overhead of approximately 20% to content processing; compare the blue lines in Figures 6.5a and 6.5b, and Figures 6.5c and 6.5d. Moreover, in both topologies, we observe that stateless packet forwarding imposes less overhead on routers compared to stateful interest and content forwarding. This is due to the fact that stateless packet forwarding does not require any PIT operations. The savings are quite significant, especially, for cache-less core routers that might process packets at rates of 100Gbps and over.

Furthermore, the overall content retrieval latency improves with stateless forwarding. Figure 6.6 shows a comparison of the RTT performance for both forwarders in the DFN topology. In this experiment, consumers always request unique content in order to avoid cache hits.[11]. On average, the content retrieval latency improves by more than 50%. The improvement reaches 77% for paths consisting of 6 hops. Although these results are dependent on the forwarder implementation inside ndnSIM, the results align with intuition and our previous experiment which show that stateless packet forwarding will, on average, improve due to the absence of the PIT.[12]

To justify this claim, we revisit the argument of Section 6.1.3 which states that the forwarding cost for stateless content objects will be closer to that of stateful interests due to the former's need for a FIB lookup instead of a PIT lookup. To estimate this overhead, consider the forwarder in [289]. With a 64MB FIB and 2MB PIT, which can forward interests at a rate of approximately 1.2 MP/s (million packets per second), 2.3 MP/s, 4.2 MP/s, and 5.9 MP/s with 1, 2, 4, and 8 threads on a 2GHz core. In comparison, the content object forwarding throughputs are approximately 1.4 MP/s, 6.1 MP/s, 12.1 MP/s, and 14.2 MP/s under the same conditions. If the cost to forward content objects in the stateless variant is equal to that of interests, then the forwarding rate degrades by 14.3%, 62.3%, 65.3%, and 117.9%, respectively. These values capture the cost of the FIB lookup operation. However, given that stateless routers are no longer susceptible to DoS attacks, we deem this cost justified.

[11]We do not take caching into consideration to eliminate any randomize effects (caused by different eviction policies) on content retrieval latency.

[12]We say on average since interests are forwarder more quickly whereas content objects necessarily require more time due to the FIB lookup in lieu of a PIT lookup.

6.2 Efficient and Opaque Network Names

CCN name-based forwarding has several undesirable consequences. First, it places unbounded computational burden on routers when forwarding interests. Existing high-speed FIB designs use data structures that range from hash tables [289] to prefix tries [98, 68]. All of them must account for variable length names and name segments. As discussed above, this can be abused as a form of DoS attack on routers. Second, and perhaps more importantly, the current design forces application layer semantics – through names – into the network.

In this section, we propose and evaluate a way to transparently decouple application-layer and wire-encoded, or network-layer, names in CCN.[13] Our approach yields network-layer names, or simply network names, that are deterministically derived from application-layer names, formatted according to the CCNx URI scheme [229]. This yields several benefits: (1) less application data percolates into the network, (2) packets carry less variable-sized names and name segments, and (3) forwarding logic is simplified and, consequently, improved. We evaluate the impact of network names on CCN entities (consumers, producers, and routers) and related protocols, such as routing. We also address security considerations related to our network names. We then present a comprehensive analysis of their characteristics and statistical properties to show they are functionally equivalent to standard application names. We then present FIB performance results with and without network names. We find that network names improve processing times across all data structure and algorithm variants considered.

6.2.1 Data Plane Attacks

In this section we discuss data plane attacks on CCN routers. We argue that each dataplane component – caches, PITs, and FIBs – are all vulnerable to trivial attacks due to their need,

[13]This is also discussed by Ghali et al. in [145].

by design, to process unbounded and untrusted data. In this section we use the following notation. Let N be a name. We refer to the i-th segment of N as N_i. $|N|$ is the number of segments in N, whereas $||N||$ is the total size in bytes of N.

PIT and CS Attacks

The PIT and CS are vulnerable data structures since they are indexed using full interest names. (Fortunately, PITs and caches can share an index, so full names need only be processed once.) The standard approach to implementing a PIT and CS index is with a hash table. This means an attacker can inflate interest names and force routers to hash an unbounded number of (name) bytes. Due to variable length names and interest fragmentation, these names are allowed to exceed standard link MTUs, meaning that the ultimate bound on name length is that which is imposed by the packet format – 64KB [225].

FIB Attacks

The FIB is indexed with a name N. The result of this query is a FIB entry, if one exists, that has the longest matching prefix N_p. Since segments are variable length, this can be viewed as a string matching problem. Consider the following two ways this can be solved: (1) Deterministic Finite Automatons (DFAs), e.g., a trie, or (2) hash tables (HTs). With DFAs, computation is bounded by DFA size. For example, suppose a router R needs to look up N^{bad} =`/xx...xx/bar` in a FIB with only entries for N_p^1 =`/a/b/c` and N_p^2 =`/d/e/f`. Using a DFA, R only needs to examine the first byte N^{bad} to determine there is no matching entry. In contrast, a router using a hash table FIB would need to compute the hash of N_1^{bad} and then perform a table lookup before learning there is no match. Clearly, in this case, DFAs are superior. However, this is not universally true.

Attacking FIBs requires exercising algorithm worst-case lookups. Let N^c_{max} and N^b_{max} be the longest prefixes in terms of segments and bytes stored in a FIB, respectively, and let N^* be an input name. For a DFA-based FIB, worst-case lookup occurs when N^* is a byte-wise prefix of N^b_{max}, i.e., it matches the longest byte-wise prefix accepted by the DFA. For a HT-based FIB, worst-case lookup occurs when N^* matches the first $|N^c_{max}| - 1$ segments of N^c_{max}. For example, let N^c_{max} = `/foo/bar/baz`. A worst case input name would then be a name of the form N^* = `/foo/bar/<random>`, where ¡random¿ is a random segment string. In this case, all segments must be checked to determine whether there is a match. This maximizes the number of bytes processed in the input name.

It is also possible to construct names that will exercise worst-case paths for both DFA- and HT-based FIBs. Recall that applications are free to choose whichever name prefix they want to register, regardless of their (topological) location. Name prefixes are not bound to topological locations, even though name prefixes are used as locators. Due to name aggregation, a router is free to collapse a long name prefix down the shortest disambiguating prefix needed to correctly route matching interests. This means that if two separate producers registered prefixes N_1 = `/xx...xx-1/bar` and N_2 = `/xx...xx-2/bar`, both must be stored for disambiguation. If a router R then received an interest with name N = `/xx...xx-3/bad`, it would need to process every byte of the first name segment to determine there is no match, regardless of whether R uses a DFA- or HT-based FIB.

In short, *FIB lookup complexity is a function of application-chosen names.* And since application names are not bound to any constraints other than the packet format, this is exploitable.

6.2.2 Network Names

In the previous section we discussed why application names are a security concern for routers. We now describe our network name construction that mitigates or alleviates several aforementioned problems.

Let $N = [N_1, N_2, \ldots, N_k, S_1, \ldots, S_l]$ be a CCN application name as per [229] where k segments $N_1, ..., N_k$ are used for locating and identifying content and l segments $S_1, ..., S_l$ carry application-specific information. We define segments containing application information as *any segment* that is not of the type `T_NAMESEGMENT`. This includes segments with the type: `T_PAYLOADID`, `T_VERSION`, `T_CHUNK`, and any other application-specific type to be defined. For example, the name `/edu/uci/ics/csdepart.html/T_VERSION=x02` can be represented as the segment vector [`edu`, `uci`, `ics`, `csdept.html`, `T_VERSION` $= 0x02$] with $k = 4$ and $l = 1$, where the version segment is the only piece of application information. As previously described, an interest carrying this name would have the following TLV encoding:

```
(T_NAME
    (T_NAMESEGMENT  “edu”)
    (T_NAMESEGMENT  “uci”)
    (T_NAMESEGMENT  “ics”)
    (T_NAMESEGMENT  “csdept”)
    (T_VERSION      0x02)
    )
```

We propose the use of network names by modifying this current format. In particular, each network segment of the name is replaced by the fixed-size output of a mapping function $\mathsf{T}(\cdot)$ computed over *all prior segments in the name.* In other words, the i-th name segment is now a fixed-size mapping computed over the first i segments. We specifically omit segments which are *not* part of the name locator, i.e., only segments of type `T_NAMESEGMENT` are included in this computation. The reasons for this requirement are discussed in Section 6.2.3. The actual mapping is computed as:

$$\begin{aligned}
\bar{N}_1 &= \mathsf{T}(N_1), \\
\bar{N}_2 &= \mathsf{T}(N_1, N_2), \\
&\dots, \\
\bar{N}_k &= \mathsf{T}(N_1, \dots, N_k)
\end{aligned}$$

Using this representation, the network name $\bar{N}$ for a given application name N is represented as $\bar{N} = [\bar{N}_1, \bar{N}_2, \dots, \bar{N}_k, S_1, \dots, S_l]$. The network name *replaces* the regular `T_NAME` field in CCN packets – it is not carried in a per-hop header or additional encapsulation layer. With a suitable $\mathsf{T}(\cdot)$, the relationship between N and $\bar{N}$ forms a bijection between the application and network namespaces so long as there are no collisions in $\mathsf{T}(\cdot)$.

We also require the network name include an additional *name fingerprint* segment N_p, computed as the mapping of the full name N with all additional identifiers, i.e., $N_p = \mathsf{T}(N||K_{ID}||C_{ID})$, where K_{ID} and C_{ID} are KeyId and ContentId, respectively. As described in Section 6.2.3, N_p is used for PIT and CS lookup operations. As a consequence of removing application names from packets, content digital signatures must be generated and verified using N_p. This is discussed later in Section 6.2.6. Algorithm 24 shows the deterministic procedure for mapping an application name to a network name with N_p.

Following the CCN TLV-encoding in [225] and assuming $\mathsf{T}(\cdot)$ is SHA-256, the network name and fingerprint can be represented as:

```
(T_NAME
    (T_FINGERPRINT      $N_p$)
    (T_MAPPED_SHA256 $[\bar{N}_1, \bar{N}_2, \bar{N}_3, \ldots, \bar{N}_i, \ldots, \bar{N}_k]$)
    (T_TYPE             $S_1$)
    ...
    (T_TYPE             $S_l$)
  )
```

where T_TYPE is replaced with the appropriate type for the corresponding segments.

We use the type T_MAPPED_SHA256 to denote a list of name segments transformed using SHA-256. We make the mapping function explicit to allow for agility. The length of the TLV-encoded segment (not shown in the above representation) is the total byte length of all $\bar{N}$ values. Given this length and output size of $\mathsf{T}(\cdot)$, the number of $\bar{N}$ values can be computed. In other words, using a fixed-size output $\mathsf{T}(\cdot)$, segments with type T_MAPPED_XXX are of fixed size. Therefore, the type and length of each $\bar{N}$ segment do not need to be encoded.

Algorithm 24 Network name mapping

```
 1: Input: $N$, $\mathsf{T}(\cdot)$
 2: Output: $\bar{N}$
 3: $\bar{N} = \emptyset$
 4: $k$ :=Number of T_NAMESEGMENT segments in $N$
 5: for $i = 1$ to $k$ do
 6:     $seq := N_1 || \ldots || N_i$
 7:     $\bar{N} = \bar{N} \cup \mathsf{T}(seq)$
 8: $\bar{N} = \bar{N} \cup [S_1, \ldots, S_l]$
 9: $name := N_1 || \ldots || N_l || S_1 || \ldots || S_l$
10: $N_p := \mathsf{T}(name)$
11: $\bar{N} = N_p \cup \bar{N}$ return $\bar{N}$
```

$\mathsf{T}(\cdot)$ Function Criteria

In the proposed naming scheme, $\mathsf{T}(\cdot)$ is used for two purposes:

1. Computing values in `T_MAPPED_XXX` segment of a network name.
2. Computing the name fingerprint N_p used by routers for PIT and CS lookups and for signature generation and verification.

Based on this, a sensible instantiation of $\mathsf{T}(\cdot)$ is a hash function. This is because the FIB lookup actually computes multiple hash values for all prefixes of a received interest name, regardless of the underlying data structure used. Currently, most CCN routers use non-cryptographic hash functions (i.e., those that are not collision-resistant) in the FIB for LPM operations. Despite the non-negligible probability of collisions in such hash functions, routers can resolve collisions since application names are explicitly included in interests. However, in the proposed scheme, application names are replaced with fixed-size segments computed using $\mathsf{T}(\cdot)$. Thus, if a collision occurs, a router cannot resolve it, which might result in an interest being forwarded on the wrong interface(s).

One way to cope is by carefully choosing names that do not cause collisions in $\mathsf{T}(\cdot)$. However, it is clearly infeasible for producers to be aware of all existing content names. Another approach is to pick a cryptographic $\mathsf{T}(\cdot)$ that offers collision-resistance. In practice, a well-known cryptographic hash function, e.g., SHA-256, can be used to provide this property, in addition to other cryptographic characteristics, such as one-wayness.

We also note that the output of the cryptographic hash function can be truncated to any output size. We denote $\mathsf{T}(\cdot)$ truncated to s most-significant bits as $\mathsf{T}_s(\cdot)$. With $\mathsf{T}_s(\cdot)$, the corresponding network name segments will carry a type of the form `T_MAPPED_XXX_S` where `S` is digest size, e.g., `T_MAPPED_XXX_64`. Note that output truncation should only be used when computing network name segments and **not** for generating N_p in order to provide the

Table 6.1: CCN entity network name impact.

Entity	**Impact**
Consumer	**Increased** online processing to compute network names (interests)
Producer	**Increased** storage for reverse mapping (interests) **Increased** off-line processing to compute network names (content)
Router	**Faster** FIB lookup with pre-computed name prefix hashes (interests) **Faster** PIT and cache lookups (interests) **Faster** PIT lookups due to lack of name hashing (content) **Decreased** storage requirement due to fixed-size N_p, instead of arbitrarily long names (content)

Table 6.2: CCN packet network name impact.

Packet	**Impact**
Interest	**Longer** name TLV encoding; based on $\mathsf{T}(\cdot)$ size, see Section 6.2.4
Content	**Shorter** (fixed-size) N_p, instead of complete application name

least collision probability for PIT and cache lookups. In Section 6.2.4, we analyze collision probabilities for $\mathsf{T}_s(\cdot)$ for different s values.

In the rest of this section, we use the terms *mapping function* and *hash function* interchangeably when referring to $\mathsf{T}(\cdot)$.

6.2.3 Network Name Integration and Ramifications

The proposed naming scheme has obvious implications for network entities as well as management and control functions, such as routing. In this section, we investigate the impact on end-hosts (consumers and producers), network entities (routers and forwarders), and management functions (routing protocols). Results are summarized in Table 6.1. Implications for CCN packets are summarized in Table 6.2. All of these are considered in detail in Section 6.2.4.

Name Translation at End-Hosts

End-hosts are the primary entities affected by the proposed naming scheme since they are tasked with translating between application and network names. In practice, this mapping would be performed by a component between the application and network layers.[14]

Consumer Mappings The impact on consumers is one name mapping per interest. However, since this operation is not on the fast-path, we expect no performance penalty for consumers. To justify this claim, we assessed the overhead for translating all names in the Cisco URI dataset [11] with the (unoptimized) PARC CCNx libraries [1]. This dataset consists of $13,549,122$ unique URIs. The average length of each URI is 57.4B, with a median length of 52B, and a standard deviation of 33.182B. Across all URIs, the average number of segments is 6.67 with a median length of 6, and standard deviation of 2.212 segments. The average, minimum, and maximum time to compute the mapped version of each name is $1,029.279$us, 3.812us, and $2,474.567$us, respectively. These numbers were generated on a desktop machine with an Intel 2.8 GHz Core i7 processor.[15] This is just over one millisecond to map a single name, which is still less than most network I/O overhead. Given the average size taken across all URIs, the average, minimum, and maximum cycles/byte throughput for this mapping is $1,577.688$c/b, $1,218.037$c/b, and $3,494.538$c/b, respectively. This is far from optimal given that the modern Intel Haswell chipsets can compute SHA-256 digests at a throughput of roughly 8.59c/b [161]. Therefore, we conclude that, given a proper modern implementation, the mapping process at the consumer incurs negligible overhead.

[14]Inverting this process is effortless at since the same stack component that is responsible for mapping N into $\bar{N}$ would remember this relationship and perform the inverse mapping when the content object response returns. The amount of state required for this procedure is directly proportional to the amount of pending interests at a consumer.

[15]The code is available at `https://github.com/chris-wood/network-names`.

Producer Inversions Network names have two implications for producers: (1) handling, and responding to, incoming interests and (2) generating content. Since our goal is to make network names transparent to applications, the producer must be able to determine the original application name from a network name contained in an incoming interest.[16] Therefore, the producer must maintain the reverse mapping from network to application name prefixes that it published.[17]

For this to work, we assume that the producer knows all content prefixes under which it publishes content. In other words, the *network locator portion* of the namespace for which a particular producer is responsible must be well-defined. Furthermore, most producers would need to maintain an index that maps application names to content objects in order to respond to interests. The cost of the network-to-application name mapping would, in the worst case, double the size of this index. Since the size of hash table key-sets in data repositories is typically negligible (compared to size of the actual data), this should not result in any significant barrier.

Assuming the producer maintains this reverse mapping, suppose it receives an interest for $\bar{N}$ that has k mapped segments, l application segments, name fingerprint N_p and optional payload. Using the inverse mapping, it can invert the first k segments, and thus recover the original full name used by the interest-issuing consumer's application. This process does not distinguish between interests for static and dynamic content, since a name for either type of content must have a prefix corresponding to at least k leading segments, i.e., the network locator portion of the name. In case of interests for dynamic content, the "dynamic" portion of the name is the suffix that has at most l segments. Recall that this portion of the name is not mapped, i.e., it remains "human-readable" as in standard CCN names.

[16]If the network name can be used to index into a repository for statically generated content, then inversion is not necessary.

[17]Note that the producer can not compute the inverse based only on network prefixes if $\mathsf{T}(\cdot)$ if a cryptographic hash function, as we propose.

One drawback of imposing this reverse mapping is that $\mathsf{T}(\cdot)$ must be fixed and adopted by all consumers and producers. This limits the scalability of the proposed naming scheme and does not facilitate seamless evolution, or replacement, of $\mathsf{T}(\cdot)$. One obvious way of relaxing this requirement is to support multiple $\mathsf{T}(\cdot)$ functions. To do so, producers should be able to map *any* received network name format to the requested content. This would require a separate index for each supported $\mathsf{T}(\cdot)$.

As far as the second implication for producers, recall that, when forwarding a content object, its name is used only for **exact matching**, i.e., there is no LPM as with interests. This means that the producer can replace the full network name with just N_p. This appreciably reduces router overhead and fixes the overall name size in the content object header. This is very beneficial when the size of a content's name exceeds that of the payload, which might be the case for small contents produced by IoT sensors [72] or NACKs [86].

Forwarding Implications

Figure 6.7 shows the computations needed to forward an interest per standard CCN in a forwarder that processes application names with a hash-based FIB. For an n-segment name, a forwarder must compute at most n hashes. The complete name hash is used to index the CS, PIT, and FIB, while the remaining $(n-1)$ hashes are used to index only the FIB (in the worst case).

Clearly, FIB lookup is the most expensive operation as it requires the most hashes. The LPM algorithm is typically implemented using Bloom filters (BFs) in hardware-based forwarders. This is because the n independent hash functions and lookups can be done in parallel. However, in software, BFs are not appropriate since the BF index operation is necessarily sequential. To the best of our knowledge, So et al. [289] present one of the leading software-based techniques for FIB lookup based on hash tables. Specifically, to lookup a name prefix

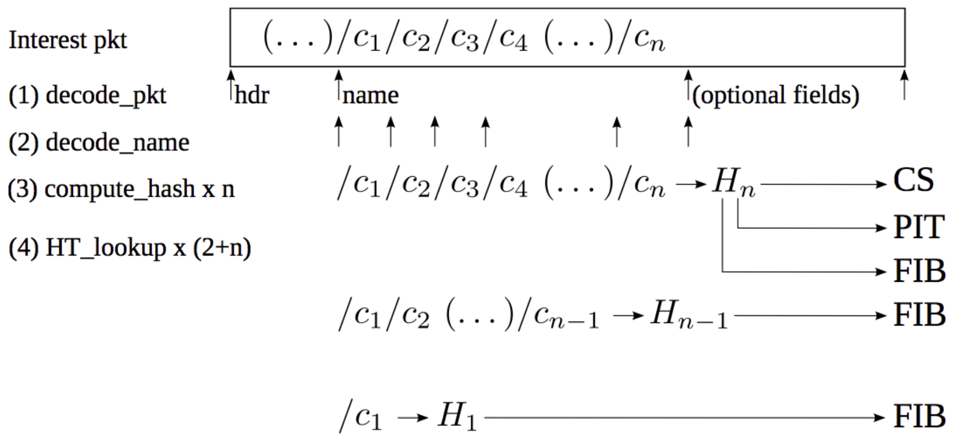


Figure 6.7: Interest forwarding with hash tables [289]

with k segments, the technique in [289] performs a single hash (for each segment), one modular reduction (based on the number of hash table buckets), and a memory lookup. The penalty of reading memory not in the data cache is over 100 memory stall cycles.[18] For example, the cost of computing a 64-bit hash with SipHash [289] is 5.94c/b. In this case, the cost of hashing a name exceeds that of the penalized lookup if the name being hashed exceeds 16 bytes. Thus, for names longer than 16 bytes, hash function computation accounts for at *least half* the cycles needed for hash table lookup.

This is where the motivation for the proposed network naming scheme becomes most apparent: our proposal removes hashing overhead from: (1) FIB hash table lookup and (2) complete name hash (for CS and PIT lookups). In the former case, this reduces the fast-path overhead by at least the cost of the mapping function (depending on the name length). In the latter case, PIT and CS lookups are simplified for reasons discussed above. Since both PIT and CS are effectively indexed by N_p, this is the only information required to read and write to these data structures. Also, since N_p is always included in interest and content packets, this computation is removed from the fast path.

[18]If the memory location is in the cache, there is no stall penalty.

As with producers, routers would have to support a range of $\mathsf{T}(\cdot)$-s by accommodating different $\mathsf{T}(\cdot)$ output sizes. This can be done by (1) re-hashing network names when indexing any of these tables, or (2) maintaining a separate index for each supported size. In case (1), we claim that the router overhead is still reduced since the input to these re-hash functions is always a fixed size, not exceeding 32 bytes; application names may very well exceed this size and therefore take more time to hash. In case (2), although table management and maintenance might become more complicated, there will be no need for re-hashing. We implemented a variety of FIB algorithms and compared their performance with and without network names. Section 6.2.5 discusses our methodology and results.

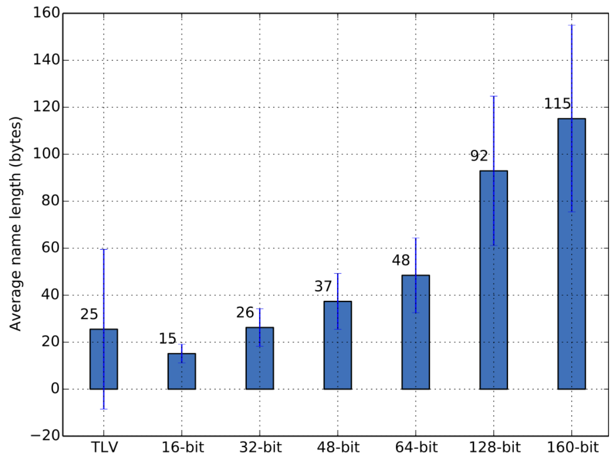
Routing Protocols

Our proposal does not require any changes to CCN routing protocols. Producers announce application name prefixes for namespaces they control. Routers then hash received prefixes to populate FIB tables. To guarantee correct interest forwarding, the function used by routers to fill FIBs and $\mathsf{T}(\cdot)$ used by consumers when generating network names *must* be the same. This is the same requirement for the producer-generated reverse mappings mentioned in Section 6.2.3. Similarly, multiple $\mathsf{T}(\cdot)$ functions can be supported by routers. This would require routers to have multiple FIB entries per prefix – one for each supported hash function. Clearly, this represents a trade-off between scalability and mapping agility, versus increased FIB table sizes. Alternatively, producers could advertise prefixes already transformed into network name formats.

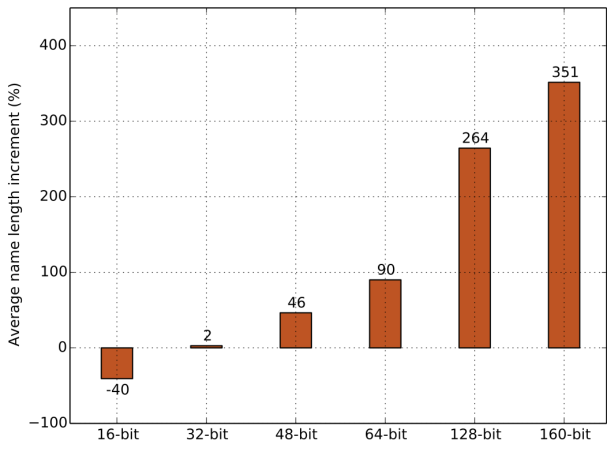
6.2.4 Experimental and Statistical Analysis

In this section, we compare our proposal to the standard CCN naming scheme, aiming to show that network names are functionally equivalent to application names, (in terms of uniqueness, size, and distribution properties) and enable correct forwarding with improved performance. We use the Unibas dataset from the The Content Name Collection [11], which contains *unique* URLs submitted by users to URL shortener websites. We convert these URLs into a CCN-compatible name format. For example, the URL `http://www.domain.com/file.html` is converted into the CCN name `/com/domain/file.html`. Table 5.2 shows some characteristics of the Unibas dataset.[19]

Table 5.3 illustrates name distribution per number of segments in each name. It shows the number of names in the dataset that contain n segments for $n \in [1, 20]$. Note that: (1) almost 30% of names have 5 segments, and (2) names of up to 20 segments account for 99.876% of the Unibas dataset. Also, we use application names that only contain locator segments, (i.e., with no identifier segments) because such identifiers are not easily distinguished from locator segments outside the application layer, and they are included in network names exactly as they appear in application ones.



(a) Unibas average name length and standard deviation for various $\mathsf{T}(\cdot)$ sizes.



(b) Average name length increment for various $\mathsf{T}(\cdot)$ sizes.

Figure 6.8: Network and application name network overhead comparison

[19]For practical reasons, we truncate names beyond the maximum size of 2,000 bytes and 80 segments [16].

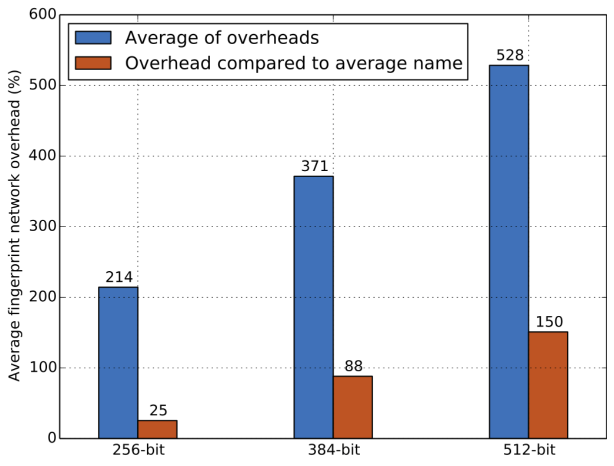


Figure 6.9: Average name fingerprint network overhead

Table 6.3: Prefix distribution per # of prefix segments

Number of prefix segments n	**Number of *unique* prefixes** m	**Percentage**
1	185,769	0.021%
2	10,159,460	1.167%
3	138,848,082	15.943%
4	305,236,992	35.049%
5	356,952,045	40.987%
6	225,092,000	25.846%
7	162,898,537	18.705%
8	98,471,166	11.307%
9	40,003,541	4.593%
10	19,165,574	2.201%
11	10,485,282	1.204%
12	7,597,933	0.872%
13	3,514,400	0.404%
14	2,540,599	0.292%
15	1,952,318	0.224%
16	1,193,997	0.138%
17	1,066,544	0.122%
18	1,044,263	0.120%
19	771,267	0.089%
20	646,401	0.074%

Encoding Overhead

An advantage of fixed-size mappings to generate network names is that all hash values in the `T_MAPPED_XXX` segment have the same size. However, a name can still have an arbitrary number of segments, since there is no such restriction. Depending on the mapping function and whether its output is truncated, network names might be longer or shorter than their application counterparts. Also, `T_MAPPED_XXX` segments an carry extra 4 bytes: 2 for the type and another 2 for the length. Therefore, the total length of a `T_MAPPED_XXX` segment is $(k * s) + 4$, where k is the number of name segments involved in $\mathsf{T}(\cdot)$'s computation and s is the truncated size of $\mathsf{T}(\cdot)$. Whereas, the total length of a standard CCN TLV-encoded name is $\sum_{i=1}^{k}(|N[i]| + 4)$.

Figure 6.8a demonstrates the Unibas dataset average name length (in bytes) and the standard deviation for various truncated output sizes of $\mathsf{T}(\cdot)$. (TLV represents application names encoded in TLV format.) Figure 6.8b shows the average name length increment (or decrement) for various truncated output sizes of $\mathsf{T}(\cdot)$, as compared to the TLV-encoded application names. We note that for $\mathsf{T}_{16}(\cdot)$, proposed network names offer size of reduction of 40%. For $\mathsf{T}_{32}(\cdot)$, network names result in only 2% additional length over application ones. However, for $\mathsf{T}_{160}(\cdot)$, network name sizes increase by a factor of 2.5.

One advantage of network names in content objects is their fixed size. Similar to the discussion above, N_p might be larger or smaller than the original application name, depending on $\mathsf{T}(\cdot)$ output size. We explore encoding overhead of name fingerprints in content headers under three $\mathsf{T}(\cdot)$ output bit-sizes: 256, 384, and 512. We assess network overhead in two ways:

- Average overhead: overhead of using the fingerprint as compared to each name in the Unibas dataset. Results are presented as the average of all the individual overhead values.

Table 6.4: Probability of collision results.

n	$\mathsf{T}_{16}(\cdot)$		$\mathsf{T}_{32}(\cdot)$		$\mathsf{T}_{48}(\cdot)$		$\mathsf{T}_{64}(\cdot)$		$\mathsf{T}_{128}(\cdot)$		$\mathsf{T}_{160}(\cdot)$	
	Exp.	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**
1	0.77	0.77	9.31E-10	9.35E-10	0	2.18E-19	0	5.07E-29	0	1.49E-67	0	8.08E-87
2	1	1	2.80E-6	2.79E-6	0	6.51E-16	0	1.52E-25	0	4.46E-64	0	2.42E-83
3	1	1	5.12E-4	5.12E-4	1.21E-13	1.22E-13	0	2.83E-23	0	8.32E-62	0	4.61E-81
4	1	1	2.41E-3	2.41E-3	7.14E-13	5.88E-13	0	1.37E-22	0	4.02E-61	0	2.18E-80
5	1	1	3.27E-3	3.27E-3	7.57E-13	8.04E-13	0	1.87E-22	0	5.50E-61	0	2.98E-80
6	1	1	1.33E-3	1.32E-3	2.52E-13	3.20E-13	0	7.44E-23	0	2.19E-61	0	1.19E-80
7	1	1	7.01E-4	7.01E-4	1.63E-13	1.67E-13	0	3.90E-23	0	1.15E-61	0	6.21E-81
8	1	1	2.59E-4	2.59E-4	7.82E-14	6.12E-14	0	1.42E-23	0	4.19E-62	0	2.27E-81
9	1	1	4.31E-5	4.31E-5	7.11E-15	1.01E-14	0	2.35E-24	0	6.91E-63	0	3.75E-82
10	1	1	9.93E-6	9.93E-6	0	2.32E-15	0	5.40E-25	0	1.59E-63	0	8.60E-83
11	1	1	2.96E-6	2.98E-6	0	6.94E-16	0	1.62E-25	0	4.75E-64	0	2.57E-83
12	1	1	1.57E-6	1.56E-6	0	3.64E-16	0	8.48E-26	0	2.49E-64	0	1.35E-83
13	1	1	3.34E-7	3.35E-7	0	7.79E-17	0	1.81E-26	0	5.33E-65	0	2.89E-84
14	1	1	1.75E-7	1.75E-7	0	4.07E-17	0	9.48E-27	0	2.79E-65	0	1.51E-84
15	1	1	1.05E-7	1.03E-7	0	2.41E-17	0	5.60E-27	0	1.65E-65	0	8.92E-85
16	1	1	3.73E-8	3.86E-8	0	9.00E-18	0	2.09E-27	0	6.16E-66	0	3.34E-85
17	1	1	3.21E-8	3.08E-8	0	7.18E-18	0	1.67E-27	0	4.91E-66	0	2.66E-85
18	1	1	3.05E-8	2.96E-8	0	6.88E-18	0	1.60E-27	0	4.71E-66	0	2.55E-85
19	1	1	1.70E-8	1.61E-8	0	3.75E-18	0	8.74E-28	0	2.57E-66	0	1.39E-85
20	1	1	1.07E-8	1.13E-8	0	2.64E-18	0	6.14E-28	0	1.80E-66	0	9.78E-86

- Overhead compared to the average name: overhead as compared to the average length of *all* names in the Unibas dataset.

Figure 6.9 shows results of using these methods for computing network overhead for the name fingerprint. It demonstrates that giving more weight to individual names yields higher overhead as compared to the average name length. There is a size increase for all $\mathsf{T}(\cdot)$s since names in the dataset are generally shorter than 256-512 bits. However, since the content namespace has substantial room to grow, this overhead will likely decrease as application names grow and diversify.

Collision Resistance

As mentioned in Section 6.2.2, $\mathsf{T}(\cdot)$ must *at least* be a collision-resistant mapping function and its output can be truncated for practical purposes. However, truncation might affect collision-resistance. To this end, we now look at the collision probabilities in network names for various truncated sizes of $\mathsf{T}_s(\cdot)$, based on prefix length of names in the Unibas dataset. Specifically, we compute the collision probability for at least two *unique* application name prefixes of n segments ($n \in [1, 20]$). Table 5.3 shows the name distribution per number of prefix segments. Over 40% of the names have a unique prefix of 5 segments.

We evaluate the collision probabilities experimentally as well as analytically. To address the latter, we use the Binomial distribution with parameters m and p as in Equation 6.4.

$$f(d) = \binom{m}{d} p^d (1 - p^d)^{m-d} \tag{6.4}$$

$f(d)$ is the probability of a unique name prefix mapped using $\mathsf{T}_s(\cdot)$ occurring at least d times among network name prefixes of length $n \in [1, 20]$, m is the size of this set, and p is the probability of a single element of that set to occur. Assuming $\mathsf{T}_s(\cdot)$ is based on a cryptographic hash function, its truncated output is guaranteed to be pseudo-random. Therefore, $p = 1/2^s$.

The probability of collision that we are trying to determine is $f(d)$ for $d \geq 2$. This is because a collision occurs whenever two or more prefixes map to the same value in the set of mapped name prefixes of length n. We denote this by $f(2^+)$, computed using Equation 6.5.

$$\begin{aligned} f(2^+) = 1 - f(0) - f(1) = 1 - \left(1 - \frac{1}{2^s}\right)^m - \frac{m}{2^s}\left(1 - \frac{1}{2^s}\right)^{m-1} &= \\ 1 - \left(1 - \frac{1}{2^s}\right)^m \left[1 + \frac{m}{2^s}\left(\frac{1}{1 - \frac{1}{2^s}}\right)\right] &= \\ 1 - \left(1 - \frac{1}{2^s}\right)^m \left[1 + \frac{m}{2^s - 1}\right] & \end{aligned} \tag{6.5}$$

Table 6.4 shows $f(2^+)$ using the given values of m in Table 6.3 (second column) and different truncated output sizes $[16, 32, 48, 64, 128, 160]$. 'Exp.' represents collision probabilities obtained from experiments and 'Ana.' represents the same probabilities computed analytically using Equation 6.5. Highlighted numbers indicates anomalies between experiments and analytical results. We compared these values of the collision probability with those computed by mapping and truncating all the names in the Unibas dataset with prefixes of length $n \in [1, 20]$. From this comparison we observe the following:

- Collision probability for $\mathsf{T}_{16}(\cdot)$ is 1 because the number of names with prefix $n \in [2, 20]$ is over 2^{16}, which is the output space size of $\mathsf{T}_{16}(\cdot)$.
- Experimental collision probability for $\mathsf{T}_{64}(\cdot)$, $\mathsf{T}_{128}(\cdot)$, and $\mathsf{T}_{160}(\cdot)$ and some n values of $\mathsf{T}_{48}(\cdot)$ is 0. This is because the number of names with prefixes n in Table 5.3 is small compared to the output space size of the previous mapping functions. Therefore, an experimental collision cannot occur.
- Collision probability for $\mathsf{T}_{32}(\cdot)$ is nearly equal to the distribution in Table 5.3. Specifically, it reaches its maximum for $n = 5$, which is the most common prefix length among the names in the Unibas dataset. Figure 6.10 depicts this comparison.

6.2.5 Performance Assessment

In this section, we provide a comprehensive assessment of network name performance in a variety of FIB implementations. We experiment with FIB implementations that would benefit from network names, including those based on hash tables and bloom filters. We refer to these types of FIBs as *hash-based* FIBs. *Trie-based* FIBs are only useful for names whose segments, and the character tokens within each segment, can be ordered. Our network naming scheme removes this property since it replaces otherwise sortable segments with (pseudo)random hash digests. Thus, network names prohibit trie-based FIBs. Before describing our experimental results, we first describe each FIB algorithm and data structure used in our study.

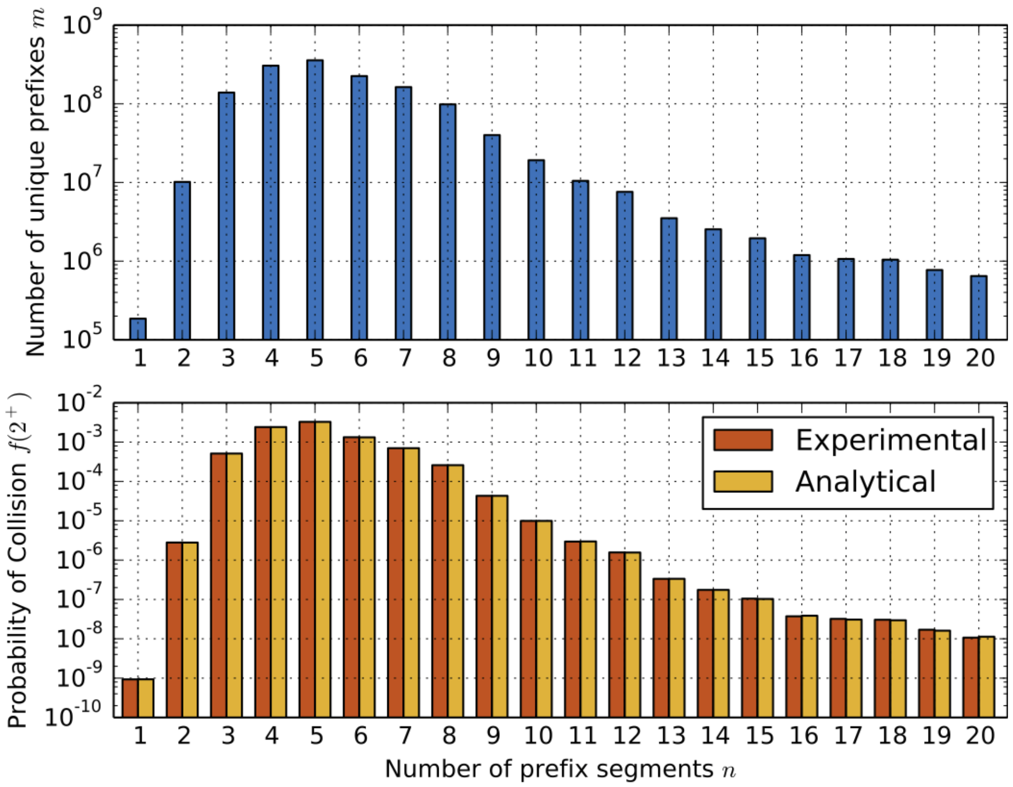


Figure 6.10: Experimental and analytical collision probabilities for $\mathsf{T}_{32}(\cdot)$

FIB Algorithms

In the following subsections, we present designs of various hash- and trie-based FIBs.

Naive FIB The naive FIB is one that performs LPM by combining a reversed sequential scan of a name and standard hash table lookup. Specifically, for each prefix, it computes the hash digest of that prefix and uses the result to index into a hash table. Each hash table entry stores a list of prefixes (or their digests) and the interface vector to which each prefix should be forwarded. If a prefix match is found in a hash table list, that prefix is returned. Otherwise, FIB lookup is repeated using a prefix of one less segment. This process continues until (a) a match is found or (b) every segment is checked and no matches are found. In case (b), the FIB typically returns an error and the forwarder is expected to drop the packet or generate a NACK.

Aside from memory lookup, this runtime is dominated by prefix hash computations. Thus, this algorithm could utilize network names by removing the hash computation entirely: the i-th hash prefix of N is simply the i-th segment of $\bar{N}$.

Cisco FIB The Cisco FIB [289] is a modification of the naive FIB tuned for high-speed forwarding. Specifically, it includes the following variations:

- SipHash [46] is used as the hash function. This was motivated by its low cycle/byte cost and high collision resistance properties. (It is not, however, a cryptographic hash function. It is only a pseudorandom function when appropriately keyed.)
- The hash table implementation is made compact to exploit memory caches. Specifically, hash table entries are compacted such that multiple can fit on a single cache line. This is done to improve memory access latency and remove worst-case memory lookup stalls.
- LPM artificially starts with the M-th segment, rather than the last segment, to avoid a complete name iteration. If the FIB indicates that a prefix containing $d > M$ segments is present, LPM restarts at the last segment as in the normal LPM case. Otherwise, LPM resumes iterating backwards from the $(M-1)$st segment. Here, M is a tunable parameter.

The Cisco FIB benefits from network names since unnecessary hash computations are removed.

Caesar FIB The Caesar FIB [312] combines hash tables and BFs to improve lookup speed. It introduces Prefix Bloom Filters (PBFs) which take as input a name and return the longest number of matching segments in the FIB, or 0 otherwise. Lookup works by first querying the PBF to determine the number of prefixes that match in the FIB. THe hash of this prefix is computed and then used to index into a hash table storing link identifiers.

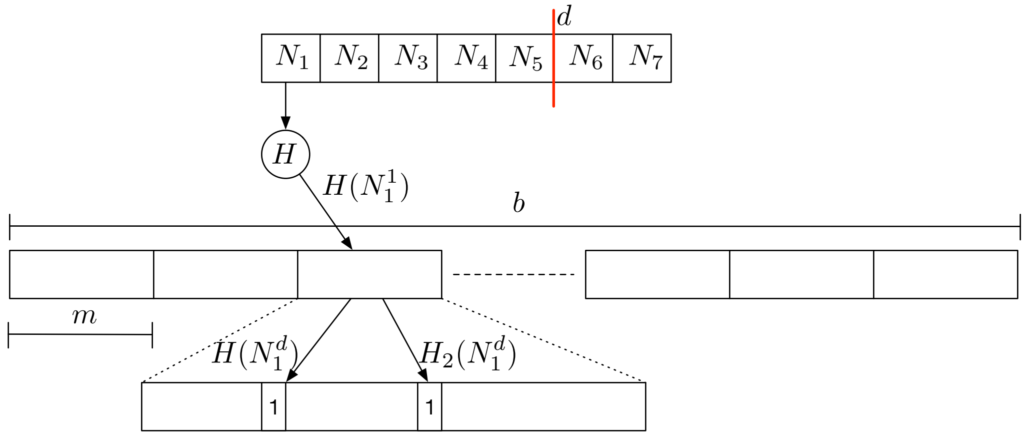


Figure 6.11: Caesar FIB Prefix Bloom Filter

A PBF data structure, shown in Figure 6.11, is a collection of b BFs of size m bits each. Each internal BF has the same number and type of k hash functions. Lookup in a PBF pbf works as follows. Given a name N, pbf computes $H(N_1^1)$ to determine which of the b BFs to inspect.[20] Once a target BF filter is identified, pbf queries it using a standard LPM algorithm. Specifically, for index $i = |N|$ to 1, pbf queries filter using $h_1(N_1^i), \ldots, h_k(N_1^i)$. If this results in a match, i.e., if all of the corresponding bits in the BF are set, i is returned as the result. If no match is found, 0 is returned.

One significant performance improvement offered by this variant is the use of linear hashing to avoid redundant and unnecessary BF hash computations. Specifically, given a name with d segments and PBF with k hash functions, a straightforward implementation would require $kd(d-1)/2$ hash computations. To avoid this $\mathcal{O}(k \times d)$ complexity, one can compute each hash as follows:

$$H_i(N_1^j) = \left\{ \begin{array}{ll} h_i(N_1^j) & \text{if } i = 1 \text{ or } j = 1 \\ H_i(N_1^1) \oplus H_1(N_1^j) & \text{otherwise} \end{array} \right\}$$

[20]This filtration step serves to group names with identical prefixes into the same BF. This is done based on the assumption that the first segment is a significant-enough differentiator to evenly distribute the load across the b BFs.

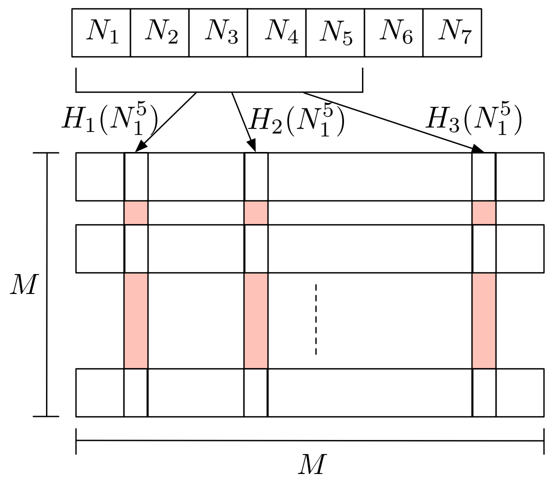


Figure 6.12: Merged BF FIB

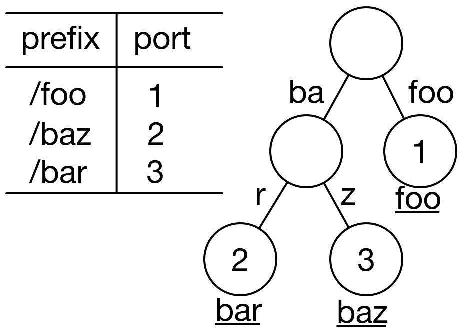
One variant of the Caesar FIB replaces the post-LPM hash table lookup with a series of BF queries. Specifically, instead of indexing a hash table to find the output interface vector, one can query a BF associated with each FIB interface. Each interface BF that returns a match is added to the output interface vector. This alternative is more appropriate for hardware implementations with a fixed number of output interfaces, since BF queries can be done in parallel.

Merged BF FIB The Merged BF FIB from Wang et al. [324] builds on the per-interface BF variant of the Caesar FIB by merging the PBF with the per-interface BF lookup. Specifically, it replaces each individual BF in the PBF with a set of N BFs, one for each output port. Lookup begins by querying all N BFs in the PBF block identified by $H(N_1^1)$. If there is a match in filter F_i, then interface i is added to the output interface vector. If there is no match among the N ports, an empty output vector is returned. This is shown in Figure 6.12.

Binary Patricia Trie FIB A Patricia trie [224] is a space-optimized trie. Each single-child node is merged with its parent to remove individual branches. Branches between parent and child nodes use labels, e.g., a character string. A prefix in a trie is composed of its root-to-leaf path labels. By design, prefix lookup has complexity linear in the length of the inputs. A binary Patricia trie is one in which branches are made based on single bit differences in prefixes. These are well-known and extensively studied data structures used, among other things, for IP address lookup in some Linux kernels [285].

Song et al. [295] presented a FIB based on binary Patricia tries, as shown in Figure 6.13. (The underlined names are the *tokens* that were removed in a speculative FIB variant – described below.) They chose this data structure because it exploits shared segments across different prefixes, is agnostic to name form and representation (since they are treated as opaque, sortable, binary strings), and does not require input name parsing before prefix lookup. Due to the possibly large amount of space needed for these tries, Song et al. implemented a "speculative" trie variant in which trie tokens are removed. (Tokens are actual labels stored alongside trie nodes.) Forwarding decisions based on this reduced trie change from that of LPM to "longest-prefix-classification" (LPC). (Without the tokens, one cannot tell if the prefix matches that of the corresponding leaf node in the trie.) In practice, this means interests without a matching FIB entry are *still* forwarded. The authors acknowledge that problems such as forwarding loops may occur with speculative forwarding, and defer their resolution to other means such as TTL exhaustion. To partly deal with this problem, their final design is based on a so-called *dual Patricia* trie, which is a data structure where, internally, there is a standard binary Patricia trie with tokens for LPM and a speculative binary Patricia trie for LPC. Lookup involves two steps: (1) LPM with the tokenized trie and then, if that fails, (2) LPC with the speculative trie. Since the former dominates in performance, we do not experiment with speculative tries.

Tree-Bitmap and BF FIB The Tree-Bitmap and BF FIB of Quan et al. [260], called TB^{2}F, is designed based on the observation that parts of CCN names are better suited for



prefix	port
/foo	1
/baz	2
/bar	3

Figure 6.13: Patricia trie FIB

tries than hash tables (and vice versa). Accordingly, the TB^{2}F design consists of both a trie and set of (counting) BFs. Trie leaves point to a single BF. Inserting a name into the FIB requires inserting its first T segments into the trie and then placing the remaining suffix, called B segment, into the single BF associated with the corresponding trie leaf. This separation permits the trie and BF to be partially processed in parallel. Specifically, B suffix segment hashes can be computed while the trie is index.

There are two cases to consider for lookup. First, if a match with shorter than T segments is found in the trie, the corresponding interface vector is returned and no further processing is done. Otherwise, a pointer to a BF is returned. This BF is then indexed to return an interface vector and name sub-prefix. (The full prefix is not returned since the BF only stores B segments.) This sub-prefix is then combined with the first T segments of the name to index into a hash table, yielding an output interface. Optimally, name lookup requires only a trie traversal. Worst case lookup requires a trie traversal, BF lookup, and hash table index. This complete flow is shown in Figure 6.14.

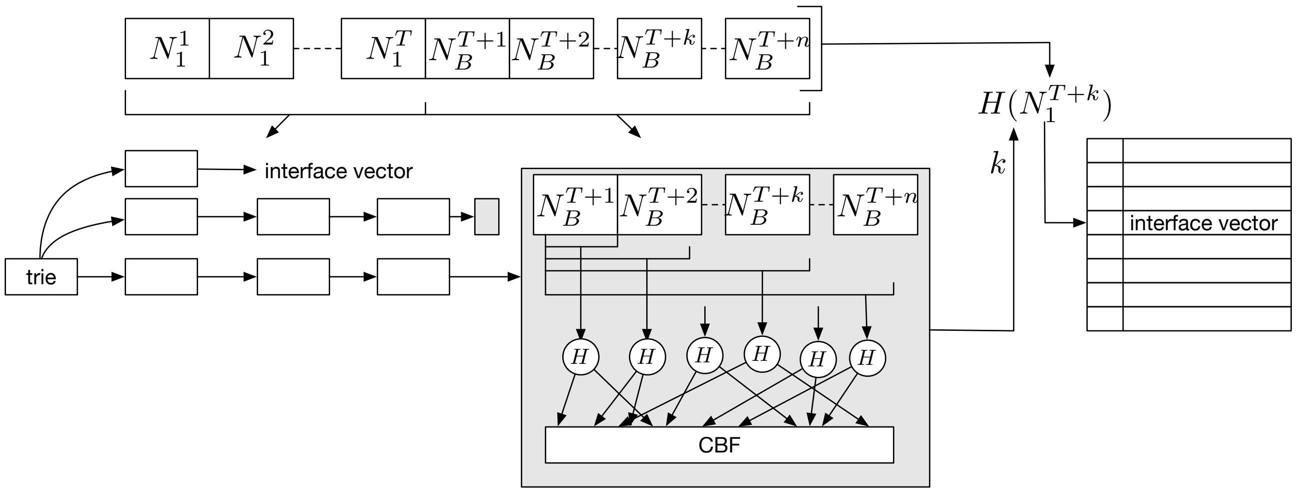


Figure 6.14: TB^{2}F FIB lookup

Experimental Methodology and Results

We implemented each FIB algorithm in software to test each with network names. The code for which is publicly available at [7]. We tested name insertion and lookup times for each FIB. Names used to populate FIBs were drawn from the URL blacklist available at [15]. (This was chosen as the FIB *load* data to make our results compatible with the experimental results conducted in [289].) To emulate real traffic loads, we extended the names in the URL blacklist dataset according to statistics in the Unibas dataset. Specifically, we extended the number of segments in each name so that the distribution of name segment counts matched that of the Unibas dataset. Moreover, the length of each new added segment was drawn from the distribution of segment lengths in the Unibas dataset.

After loading FIBs, we profiled insertion and lookup times by varying: (1) FIB algorithm, (2) network name hash digest lengths, (3) number of BF hash functions used in BF-based FIBs, (4) hash tables load factors, and (5) total FIB load (before lookup). We ran our experiments on a desktop computer with an Intel Core i7-3820 CPU with 3.6GHz cores and 24GB of DDR3 RAM running Ubuntu 14.04 LTS. The following subset of our results fix the number of BF hash functions to 8, hash table load factor to 42, and total pre-lookup load to 650.

Figure 6.15 shows the percentage improvement using network names for hash-based FIBs. Removing hash computations on unbounded name segments improves efficiency by approximately 80%. Figure 6.16 compares the lookup times of a Patricia FIB to Naive and Cisco hash-based FIBs, with and without the network names. We observe that lookup times for Cisco FIBs with network names are much less than Patricia tries. Figures 6.17 and 6.18 show a similar comparison for Caesar, Caesar-Filter, and Merged-Filter FIBs, except that we compare against Patricia and TB^{2}F FIBs. (The parameters of each FIB are shown in the figures.) In all cases, we see that software-based BFs worsen the performance. However, with network names, Caesar FIBs can surpass both Patricia and TB^{2}F FIBs.

Given that these results are highly dependent on the platform, name dimensions, and FIB parameters, it is difficult to affirmatively say which FIB is superior. However, what is clear is that network names offer performance improvements to hash-based FIBs FIBs when memory allocations are minimized. We leave assessment of hardware implementations to future work.

6.2.6 Security Considerations

From a security perspective, network names prompt several issues. Perhaps the most important is related to content signatures. Traditionally, signatures include the (application) name and payload of a content object. However, since packets no longer carry application names, the question of how one might verify content signatures in the network arises. Notice, however, that the name fingerprint N_p is included in interests and content objects. Cryptographically, there is no difference between hashing over the application name and payload vs hashing over the name fingerprint and payload. Thus, if signatures are generated using N_p instead of the application name, then routers can still perform in-network verification.

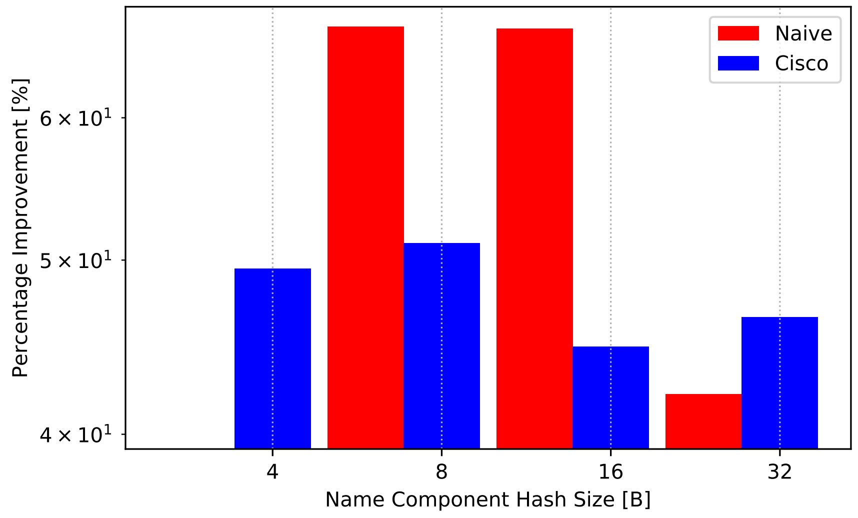
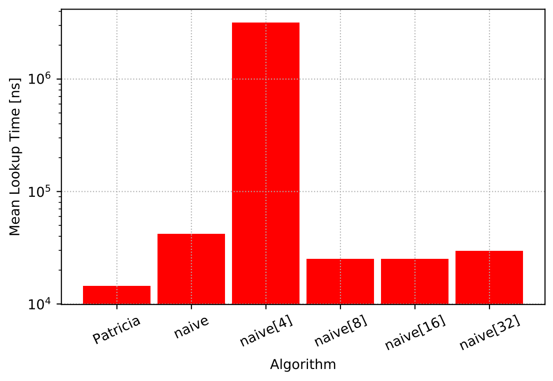
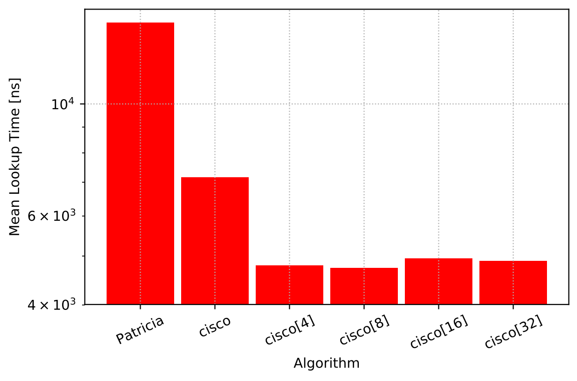


Figure 6.15: Network name percentage improvement hash-based FIBs (Naive and Cisco)



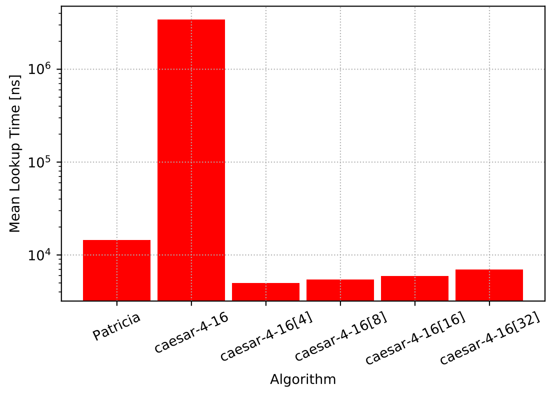
(a) Patricia vs Naive hash-based FIB



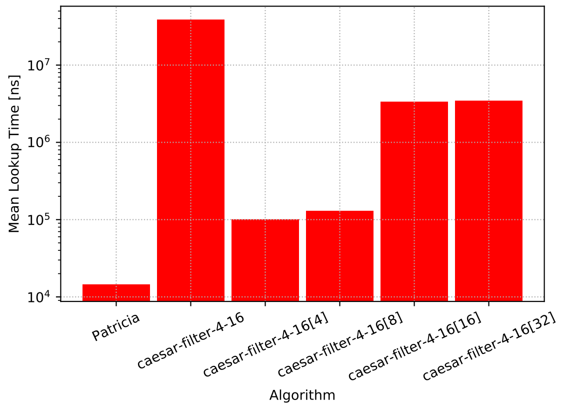
(b) Patricia vs Cisco hash-based FIB

Figure 6.16: Patricia trie FIB vs Naive and Cisco hash-based FIBs

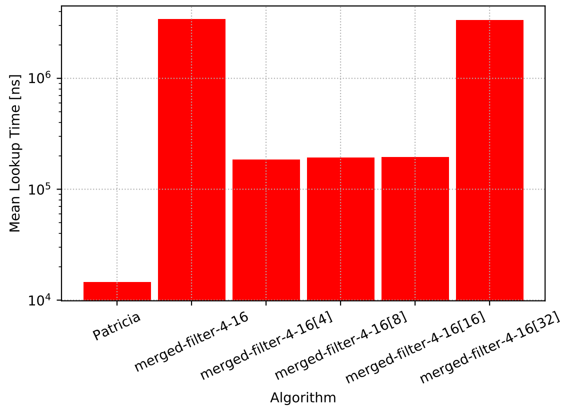
Second, since network names are computed with (possibly truncated) obfuscation functions, name prefix collision in a FIB would cause an interest to be forwarded on an incorrect interface. We believe that this is not an issue in practice because a FIB is populated by a routing protocol. Thus, the routing protocol (which is exposed to application names)



(a) Patricia vs Caesar-4-128 FIB



(b) Patricia vs Caesar-Filter-4-128 FIB

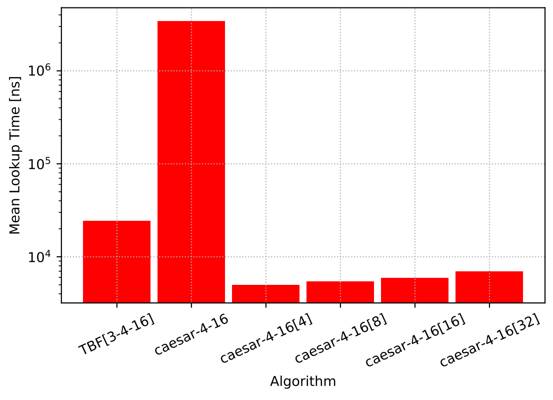


(c) Patricia vs Merged-Filter-4-128 FIB

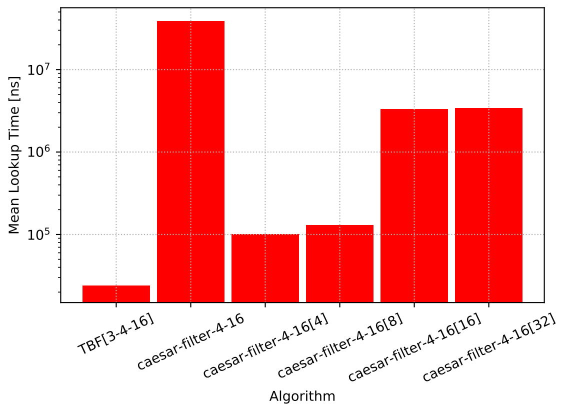
Figure 6.17: Patricia trie FIB vs Caesar-4-128, Caesar-Filter-4-128, and Merged-Filter-4-128 FIBs

can explicitly prevent colliding names from being inserted into the FIB. Moreover, with cryptographic hashes, collision probability is negligible, as supported by experiments and analysis.

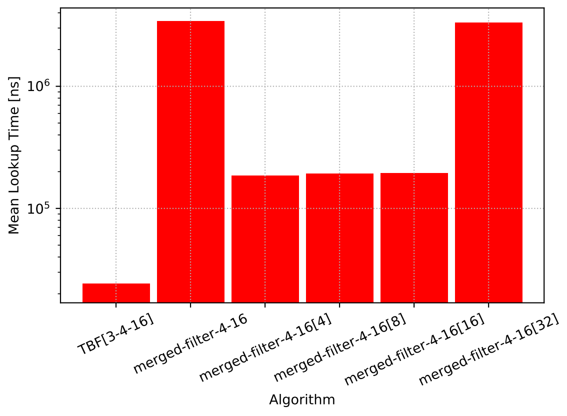
The third issue is that an adversary might provide fake network names that would cause interests to be forwarded towards a given router or producer as a form of denial-of-service (DoS). For example, if an adversary knows that a network name with the first segment equals hash digest x, then it can generate and issue fake interests with that prefix even if it does



(a) TBF-3-4-128 vs Caesar-4-128 FIB



(b) TBF-3-4-128 vs Caesar-Filter-4-128 FIB



(c) TBF-3-4-128 vs Merged-Filter-4-128 FIB

Figure 6.18: TBF-3-4-128 FIB vs Caesar-4-128, Caesar-Filter-4-128, and Merged-Filter-4-128 FIBs

not know the corresponding application name segment. However, this is no different from currently possible interest flooding attacks [24, 85, 315]. Therefore, any solution to standard CCN interest flooding attacks would also prevent this type of DoS.

A more relevant and harmful DoS attack in the current CCN architecture is where an adversary generates interests for names that collide in the (hash-table-based) PIT and induce expensive collision-resolution steps [289]. This attack is not applicable with network names, since N_p is pre-computed using a collision-resistant hash function. That is, if an adversary sends multiple interests with the same network name, then the probability that they correspond to different application names is negligibly small.

Revisiting Data Plane Attacks

In this section we revisit the data plane attacks of Section 6.2.1. We show how network names obviate PIT and CS attacks while simultaneously increasing FIB attack difficulty.

PIT and CS Attacks Computational DoS attacks on the PIT and CS are effectively eliminated with network names. Since it is required that N_p is the *first* segment of a network name $\bar{N}$, this value can be quickly parsed and used to index either data structure in constant time.

FIB Attacks As discussed in Section 6.2.1, the core problem is that data names carry application-layer meaning, and the network cannot (or should not) restrict this for performance or traffic management reasons. Thus, there is no CCN-friendly name format that could mitigate FIB attacks. However, as shown in the previous section, network names can significantly reduce router work when performing FIB lookups.

6.3 Network-Layer Integrity Checks

Content signature verification is optional in CCN. Thus, some routers may blindly forward (possibly invalid) content. This permits on-path attackers that hijack prefixes or modify content in transit to go undetected on some network paths. In the worst case, consumers may be first to learn that content is fake or poisoned, and they will have no information about which upstream router(s) is (are) malicious.

The only way around an on-path attacker is to bypass it by adjusting routers' forwarding information. However, without knowledge of attack origin or network topology, this adjustment can be done very far away from the actual adversary, e.g., at consumers. To aid routers in making informed forwarding adjustments, DiBenedetto et al. [106] proposed an adaptive forwarding strategy based on consumer-provided content "complaints." Specifically, consumers that detect poisoned content inform upstream routers about said content and provide them with information needed to check its validity. A router verifies such downstream complaints and adjusts its forwarding strategy to avoid paths that lead to corresponding producers. Complaints are recursively forwarded upstream so that every router can adjust its forwarding table. Despite its simplicity, this approach has several undesirable features:

- Complaint messages are a form of computational DoS on routers, due to cryptographic operations needed to validate them. If complaint messages are unprotected, another type of DoS would occur since attackers can generate spurious complaints.
- Complaint messages can be very large, since they encompass fake content and all information needed to verify its authenticity.
- Routers are forced to update forwarder routes even though the immediate upstream routers which provided content may not be malicious. This can lead to suboptimal route selection that harms downstream routers. For example, routers could unnecessarily choose a more expensive link.

More importantly, DiBenedetto et al. propose an application-layer remedy to a network-layer problem. Certainly, on-path attackers *are not unique to ICN*. They may exist in IP networks as transit routers which modify packet payloads in motion, or drop packets altogether. Thus, on-path attackers must be dealt with at the network layer. In other words, while content authenticity is strictly an end-to-end issue, content retrieval, i.e., avoidance of on-path attackers, is a network-layer concern.

In this section we reconcile content authenticity and retrieval with an efficient per-hop content integrity checking mechanism. The main idea is to allow routers to immediately detect when (at least) one upstream peer has modified a content object. Intended contributions are:

1. A novel "adversary leap frog" scheme that uses cryptographic MACs to ascertain authenticity of upstream content packets.
2. A merge of IKB with the above scheme to enable in-network content authenticity and secure content retrieval.
3. A security analysis and experimental assessment of the proposed scheme to show its utility and practicality.

6.3.1 Content Poisoning and Namespace Arbitration

Content poisoning is an attack on *content retrieval* that prevents consumers from obtaining valid (authentic) data. This attack comes in two flavors: proactive and reactive, as defined by Gasti et al. [137]. In the remainder of this section, we focus on the latter and use the term *content poisoning* strictly in that context.

Reactive content poisoning attacks occur when an on-path $\mathcal{A}$ injects fake content objects into the network. Consumers and routers can identify fake content by verifying content signatures. However, not all routers verify content signatures since it can be too computationally expensive. Thus, content injected by $\mathcal{A}$ percolates through the network until some entity verifies it. When a router or a consumer detects fake content, it only learns that *some upstream router or a producer tampered with that content.* The verifying entity has no insight into *where* the attack occurred.

Consider a path of length n with intermediate routers R_{i-1}, R_i, and R_{i+1}, between a consumer Cr and producer P shown in Figure 6.19. Downstream (left-facing) arrows indicate content

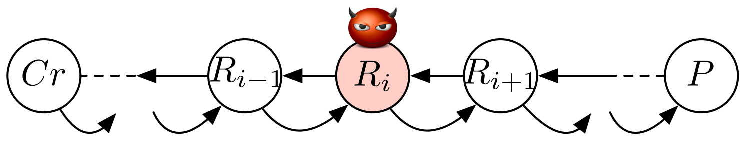


Figure 6.19: Content poisoning attack

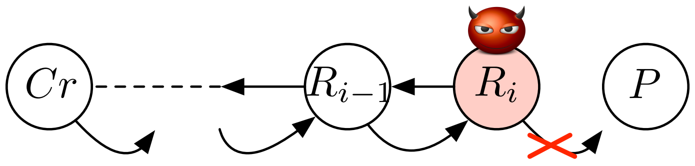


Figure 6.20: Content generation attack

flow from P to Cr. $\mathcal{A}$ is an on-path attacker that can attack in (at least) two ways: (1) by modifying content packets in transit from P, or (2) by responding with fake content. We refer to these attacks, respectively, as *on-path modification* and *on-path generation* attacks. They are illustrated in Figures 6.19 and 6.20, respectfully. Upon authenticating content, any router R_j, $j \in [1, i-1]$, only learns that either some router $R_k, k \in [j+1, n]$ or P provided fake content. We claim that detecting on-path attacks by an intermediate router is equivalent to checking content integrity and authenticity as content flows through the network.

To obviate the need for signature verification in the fast path, DiBenedetto et al. [106] proposed a reactive technique to detect and bypass on-path attacks. Their proposal is premised on the argument that undetected modification and generation attacks result from producers publishing under unauthorized namespaces. This is addressed by requiring consumers to report fake content to routers that verify these reports and forward them upstream, if necessary. Routers then greedily find an alternate path to the valid content, by either: choosing a different forwarding interface for the content, or probing all interfaces until valid content is returned. Although this mitigation strategy works in theory, it has certain drawbacks discussed in Section 6.3.

Moreover, this approach is based on the dubious assumption that namespace ownership is proven by content published under that namespace. This means that any application can publish under any namespace provided that it possess the public key that a consumer trusts. However, there is nothing preventing multiple applications from claiming ownership of identical namespaces. To see why this is problematic, consider two applications A_1 and A_2 that both claim ownership over namespace $N =$`/a/b`, i.e., both advertise themselves as producers under N. Moreover, neither one is malicious. Since each application has its own key pair, data produced by A_1 and A_2 are signed and then verified with *different* public keys. Now assume consumer Cr wishes to obtain content `/a/b/x` from A_1 using KeyId K_1. Cr issues an interest and provides K_1's ID. However, if A_2 is closer to Cr than A_1, A_2 would produce content signed by K_2, not K_1. Therefore, all on-path routers as well as Cr would deem the response invalid. However, whether it is invalid depends on who has proper ownership over N. Clearly, this problem occurs since the network allows both applications to advertise under N without verifying ownership.

Without a namespace arbitration mechanism, this problem is unavoidable. Thus, we conclude that a node can not unilaterally advertise under *any* namespace it wants. There must exist some trusted authority that manages namespace ownership. This authority can help prove ownership of a namespace. This way, namespace ownership can be ascertained before routes are injected into the network. As a result, any after-the-fact content modification or generation attack would stem from failure to check packet authenticity and integrity in the data plane. Addressing this problem would thus mitigate on-path attackers.

6.3.2 Threat Model

We now outline our system threat model. We assume a network composed of routers, producers, and consumers. Routers have identities (for network management purposes) and can learn identities of nearby peers in the network. Also, routers may be organized into autonomous systems (ASes). Routers are added to the network in a controlled and secure fashion.

Each namespace is owned by an application (or identity) and is associated with a unique public-private key pair. It is possible for multiple producers to serve the same namespace. In this case, they would all belong to the same application. Lastly, there exists a globally trusted authority responsible for managing namespace ownership. It may be implemented as a centralized arbiter similar to IANA, a decentralized system such as DNS, or a distributed system such as Namecoin [175]. For simplicity, we assume that the authority is a key-value store which maps namespaces to the corresponding public key or an empty string (if no application owns the namespace). Moreover, we assume that the authority is queried before producers are allowed to inject namespace prefixes into the routing protocol.

We assume a general on-path adversary $\mathcal{A}$ that can compromise a polynomial number of routers (up to k) and any producer. $\mathcal{A}$ has complete control over each compromised entity. In particular, it controls the content and timing of messages generated and processed by such entities. $\mathcal{A}$ can also spawn authenticated nodes under its control on demand. This capability can be used to modify packets or generate fake content. Security against $\mathcal{A}$ means that it is infeasible for $\mathcal{A}$ to perform these actions without detection. Once $\mathcal{A}$ is detected, honest entities can take action to bypass it.

6.3.3 Integrity Zones

As mentioned above, on-path attacks are possible because there is no efficient means for routers to verify packet integrity and origin authenticity in the fast path. Meanwhile, application-driven solutions (see, e.g., [106]) have some prominent drawbacks. We believe this problem should be addressed at the network-layer. With respect to integrity, hop-by-hop packet mechanisms are insufficient since $\mathcal{A}$ can compromise individual routers. At the same time, end-to-end integrity mechanisms are infeasible since "ends" are undefined: one end could be a producer or an intermediate router's cache. Ideally, we need something between these two extremes.

For origin authenticity, digital signatures are insufficient for identifying the origin of fake content. A router should be able to learn if a packet was generated by a producer who owns the corresponding namespace. This is *not* the same as verifying that the packet is authentic, since the verifying public key need not be the same as the namespace ownership key.

Our approach relies on *integrity zones*, which work as follows: Every router shares (and rotates) a unique key with every router that is $k \geq 2$ hops away. These keys are used to generate and verify packet MACs to check packet integrity in transit. Upon receipt of a content packet, a router verifies at most k MAC tags and, if all are valid, replaces them with k new MAC tags generated locally. Upon receipt of an interest packet, each router R appends its ID to the packet. If the number of downstream IDs exceeds k, R also drops the oldest. In effect, MAC tags and router IDs form a sliding window of size k that moves as packets propagate between producers and consumers. If $\mathcal{A}$ compromises at least k contiguous routers, the scheme fails.

Integrity zones require distance knowledge to content producers, which can be provided by a secure distance-vector routing protocol, such as DCR [135]. This is needed so that routers can check if content was actually generated by an appropriate producer. When generating content in response to an interest, which carries identities of the k previous hops through which the packet flows, each producer computes and includes k MACs using keys associated with these k routers. Downstream routers then validate producer- or router-generated MACs before including their own, as described above. Also, the i-th downstream router, where $i \leq k$, *must* check that there are at least i valid MACs in the content packet. This ensures that original MACs were not generated by an on-path $\mathcal{A}$ that hijacked a namespace.

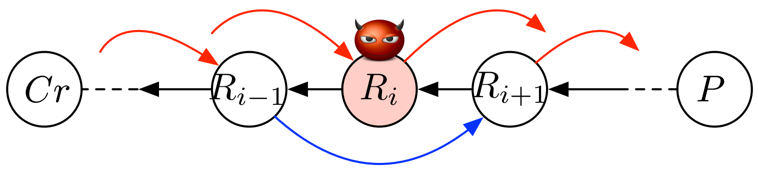


Figure 6.21: $\mathcal{A}$-controlled intermediate router ($k = 2$)

Integrity zones allow routers to play "adversary leap frog." In other words, when less than k contiguous nodes are compromised, a router can always learn exactly which router is malicious and can take steps to remedy the situation in the forwarding plane. Inspired by [155], the crux of our proposal is for routers to verify each content packet as it is routed through the network. Consider the following example, where:

$$\mathsf{path} = R_1, R_2, R_3, \ldots, R_{n-2}, R_{n-1}, R_n$$

is a path between some Cr and P. Moreover, assume that Cr previously issued an interest for P that resulted in a content C being forwarded along the reverse of path. Now, suppose that $\mathcal{A}$ controls $R_i \in \mathsf{path}$, and modifies either the content packet or its signature. This means that R_{i+1} sees a valid content packet while R_{i-1} does not. The goal is for R_{i-1} to learn that R_i is misbehaving. Ideally, R_{i+1} computes a content MAC using a key shared with R_{i-1}. Then, R_{i-1} fails to verify this MAC and learns that R_i is malicious. This flow is shown in Figure 6.21. Blue arrow shows dependency between R_{i-1} that detects a problem and R_{i+1} that does not. The network can then attempt to avoid and bypass R_i for *all interests* and heal itself without requiring any consumer involvement.

To summarize, within a given integrity zone, each router is responsible for: (a) appending its identifier to each interest, (b) verifying and removing k MACs of each content packet, and (c) computing k MACs and appending them to each content packet. Thus, every router performs $2k$ MAC operations for each interest and content pair. (See Section 6.3.5.) Below, we discuss this approach in detail, including: (1) how routers obtain shared keys, (2) why producer (anchor) distances are required, (3) packet formats and processing logic, and (4) network recovery steps.

Leap-Frog Key Distribution

Each router shares a key with every router $k \geq 2$ hops away. Depending on the network ecosystem, these keys may be pre-shared between routers, i.e., within a single AS. Otherwise, keys must be established using a key exchange protocol. One potential obstacle is that such protocols require some form of a PKI to ascertain each peer's identity. However, this is not an insurmountable challenge, since most networks are managed and contain routers controlled by the same administration. We therefore defer this bootstrapping phase to future work.

Anchor Distances

Integrity zones allow a router to prove that its k upstream routers did not modify a content packet. However, if $\mathcal{A}$ hijacks a namespace there could be less than k MACs in a content packet. This is only possible if the consumer-to-producer path has $j < k$ hops. The packet would then carry j MACs and each router on the path would know this distance and expect the appropriate number of MACs.

There is one more edge-case: Let R be a benign router $j \geq 2$ hops away from P. Suppose that $\mathcal{A}$ is adjacent to R on the $R \rightarrow P$ path, and assume that $\mathcal{A}$ attempts a data generation attack. In doing so, it must provide k valid MACs verifiable by k downstream routers, including R. Upon receipt of the fake content packet, R can only verify *one* MAC – the one generated by $\mathcal{A}$. If R does not know that P is after R on the path, R may incorrectly conclude that the content packet is valid and forward it. However, if R knows that P is j hops away, it knows to verify j MACs. R can therefore immediately learn that an attack took place. Thus, security of this scheme relies on accurate anchor distances. We assume the existence of a secure protocol that allows routers to learn the minimum distance to a namespace anchor. This should be feasible assuming a global namespace ownership authority discussed earlier. However, we defer exploration of this topic to future work.

Packet Formats and Processing

Each content packet carries MACs needed to check validity of previous k hops (see Figure 6.22). Also, each packet carries MACs necessary for the i-th *downstream* router to verify previous $k-i$ hops. Thus, a content packet carries no more than $k(k+1)/2$ MACs at a time. MACs generated by each router are stored in a list. Also, each MAC is associated with a symmetric key shared between two routers. For example, a MAC generated by a key shared by R_i and R_j with IDs i and j is $\sigma_{i,j}$. Each MAC has an associated ID of the router that generated it, denoted $\sigma_{i,j}.ID$. A router stores its shared keys in a table called KeyTable. A key is fetched using an identifier, e.g., $k_{i,j} = \mathsf{KeyTable}[\sigma_{i,j}.ID]$. In S-expression notation, the set of MAC lists stored in a packet header is represented as follows:

```
(T_INTEGRITY_ZONE
                    (T_PACKET_MACS   [$\sigma_{i,i+1}$]
                    (T_PACKET_MACS   [$\sigma_{i,i+1}, \sigma_{i,i+2}$])
                    ...
                    (T_PACKET_MACS   [$\sigma_{i,i+1}, \sigma_{i,i+2}, \ldots, \sigma_{i,i+k}$])
                   )
```

When a router processes a content packet it deletes at most k MACs since they are useless to downstream routers. This procedure is detailed in Algorithm 26. Processing an interest is much easier: a router simply injects its own identifier in the interest header and forwards it, if necessary. This is detailed in Algorithm 25.

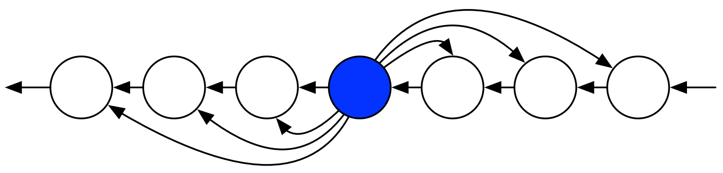


Figure 6.22: Router MAC dependencies

Algorithm 25 Interest processing

```
Require: $k$
 1: Input: $I[N, \mathsf{IDs}]$, $F_{in}$, $R_{\mathsf{ID}}$
 2: if $N$ in the PIT then
 3:     Add $F_{in}$ and $\mathsf{IDs}$ to the PIT entry with name $N$
 4: else
 5:     Append $R_{\mathsf{ID}}$ to $\mathsf{IDs}$ to form $\mathsf{IDs}'$
 6:     Lookup $F_{out}$ in the FIB using $N$
 7:     Forward $I[N, \mathsf{IDs}']$ to $F_{out}$
```

Algorithm 26 Content processing

```
Require: $k$
 1: Input: $C[\mathsf{MACs}, D, V]$,
 2: Output: $C[\mathsf{MACs}', D, V]$, $F_{out}$
 3: for $l_i$ in $\mathsf{MACs}$ do
 4:     $\sigma := l_i.\mathsf{pop}()$
 5:     $k_{ID} := \mathsf{KeyTable}[\sigma.ID]$
 6:     if $\sigma \neq MAC(k_{ID}, D||V)$ then
 7:         Drop $C$ and flag upstream router as malicious
 8: Drop first MAC for each $l_i$ in $\mathsf{MACs}$
 9: $Entry := PIT.\mathsf{Lookup}(C)$
10: if $|\mathsf{MACs}| < Entry.d\}$ then
11:     Drop $C$ and flag upstream router as malicious
12: $l_R = []$
13: for $ID$ in $Entry.IDs$ do
14:     $k_{ID} := \mathsf{KeyTable}[ID]$
15:     $l_R.\mathsf{Append}(MAC(k_{ID}, D||V)$
16: $C' := C.\mathsf{MACs}.\mathsf{Append}(l_R)$
17: return $C'$, $Entry.F_{out}$
```

Currently, all MACs must fit in the per-hop header of the content packet. However, the current CCN packet format restricts this header to a maximum size of 256 bytes. Assuming a MAC of 128 bits, or 16 bytes, each packet can support a radius of at most $k = 4$.[21] This a

[21] This limit assumes there are no other per-hop headers, such as the interest lifetime or recommended cache time. In practice, these headers are usually present in packets, meaning that $k < 4$. However, given

reasonable restriction, since it requires $\mathcal{A}$ to compromise $k \geq 4$ contiguous routers to subvert integrity zones.

MAC Compression

The packet format described in the previous section requires all MACs to be listed separately in each content packet. The resulting overhead is $\mathcal{O}(k^2)$ MACs per packet. To lower it, MACs can be compressed. Consider the verification process at each hop. Each router verifies k MACs individually and detects an attack if at most one fails. These k MACs could be aggregated into a single MAC. Each router could then verify an aggregate. With this format, the packet header would only need to carry a list of k aggregate MACs and list of k upstream router IDs, as shown below:

```
(T_INTEGRITY_ZONE
                (T_ROUTER_IDS  [ID_{i+1}, ID_{i+2}, ..., ID_{i+k}]
                (T_PACKET_MACS  [σ_{i+1}, σ_{i+2}, ..., σ_{i+k}])
               )
```

With this format, each packet is processed as shown in Algorithm 27. This variant also modifies per-hop headers in place, which is far more efficient.

Recovery

When a R_i detects that *only one* of its upstream neighbors $R_j, j > i$ tampered with a content packet, it concludes that R_{i+1} is also malicious. This is because, if R_{i+1} were not malicious, it would have detected the attack by R_j, $j > i + 1$, and dropped the packet accordingly. Therefore, R_i can remove all FIB entries that point to R_{i+1}. It does not need to probe the

that the packet format is flexible, we could easily extend the per-hop header capacity to accommodate these new fields.

Algorithm 27 Content with compressed MAC processing

```
Require: $k$
 1: Input: $C[\mathsf{IDs}, \mathsf{MACs}, D, V]$,
 2: Output: $C[\mathsf{IDs}', \mathsf{MACs}', D, V]$, $F_{out}$
 3: $\sigma = 0^{\lambda}$                                   ▷ Empty string to start
 4: for $i := 1, \ldots, k$ do
 5:     $k_{ID} := \mathsf{KeyTable}[\mathsf{IDs}[i]]$
 6:     $t = MAC(k_{ID}, D||V)$
 7:     $\sigma = \sigma \oplus t$                           ▷ Aggregate the MAC
 8: if $\sigma \neq \mathsf{MACs}[1]$ then
 9:     Drop $C$ and flag upstream router as malicious
10: Drop the first element of MACs and shift to the left
11: $Entry := PIT.\mathsf{Lookup}(C)$
12: if $|\mathsf{MACs}| < Entry.d\}$ then
13:     Drop $C$ and flag upstream router as malicious
14: for $i := 1, \ldots, |Entry.IDs|$ do
15:     $k_{ID} := \mathsf{KeyTable}[Entry.IDs[i]]$
16:     $\mathsf{MACs}[i] = \mathsf{MACs}[i] \oplus (MAC(k_{ID}, D||V)$
17: return $C$, $Entry.F_{out}$
```

network to find a next-best route. If there are no longer any viable routes, R_i must generate interest NACKs.

6.3.4 Security Analysis

Security of our integrity zones scheme is premised on whether $\mathcal{A}$ can compromise more than k contiguous routers. If $k = 2$ and $\mathcal{A}$ compromises two contiguous routers, any part of the packet can be modified without any downstream router detecting the problem. However, if k is large enough such that $\mathcal{A}$ can not succeed, then integrity zones protects against generation and modification attacks. We argue this below.

Claim 6.1. *Assuming a MAC scheme secure against existential forgeries, and $\mathcal{A}$ that is unable to: (a) compromise k contiguous routers along paths of length at least k routers, or (b) compromise l contiguous routers along paths of length $l < k$, and a distance-vector* *routing protocol that provides correct anchor distances, the integrity zones scheme is secure against content modification and generation attacks.*

Proof. First, we show that modification attacks are impossible. This follows from security properties of the underlying MAC scheme. If $c < k$ contiguous routers are compromised and a packet is modified, then at least one MAC tag in a downstream router would fail verification. This MAC corresponds to the upstream router which is malicious.

Assuming a namespace ownership authority, a producer can only advertise a namespace which it owns. Therefore, data generation occurs when: (a) an on-path router maliciously intercepts interests and responds with fake content, or (b) the producer provides fake content. We address these cases together since they differ only in the distance to the producer. Assume that the distance to the producer for the fake content from the verifying router is d. Also, suppose that a packet with l MACs verifies correctly. There are three cases with respect to l, k, and d: (1) $l = k$ and $d \geq k$, (2) $l = k$ and $d < k$, and (3) $l < k$. (Note that $l > k$ is not possible based on the packet format.) In (1) and (2), R fails to detect the attack only if a MAC was forged.[22] Whereas, (3) only occurs if the packet traverses a path shorter than k hops. Per our assumption, $\mathcal{A}$ can not compromise routers along this path and thus an attack can not occur.

We note that interests only carry router IDs and not MACs. Tampering with this list by appending, removing, or changing IDs would only cause downstream routers to fail verification. This is because upstream routers would use wrong keys to generate MACs. □

As indicated above, one limitation of the integrity zone scheme is with paths of length $l < k$ routers. If $\mathcal{A}$ can compromise all l routers, the scheme fails to detect the attack.

[22]A MAC forgery occurs when $\mathcal{A}$ generates a valid MAC without knowledge of the secret key.

6.3.5 Performance Assessment

We now assess the overhead of per-packet verification with integrity zones. From a performance perspective, computation overhead is incurred at every hop. Specifically, the scheme involves at most $2k$ MAC operations: k verifications and k generations. Therefore, network topology has no impact on per-packet overhead. Thus, our assessment has two parts: (1) measuring efficiency of various MAC schemes, and (2) measuring impact of integrity zones on consumer latency.

MAC Overhead

Let m be a message of $|m|$ blocks that serves as MAC input and K be the corresponding key. There are many popular MAC schemes, e.g.,: CMAC [294] and HMAC [193]. HMAC is defined as follows:

$$HMAC(K, m) = H((K' \oplus \mathsf{opad})||H((K' \oplus \mathsf{ipad})||m)),$$

where opad and ipad are constants defined in the standard and K' is a key derived from the master key K. CMAC, or CBC-MAC, is defined (roughly) as follows:

$$\begin{aligned} B_0 &= AES(K, m_0) \\ B_i &= AES(K, B_{i-1} \oplus m_i) \\ B_{|m|} &= AES(K, B_{i-1} \oplus m_i \oplus K') \\ CMAC(K, m) &= B_{|m|} \end{aligned}$$

where K' is a key derived from K and B_i is the i-th output of encrypting m in CBC mode.

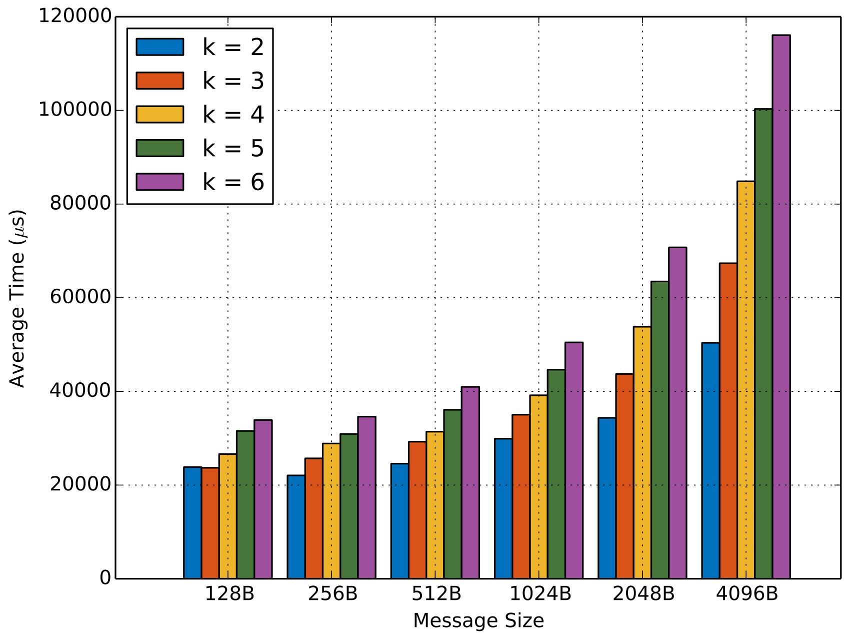


Figure 6.23: HMAC overhead

Both HMAC and CMAC require a complete pass over m. The choice depends on what is available on the run-time platform. In general, we view HMAC and CMAC as approximately equivalent in terms of performance (in the absence of AES acceleration). Figure 6.23 shows the performance of HMAC as a function of message size. We see that overhead reaches the millisecond mark when k approaches 6 and $|m|$ exceeds 4KB.

However, this can be improved with the following optimization. Notice that both HMAC and CMAC process all of m to produce a digest. Cryptographically, these functions provide the equivalent security guarantees if a cryptographic hash digest of m, i.e., $H(m)$ for some pre-image resistant hash function, is the input to MAC. Pre-image resistance stipulates that it must be infeasible to find another message m' such that $H(m) = H(m')$, i.e., forgery remains infeasible. With this in mind, our optimization works by first computing $H(m)$ and using that as input to each MAC operation. Figure 6.24 shows this optimization as a function of message size and k. (We use SHA-256 as H, given its widespread use in CCN.) Results indicate a significant reduction in processing time for nearly all data points.

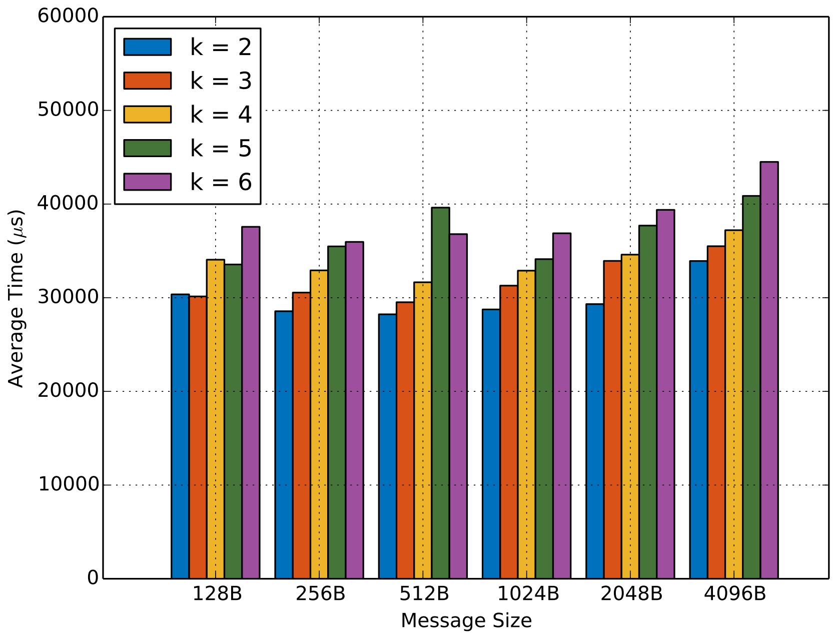


Figure 6.24: Hashed HMAC overhead

Network Impact

To assess impact on consumer latency, we modified the ccns3Sim simulator [247, 2], based on ns3 [269], to support our scheme. We then created a simple N-node path between Cr and P. Every second, Cr issues an interest for a random content produced by P. We measure end-to-end latency as a function of N with $k = 2$, which includes per-packet overhead induced by each router. Results are shown in Figure 6.25, which plots degradation, i.e., percentage latency increase over standard CCN.

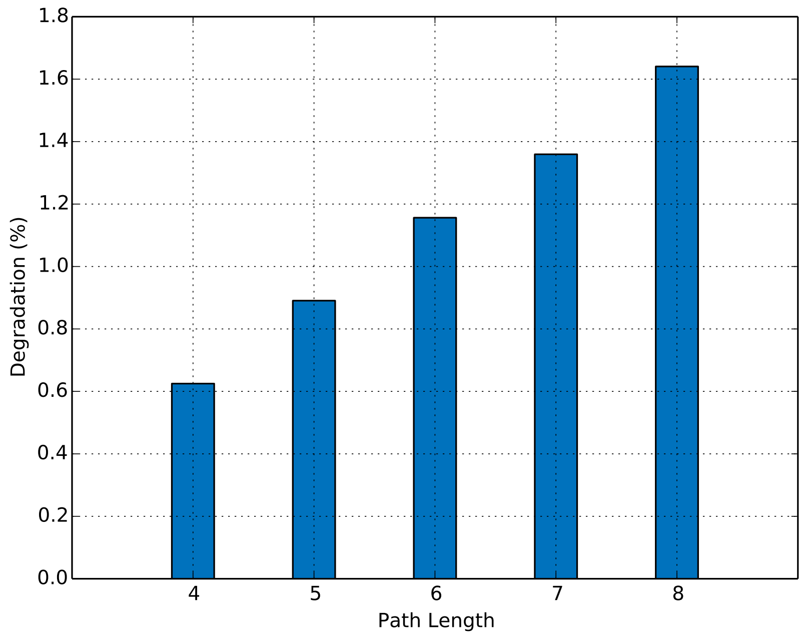


Figure 6.25: Integrity zone latency reduction

6.3.6 Discussion and Challenges

Scalability

The integrity zones scheme entails lightweight hop-by-hop packet integrity checking that is both computation- and bandwidth-efficient. However, it still incurs certain overhead. Since each router must share a key with every router $i = 2, \ldots, k$ hops away, the number of keys can become quite large. For example, suppose that R has 5 neighbors, each of which has its own 5 (distinct) neighbors, excluding R. Then, if $k = 2$, R would have to store 25 keys for each of these routers. Thus, storage overhead is a function of k and network topology. In a highly connected network where routers have many neighbors, this may be prohibitive. However, in such a scenario, it is far less likely for $\mathcal{A}$ to compromise all routers $i = 2, \ldots, k$ hops away. Therefore, k can actually be lower. When there are fewer path choices for a router, it pays to have a larger k, since $\mathcal{A}$'s job becomes harder.

The per-packet overhead in each packet could be dealt with by only generating and verifying MACs at AS boundaries. This would require ASes to share keys with other ASes in the network. This may be more plausible than individual routers sharing keys since trust relationships at the AS level are more common. Specifically, two AS operators can decide whether or not to trust each other and, if so, allocate a key to the corresponding gateways. Although this may simplify key management, it makes the attack origin detection less accurate. When MACs are verified and generated by individual routers, i.e., not just gateways, a router learns precisely where the on-path attack took place. At the AS level, a router (gateway) only learns that an attack took place within an upstream AS.

Global Zone Size

We assumed that k is a global constant. However, this need not be the case. Indeed, k may be AS-specific. It is the responsibility of a gateway between ASes to bridge the gap between different zones sizes. For example, suppose that an interest is generated in AS D_1 with $k = k_1$. It traverses a gateway G to another AS D_2 with $k = k_2$. G must share keys with all of its neighbors at most $\max\{k_1, k_2\}$ hops away. When it receives the interest, it verifies k_1 MACs (if present) and injects its own. Upon receipt of the corresponding content, it verifies k_2 MACs and injects $\max\{k_1, k_2\}$ MACs before forwarding it downstream. If $k_1 < k_2$, each router in D_1 would be able to verify its k_1 MACs. However, if $k_1 \geq k_2$, no router in D_1 that is k_1 or more hops away from G can verify MACs generated upstream of G.

Privacy

At each hop, the integrity zones scheme requires interests (respectively, content objects) to identify k downstream (respectively upstream) routers using their KeyId-s. This reveals path information to each router, something that was not previously visible in CCN. This may be problematic at the network edges, specifically, when a R is less than k hops away from a consumer-facing router.

Chapter 7

Conclusion

This dissertation addressed security, privacy, and availability challenges in CCN. Chapter 4 explored different facets of access control. We began in Section 4.1 by proposing CCN-PRE, a variant of content-based access control (CBAC) that uses (identity-based) proxy re-encryption to simplify principal access key credential management by building off unique name-to-data bindings. Using hybrid encryption, CCN-PRE incurs negligibly small cryptographic overhead compared to network overhead. Certain variants of CCN-PRE also permit in-network nodes to pre-fetch content re-encryption keys on behalf of consumers at the cost of privacy.

We also discussed interest-based access control (IBAC) in Section 4.2, wherein access control information is bound to requests (interests) instead of responses (content). We described two IBAC variants based on different name obfuscation functions: encryption- and hash-based. IBAC is secure when $\mathcal{A}$ is not on-path and can be modified to mitigate replay attacks from an on-path $\mathcal{A}$. Replay prevention requires network participation since caching routers must honor producer-specified access control preferences and verify interest signatures. IBAC is suitable for name privacy and access control on its own or in conjunction with CBAC. The latter is necessary if, for example, router trust and participation is infeasible.

One problem common to CBAC and IBAC is revocation. Producers have no way to flush stale or sensitive content from network caches on demand. To address this shortcoming, Section 4.3 presented BEAD – Best Effort Autonomous Deletion – as a way to securely communicate content deletion requests to caches. BEAD includes numerous deletion request forwarding techniques, such as reverse-path flooding, content traceback with in-cache and local forwarding histories, and packet tracing. Experiments show that BEAD network processing is negligible, though with imperfect coverage.

Chapter 5 surveyed privacy problems in CCN, with particular focus on elements of and requirements for data privacy. In Section 5.1, we described two variants of data privacy: weak and strong. Weak privacy requires deterministic name obfuscation and content encryption. In contrast, strong privacy requires randomized name and content encryption, which (essentially) reduces to end-to-end communication akin to TLS-protected traffic in today's Internet. Weak privacy via deterministic encryption benefits consumers that request identical content (if it is cached). However, deterministic encryption is also susceptible to frequency analysis attacks. In Section 5.2, we showed how a globally or locally distributed adversary can use auxiliary popularity information about plaintext content along with observed (encrypted) request frequencies to map encrypted requests to plaintext responses. Our experiments indicate that caching helps mitigate this attack by limiting adversary visibility of request frequency. However, caches also permit adversaries to probe for recently requested content, which is a different form of privacy attack. (See Acs et al. [23] for more details.) Thus, caching benefits in CCN remain unclear at best.

Total encapsulation, or tunneling, is useful when consumers or producers require stronger notions of privacy. For this purpose, we presented $\mathsf{AC^3N}$, a modification of ANDāNA, which is the first Tor-like system for anonymous communication in CCN (and NDN). Similar to

Tor, $\mathsf{AC^3N}$ creates concentric circuits of anonymizing routers that tunnel traffic between consumers and producers. $\mathsf{AC^3N}$ uses only symmetric key cryptography for efficient processing, without session linkability problems. Specifically, static key identifiers are not sent in the clear. $\mathsf{AC^3N}$ performs favorably compared to unlinkable variants of ANDāNA.

Section 5.4 presented CCVPN, a network-layer tunnel akin to IPsec. CCVPN enables private tunnels between a consumer or origin gateway and gateways associated with specific namespaces. Interests and content are totally encapsulated in transit between tunnel endpoints. This restricts caching and other in-network processing techniques to private domains, similar to IPsec tunnels. CCVPN tunnels can be nested to build $\mathsf{AC^3N}$ anonymizing circuits.

In cases where end-to-end encryption or tunneling are not appropriate, privacy should still be feasible without an opt-in requirement. To that end, Section 5.5 presented TRAPS, a mechanism that encrypts static content $C(N)$ for any consumer who knows N. TRAPS builds on message-locked encryption [52] to encrypt content with a key derived from the (a) name, (b) content payload, or (c) both. Content that requires a ContentId to obtain is thus difficult to decrypt with a priori knowledge. In contrast, content with popular names are trivial to decrypt. TRAPS is designed to offer protection for unpopular content that is known by some, though not all, consumers.

Chapter 6 focused on availability concerns in CCN. We began with an assessment of PITs and their mandatory role in CCN. We showed that many claimed uses and benefits are unnecessary or feeble at best. As an alternate design, we proposed a stateless CCN architecture in Section 6.1 which replaces PIT state with Backwards Routable Names (BRNs). Interests carry a content name and originating BRN. Routers forward content with BRNs back towards origins using FIB state. Experiments indicate that stateless content forwarding is no worse than standard interest forwarding. Thus, little additional network overhead is added. Furthermore, interest origins need not be a consumer; they can be gateway routers that bridge stateful CCN with our stateless CCN variant. This permits hybrid stateless and stateful CCN deployments.

In Section 6.2, we also proposed a new network naming scheme that removes static application name segments from wire-encoded packets. Network names replace hierarchical name segments with a sequence of (truncated) cryptographic hash digests, preceded by a full cryptographic hash of the entire name (fingerprint). Name fingerprints are used to index PITs and caches, thereby removing expensive name hash computations. This also removes a DoS attack vector from processing long names. A secondary benefit is that less application data is conveyed to the network. Analytical results suggest that network name segment hashes need to be at least 24B for resistance against name collision attacks. Experimental results show FIB lookup times with a variety of FIB algorithms and data structures are substantially improved in contrast to standard CCN names.

Finally, we finished with a study of on-path attacks in Section 6.3. We argued that on-path attacks in practice reduce to a lack of network-layer integrity checks. We then presented integrity zones, which are network subsets in which all network routers share pair-wise keys. Each router in a zone shares a key with every other router $l = 2, \ldots, k$ hops away from itself. Routers MAC content objects in transit using keys identified by downstream interest paths. Upon receipt of content, routers verify each MAC from (at most) k upstream routers. If any MAC fails, an attack is detected and forwarding tables are adjusted to bypass the culprit. Integrity zone MACs add no more than 256B to each packet. Processing times are low with a suitable MAC algorithm, e.g., CMAC, and accelerated implementation.

Our study of security and privacy challenges highlights that, although CCN offers improvements as a network architecture, it leaves much to be desired from a security and privacy perspective. Some challenges, such as access control, can and should be addressed by applications. However, such techniques must not negate CCN benefits. Our work in Chapter 4 can be extended to design a holistic access control design that uses both CBAC and IBAC for enforceable access control. BEAD could also be joined with the secure accounting scheme of Ghali et al. [149] to give producers insight into where content is cached. BEAD flooding mechanism can be used in select network domains from where accounting information originated. Other issues, such as privacy, remain elusive. Tunneling and end-to-end encryption help mitigate these problems at significant cost. Conversely, encrypting names and content is more efficient, though insufficient for privacy. Information leaks from name lengths and content popularity can be used to violate user privacy. Our work in Chapter 5 can be extended to study the extent to which specific auxiliary and real popularity distributions affect frequency analysis attacks. Lastly, availability concerns remain a long-standing open problem in CCN. Our proposed architectural adjustments help reduce the attack surface and bring CCN towards a more deployable architecture. Future work could explore hybrid stateful and stateless deployments described in Chapter 6. By leaving PITs and caches at network edges, where they serve more value, a hybrid design maintains many CCN benefits while allowing scalability in network cores.

Bibliography

[1] CCNx distillery. https://github.com/parc/CCNx_Distillery. Accessed: May 14, 2016.

[2] CCNx integrity zone simulator. https://github.com/chris-wood/ccn-onpath-simulation-ccnsim. Accessed: November 23, 2017.

[3] Crypto++. https://www.cryptopp.com. Accessed: November 21, 2016.

[4] Crypto++ 5.6.0 Benchmarks. https://www.cryptopp.com/benchmarks.html. Accessed: November 21, 2016.

[5] DFN-Verein. http://www.dfn.de/. Accessed: February 18, 2016.

[6] DFN-Verein: DFN-NOC. http://www.dfn.de/dienstleistungen/dfninternet/noc/. Accessed: February 18, 2016.

[7] libfib: Library of CCN FIB Implementations. https://github.com/chris-wood/fiblib. Accessed: January 15, 2017.

[8] Mobilityfirst future internet architecture project. http://mobilityfirst.winlab.rutgers.edu. Accessed: August 25, 2017.

[9] Network simulator 3 (NS-3). http://www.nsnam.org/. Accessed: February 18, 2016.

[10] ProjectCCNx: Content-Centric Networking CCNx Reference Implementation. https://github.com/ProjectCCNx/ccnx.

[11] The Content Name Collection. http://www.icn-names.net/. Accessed: April 8, 2016.

[12] The Metis CCNx Forwarder. https://github.com/parc/Metis. Accessed: May 14, 2016.

[13] Tor project website. https://www.torproject.org/.

[14] TRAPS performance evaluation code. https://github.com/chris-wood/tsec-performance.

[15] URLBlacklist. http://urlblacklist.com.

[16] WWW FAQs: What is the maximum length of a URL? `https://boutell.com/newfaq/misc/urllength.html`. Accessed: April 8, 2016.

[17] January 2016. Accessed: August 25, 2017.

[18] Akamai's State of the Inernet Security, Q1 2017 Execute Summary. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-security-executive-summary.pdf., 2017.

[19] libsodium: The sodium crypto library. `https://github.com/jedisct1/libsodium`, 2017.

[20] IETF QUIC Working Group, 2017 (Accessed June 24, 2017).

[21] M. Aamir and S. M. A. Zaidi. Denial-of-service in content centric (named data) networking: a tutorial and state-of-the-art survey. *Security and Communication Networks*, 8(11):2037–2059, 2015.

[22] G. Acs, M. Conti, P. Gasti, C. Ghali, and G. Tsudik. Cache privacy in named-data networking. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 41–51. IEEE, 2013.

[23] G. Acs, M. Conti, P. Gasti, C. Ghali, G. Tsudik, and C. Wood. Privacy-aware caching in information-centric networking. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2017.

[24] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang. Interest flooding attack and countermeasures in named data networking. In *IFIP Networking Conference*, 2013.

[25] A. Afanasyev, J. Shi, L. Wang, B. Zhang, and L. Zhang. Packet fragmentation in NDN: why NDN uses hop-by-hop fragmentation. Technical report, NDN-0032, rev. 1, 2015.

[26] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto, et al. NFD developer's guide. *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021*, 2014.

[27] P. K. Agyapong and M. Sirbu. Economic incentives in information-centric networking: implications for protocol design and public policy. *IEEE Communications Magazine*, 50(12), 2012.

[28] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone. Design considerations for a network of information. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 66. ACM, 2008.

[29] M. Aiash and J. Loo. A formally verified access control mechanism for information centric networks. In *e-Business and Telecommunications (ICETE), 2015 12th International Joint Conference on*, volume 4, pages 377–383. IEEE, 2015.

[30] S. Al-Sheikh, M. Wählisch, and T. C. Schmidt. Revisiting countermeasures against ndn interest flooding. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 195–196. ACM, 2015.

[31] M. Almishari, P. Gasti, N. Nathan, and G. Tsudik. Optimizing bi-directional low-latency communication in named data networking. *ACM SIGCOMM Computer Communication Review*, 44(1):13–19, 2013.

[32] B. A. Alzahrani, M. J. Reed, and V. G. Vassilakis. Enabling z-filter updates for self-routing denial-of-service resistant capabilities. In *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*, pages 100–105. IEEE, 2012.

[33] B. A. Alzahrani, V. G. Vassilakis, and M. J. Reed. Key management in information centric networking. *International Journal of Computer Networks & Communications*, 5(6):153, 2013.

[34] B. A. Alzahrani, V. G. Vassilakis, and M. J. Reed. Securing the forwarding plane in information centric networks. In *Computer Science and Electronic Engineering Conference (CEEC), 2013 5th*, pages 174–178. IEEE, 2013.

[35] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. Freedman, A. Haeberlen, Z. Ives, A. Krishnamurthy, et al. Nebula-a future internet that supports trustworthy cloud computing. *White Paper*, 2010.

[36] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, et al. *The nebula future internet architecture*. Springer, 2013.

[37] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, et al. A brief overview of the NEBULA future internet architecture. *ACM SIGCOMM Computer Communication Review*, 44(3), 2014.

[38] F. Angius, C. Westphal, M. Gerla, and G. Pau. Drop dead data-what to expect securing data instead of channels. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 267–275. IEEE, 2015.

[39] A. Araldo, M. Mangili, F. Martignon, and D. Rossi. Cost-aware caching: optimizing cache provisioning and object placement in icn. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 1108–1113. IEEE, 2014.

[40] A. Araldo, D. Rossi, and F. Martignon. Design and evaluation of cost-aware information centric routers. In *Proceedings of the 1st international conference on Information-centric networking*, 2014.

[41] A. Araldo, D. Rossi, and F. Martignon. Cost-aware caching: Caching more (costly items) for less (isps operational expenditures). *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1316–1330, 2016.

[42] S. Arianfar, T. Koponen, B. Raghavan, and S. Shenker. On preserving privacy in content-oriented networks. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 19–24. ACM, 2011.

[43] M. Arye, R. Kiefer, K. Super, E. Nordström, M. J. Freedman, E. Keller, T. Rondeau, and J. M. Smith. Increasing network resilience through edge diversity in NEBULA. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3), 2012.

[44] M. R. Asghar, C. Bernardini, and B. Crispo. Protector: Privacy-preserving information lookup in content-centric networks. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–7. IEEE, 2016.

[45] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[46] J.-P. Aumasson and D. J. Bernstein. Siphash: a fast short-input prf. In *International Conference on Cryptology in India*, pages 489–508. Springer, 2012.

[47] F. Baker and P. Savola. RFC 3704: Ingress filtering for multihomed networks. Technical report, 2004.

[48] M. Baugher, B. Davie, A. Narayanan, and D. Oran. Self-verifying names for read-only named data. In *INFOCOM Workshops*, 2012.

[49] F. Begtasevic and P. Van Mieghem. Measurements of the hopcount in internet. In *PAM*, 2001.

[50] A. Belenky and N. Ansari. IP traceback with deterministic packet marking. *IEEE communications letters*, 7(4), 2003.

[51] W. Bellante, R. Vilardi, and D. Rossi. On netflix catalog dynamics and caching performance. In *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2013 IEEE 18th International Workshop on*, pages 89–93. IEEE, 2013.

[52] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 296–312. Springer, 2013.

[53] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 3986: Uniform resource identifier (URI): Generic syntax. 2005.

[54] D. Bernstein. Curvecp: Usable security for the internet. *URL: http://curvecp.org*, 2011.

[55] D. J. Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[56] D. J. Bernstein. Dnscurve: Usable security for dns, 2009.

[57] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[58] G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi. Check before storing: What is the performance price of content integrity verification in lru caching? *ACM SIGCOMM Computer Communication Review*, 43(3):59–67, 2013.

[59] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[60] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. The case for ubiquitous transport-level encryption. In *USENIX Security Symposium*, pages 403–418, 2010.

[61] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. Simple opportunistic encryption. 2014.

[62] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. *Advances in Cryptology—EUROCRYPT'98*, pages 127–144, 1998.

[63] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[64] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on gcm in tls. 2016.

[65] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 440–456. Springer, 2005.

[66] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.

[67] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[68] P. Brass. *Advanced data structures.* Cambridge University Press Cambridge, 2008.

[69] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134. IEEE, 1999.

[70] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4), 2004.

[71] J. Burke, P. Gasti, N. Nathan, and G. Tsudik. Securing instrumented environments over content-centric networking: the case of lighting control and ndn. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 394–398. IEEE, 2013.

[72] J. Burke, P. Gasti, N. Nathan, and G. Tsudik. Secure sensing over named data networking. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 175–180. IEEE, 2014.

[73] G. Carofiglio, M. Gallo, and L. Muscariello. Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 37–42. ACM, 2012.

[74] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling data transfer in content-centric networking. In *Proceedings of the 23rd international teletraffic congress*, pages 111–118. International Teletraffic Congress, 2011.

[75] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Pending interest table sizing in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 49–58. ACM, 2015.

[76] A. Chaabane, E. De Cristofaro, M. A. Kaafar, and E. Uzun. Privacy in content-oriented networking: Threats and countermeasures. *ACM SIGCOMM Computer Communication Review*, 43(3):25–33, 2013.

[77] N. Chand, R. C. Joshi, and M. Misra. Cooperative caching in mobile ad hoc networks based on data utility. *Mobile Information Systems*, 3(1):19–37, 2007.

[78] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[79] H. Che, Z. Wang, and Y. Tung. Analysis and design of hierarchical web caching systems. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1416–1424. IEEE, 2001.

[80] T. Chen, K. Lei, and K. Xu. An encryption and probability based access control model for named data networking. In *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, pages 1–8. IEEE, 2014.

[81] S. S. Chow, J. Weng, Y. Yang, and R. H. Deng. Efficient unidirectional proxy re-encryption. In *International Conference on Cryptology in Africa*, pages 316–332. Springer, 2010.

[82] CNN. Latest NSA leaks point finger at high-tech eavesdropping hub in UK, 2013. http://www.cnn.com/2013/12/20/world/europe/nsa-leaks-uk/.

[83] B. Cohen. The BitTorrent protocol specification, 2008.

[84] A. Compagno, M. Conti, P. Gasti, L. V. Mancini, and G. Tsudik. Violating consumer anonymity: Geo-locating nodes in named data networking. In *Applied Cryptography and Network Security*, 2015.

[85] A. Compagno, M. Conti, P. Gasti, and G. Tsudik. Poseidon: Mitigating interest flooding ddos attacks in named data networking. In *38th Conference on Local Computer Networks (LCN)*, 2013.

[86] A. Compagno, M. Conti, C. Ghali, and G. Tsudik. To NACK or not to NACK? negative acknowledgments in information-centric networking. In *The 24th International Conference on Computer Communications and Networks (ICCCN)*, 2015.

[87] M. Conti, P. Gasti, and M. Teoli. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks*, 57(16), 2013.

[88] A. Cooper, H. Tschofenig, B. Aboba, et al. Privacy considerations for internet protocols. *Internet Architecture Board*, 2013.

[89] D. Cooper. RFC 3280: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. 2008.

[90] R. S. da Silva and S. D. Zorzo. An access control mechanism to ensure privacy in named data networking using attribute-based encryption with immediate revocation of privileges. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 128–133. IEEE, 2015.

[91] A. Dabirmoghaddam, M. Dehghan, and J. Garcia-Luna-Aceves. Characterizing interest aggregation in content-centric networks. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 449–457. IEEE, 2016.

[92] H. Dai, B. Liu, Y. Chen, and Y. Wang. On pending interest table in named data networking. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 211–222. ACM, 2012.

[93] H. Dai, Y. Wang, J. Fan, and B. Liu. Mitigate ddos attacks in ndn by interest traceback. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 381–386. IEEE, 2013.

[94] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15. IEEE, 2003.

[95] C. Dannewitz, J. Golić, B. Ohlman, and B. Ahlgren. Secure naming for a network of information. In *INFOCOM IEEE Conference on Computer Communications Workshops*, 2010.

[96] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (NetInf)–an information-centric networking architecture. *Computer Communications*, 36(7), 2013.

[97] A. De Caro and V. Iovino. jpbc: Java pairing based cryptography. In *Computers and communications (ISCC), 2011 IEEE Symposium on*, pages 850–855. IEEE, 2011.

[98] R. De La Briandais. File searching using variable length keys. In *Western joint computer conference*, 1959.

[99] D. De Vleeschauwer and K. Laevens. Performance of caching algorithms for iptv on-demand services. *IEEE Transactions on broadcasting*, 55(2):491–501, 2009.

[100] S. E. Deering. RFC 2460: Internet protocol, version 6 (IPv6) specification, 1998.

[101] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley. On the analysis of caches with pending interest tables. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 69–78. ACM, 2015.

[102] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *SIGMOD/PODS*, 2006.

[103] R. H. Deng, J. Weng, S. Liu, and K. Chen. Chosen-ciphertext secure proxy re-encryption without pairings. In *International Conference on Cryptology and Network Security*, pages 1–17. Springer, 2008.

[104] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212. ACM, 2003.

[105] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. Andana: Anonymous named data networking application. In *NDSS*, 2011.

[106] S. DiBenedetto and C. Papadopoulos. Mitigating poisoned content with forwarding strategy. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pages 164–169. IEEE, 2016.

[107] T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[108] M. Dworkin. *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC.* US Department of Commerce, National Institute of Standards and Technology, 2007.

[109] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *ACM SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

[110] A. Elabidi, G. Ben Ayed, S. Mettali Gammar, and F. Kamoun. Towards hiding federated digital identity: Stop-dissemination mechanism in content-centric networking. In *Proceedings of the 4th international conference on Security of information and networks*, pages 239–242. ACM, 2011.

[111] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, 2000.

[112] S. Farrell, E. Davies, and D. Kutscher. The NetInf protocol. *IRTF Draft, Trinity College Dublin*, 2013.

[113] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker. RFC 6920: Naming things with hashes, 2013.

[114] S. Farrell and H. Tschofenig. Pervasive monitoring is an attack. 2014.

[115] FD.io. Community icn (cicn). `https://wiki.fd.io/view/Cicn`. Accessed: December 2, 2017.

[116] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring HTTPS Adoption on the Web. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017.

[117] P. Ferguson. RFC 2827: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, 2000.

[118] I. Fette and A. Melnikov. RFC 6455: The websocket protocol. 2011.

[119] A. Fiat and M. Naor. Broadcast encryption. In *Annual International Cryptology Conference*, pages 480–491. Springer, 1993.

[120] N. Fotiou, S. Arianfar, M. Särelä, and G. C. Polyzos. A framework for privacy analysis of icn architectures. In *Annual Privacy Forum*, pages 117–132. Springer, 2014.

[121] N. Fotiou, G. Marias, and G. Polyzos. Towards a secure rendezvous network for future publish/subscribe architectures. *Future Internet-FIS 2010*, pages 49–56, 2010.

[122] N. Fotiou, G. F. Marias, and G. C. Polyzos. Publish–subscribe internetworking security aspects. In *Trustworthy Internet*, pages 3–15. Springer, 2011.

[123] N. Fotiou, G. F. Marias, and G. C. Polyzos. Access control enforcement delegation for information-centric networking architectures. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, 2012.

[124] N. Fotiou, P. Nikander, D. Trossen, G. C. Polyzos, et al. Developing information networking further: From psirp to pursuit. In *Broadnets*, pages 1–13. Springer, 2010.

[125] N. Fotiou, D. Trossen, G. F. Marias, A. Kostopoulos, and G. C. Polyzos. Enhancing information lookup privacy through homomorphic encryption. *Security and Communication Networks*, 7(12):2804–2814, 2014.

[126] N. Fotiou, D. Trossen, and G. C. Polyzos. Illustrating a publish-subscribe internet architecture. *Telecommunication Systems*, pages 1–13, 2012.

[127] M. Franz, B. Meyer, and A. Pashalidis. Attacking unlinkability: The importance of context. In N. Borisov and P. Golle, editors, *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2007.

[128] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206. ACM, 2002.

[129] M. Fukushima, A. Tagami, and T. Hasegawa. Efficiently looking up non-aggregatable name prefixes by reducing prefix seeking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 340–344. IEEE, 2013.

[130] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent, yet anonymous, web access with lpwa. *Communications of the ACM*, 42(2):42–47, 1999.

[131] J. Garcia-Luna-Aceves. New directions in content centric networking. In *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*, pages 494–499. IEEE, 2015.

[132] J. Garcia-Luna-Aceves and M. M. Barijough. Content-centric networking using anonymous datagrams. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 171–179. IEEE, 2016.

[133] J. Garcia-Luna-Aceves, A. Dabirmoghaddam, and M. Mirzazad-Barijoug. Understanding optimal caching and opportunistic caching at" the edge" of information-centric networks. In *Proceedings of the 1st international conference on Information-centric networking*, 2014.

[134] J. Garcia-Luna-Aceves and M. Mirzazad-Barijough. Enabling correct interest forwarding and retransmissions in a content centric network. In *ANCS*, 2015.

[135] J. J. Garcia-Luna-Aceves. Name-based content routing in information centric networks using distance information. In *Proceedings of the 1st international conference on Information-centric networking*, pages 7–16. ACM, 2014.

[136] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang. DoS and DDoS in named data networking. In *22nd International Conference on Computer Communications and Networks (ICCCN)*, 2013.

[137] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang. DoS DDoS in named-data networking. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2013.

[138] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. *Advances in cryptology—ASIACRYPT 2002*, pages 149–155, 2002.

[139] C. Ghali, A. Narayanan, D. Oran, et al. Secure fragmentation for content-centric networks. In *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, pages 47–56. IEEE, 2015.

[140] C. Ghali, A. Narayanan, D. Oran, G. Tsudik, and C. A. Wood. Secure fragmentation for content-centric networks (extended version). *arXiv preprint arXiv:1405.2861*, 2014.

[141] C. Ghali, M. A. Schlosberg, G. Tsudik, and C. A. Wood. Interest-based access control for content centric networks. In *Proceedings of the 2Nd ACM Conference on Information-Centric Networking*, ACM-ICN '15, pages 147–156, New York, NY, USA, 2015. ACM.

[142] C. Ghali, G. Tsudik, and E. Uzun. Needle in a haystack: Mitigating content poisoning in named-data networking. In *Proceedings of NDSS Workshop on Security of Emerging Networking Technologies (SENT)*, 2014.

[143] C. Ghali, G. Tsudik, E. Uzun, and C. A. Wood. Closing the floodgate with stateless content-centric networking. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–10, July 2017.

[144] C. Ghali, G. Tsudik, and C. A. Wood. Bead: Best effort autonomous deletion in content-centric networking. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 180–188, May 2016.

[145] C. Ghali, G. Tsudik, and C. A. Wood. Network names in content-centric networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, ACM-ICN '16, pages 132–141, New York, NY, USA, 2016. ACM.

[146] C. Ghali, G. Tsudik, and C. A. Wood. (the futility of) data privacy in content-centric networking. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, pages 143–152, New York, NY, USA, 2016. ACM.

[147] C. Ghali, G. Tsudik, and C. A. Wood. Mitigating on-path adversaries in content-centric networks. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 27–34, Oct 2017.

[148] C. Ghali, G. Tsudik, and C. A. Wood. When encryption is not enough: Privacy attacks in content-centric networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ICN '17, pages 1–10, New York, NY, USA, 2017. ACM.

[149] C. Ghali, G. Tsudik, C. A. Wood, and E. Yeh. Practical accounting in content-centric networking. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 436–444. IEEE, 2016.

[150] Ghali, Cesar and Tsudik, Gene and Uzun, Ersin. Network-layer trust in named-data networking. *SIGCOMM Comput. Commun. Rev.*, 44(5):12–19, Oct. 2014.

[151] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, 2011.

[152] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011.

[153] D. Goergen, T. Cholez, J. François, and T. Engel. A semantic firewall for content-centric networking. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 478–484. IEEE, 2013.

[154] M. T. Goodrich. Efficient packet marking for large-scale IP traceback. In *CCS*, 2002.

[155] M. T. Goodrich. Leap-frog packet linking and diverse key distributions for improved integrity in network broadcasts. In *Security and Privacy, 2005 IEEE Symposium on*, pages 196–207. IEEE, 2005.

[156] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. ACM CCS 2006*, pages 89–98, Oct.–Nov. 2006.

[157] M. Green and G. Ateniese. Identity-based proxy re-encryption. In *Applied Cryptography and Network Security*, pages 288–306. Springer, 2007.

[158] D. Gross. *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[159] F. Guimarães, A. Rocha, C. Albuquerque, and I. Ribeiro. Modeling ndn pit to analyze the limits of timeout on the effectiveness of flooding attacks. In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pages 1245–1250. IEEE, 2016.

[160] C. Gulcu and G. Tsudik. Mixing e-mail with babel. In *Network and Distributed System Security, 1996., Proceedings of the Symposium on*, pages 2–16. IEEE, 1996.

[161] S. Gulley and V. Gopal. Haswell cryptographic performance. *Intel Corporation*, 2013.

[162] P. Gusev and J. Burke. Ndn-rtc: Real-time videoconferencing over named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 117–126. ACM, 2015.

[163] B. Hamdane and S. G. El Fatmi. A credential and encryption based access control solution for named data networking. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1234–1237. IEEE, 2015.

[164] B. Hamdane, A. Serhrouchni, and S. G. El Fatmi. Access control enforcement in named data networking. In *Internet Technology and Secured Transactions (ICITST), 2013 8th International Conference for*, pages 576–581. IEEE, 2013.

[165] D. Han, A. Anand, F. R. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, et al. XIA: Efficient support for evolvable internetworking. In *the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[166] S. Hares, Y. Rekhter, and T. Li. RFC 4271: A border gateway protocol 4 (BGP-4), 2006.

[167] C. L. Hedrick. Rfc 1058: Routing information protocol. Technical report, 1988.

[168] C. V. N. Index. Forecast and methodology, 2014-2019 white paper. Technical report, Cisco, 2015.

[169] M. Ion, J. Zhang, and E. M. Schooler. Toward content-centric privacy in ICN: Attribute-based encryption and routing. In *Proc. ACM SIGCOMM ICN 2013*, pages 39–40, Aug. 2013.

[170] R. Ishiyama, K. Tsukamoto, Y. Koizumi, H. Ohsaki, K. Hato, J. Murayama, and M. Imase. On the effectiveness of diffusive content caching in content-centric networking. In *Information and Telecommunication Technologies (APSITT), 2012 9th Asia-Pacific Symposium on*, pages 1–5. IEEE, 2012.

[171] T. Isshiki, M. H. Nguyen, and K. Tanaka. Proxy re-encryption in a stronger security model extended from ct-rsa2012. In *CT-RSA*, pages 277–292. Springer, 2013.

[172] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th International Conference on Emerging networking experiments and technologies*, 2009.

[173] C. Jayasundara, A. Nirmalathas, E. Wong, and N. Nadarajah. Popularity-aware caching algorithm for video-on-demand delivery over broadband access networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[174] W. John, M. Dusi, and K. C. Claffy. Estimating routing symmetry on single links by passive flow measurements. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 473–478. ACM, 2010.

[175] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *Workshop on the Economics of Information Security (WEIS)*. Citeseer, 2015.

[176] A. Karami and M. Guerrero-Zapata. An anfis-based cache replacement method for mitigating cache pollution attacks in named data networking. *Computer Networks*, 80:51–65, 2015.

[177] A. Karami and M. Guerrero-Zapata. A fuzzy anomaly detection system based on hybrid pso-kmeans algorithm in content-centric networks. *Neurocomputing*, 149:1253–1269, 2015.

[178] K. Katsaros, G. Xylomenos, and G. C. Polyzos. Multicache: An overlay architecture for information-centric networking. *Computer Networks*, 55(4):936–947, 2011.

[179] K. V. Katsaros, L. Saino, I. Psaras, and G. Pavlou. On information exposure through named content. In *Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International Conference on*, pages 152–157. IEEE, 2014.

[180] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols.* CRC press, 2007.

[181] J. Katz and Y. Lindell. *Introduction to modern cryptography.* CRC Press, 2014.

[182] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. RFC 7296: Internet key exchange protocol version 2 (IKEv2), 2014.

[183] H. Kazmi, H. Lakhani, A. Gehani, R. Tahir, and F. Zaffar. To route or to secure: Tradeoffs in icns over manets. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 367–374. IEEE, 2016.

[184] S. Kent. RFC 4302: IP authentication header, 2005.

[185] S. Kent. RFC 4303: IP encapsulating security payload (ESP), 2005.

[186] S. Kent and R. Atkinson. Security architecture for the internet protocol, 1998.

[187] S. Khanvilkar and A. Khokhar. Virtual private networks: an overview with performance evaluation. *IEEE Communications Magazine*, 42(10):146–154, 2004.

[188] D. Kim, S. Nam, J. Bi, and I. Yeom. Efficient content verification in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 109–116. ACM, 2015.

[189] M. Kim, M.-O. Stehr, J. Kim, and S. Ha. An application framework for loosely coupled networked cyber-physical systems. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 144–153. IEEE, 2010.

[190] G. Koloniari, N. Ntarmos, E. Pitoura, and D. Souravlias. One is enough: distributed filtering for duplicate elimination. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 433–442. ACM, 2011.

[191] D. Kondo, T. Silverston, H. Tode, T. Asami, and O. Perrin. Name anomaly detection for icn. In *Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on*, pages 1–6. IEEE, 2016.

[192] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37(4), 2007.

[193] H. Krawczyk, R. Canetti, and M. Bellare. RFC 2104: HMAC: Keyed-hashing for message authentication, 1997.

[194] H. Krawczyk and P. Eronen. Hmac-based extract-and-expand key derivation function (hkdf). Technical report, 2010.

[195] J. Kurihara, C. Wood, and E. Uzuin. An encryption-based access control framework for content-centric networking. *IFIP*, 2015.

[196] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda. Privacy implications of ubiquitous caching in named data networking architectures. *Technical Report TR-iSecLab-0812-001, ISecLab, Tech. Rep.*, 2012.

[197] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda. Privacy risks in named data networking: what is the cost of performance? *ACM SIGCOMM Computer Communication Review*, 42(5):54–57, 2012.

[198] K. Leung, E. W. Wong, and K.-H. Yeung. Designing efficient and robust caching algorithms for streaming-on-demand services on the internet. *World Wide Web*, 7(3):297–314, 2004.

[199] A. Lewko and B. Waters. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–588. Springer, 2011.

[200] B. Li, D. Huang, Z. Wang, and Y. Zhu. Attribute-based access control for icn naming scheme. *IEEE Transactions on Dependable and Secure Computing*, 2016.

[201] B. Li, Z. Wang, D. Huang, and Y. Zhu. Toward privacy-preserving content access control for information centric networking. Technical report, ARIZONA STATE UNIV TEMPE OFFICE OF RESEARCH AND SPONSORED PROJECT ADMINISTRATION, 2014.

[202] Z. Li and J. Bi. Interest cash: an application-based countermeasure against interest flooding for dynamic content in named data networking. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, page 2. ACM, 2014.

[203] K. Liang, Z. Liu, X. Tan, D. S. Wong, and C. Tang. A cca-secure identity-based conditional proxy re-encryption without random oracles. In *International Conference on Information Security and Cryptology*, pages 231–246. Springer, 2012.

[204] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE mobile computing and communications review*, 7(3):19–20, 2003.

[205] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson. F10: A fault-tolerant engineered network. In *the 10th USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[206] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson. Tor instead of IP. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011.

[207] X. Liu, W. Trappe, and Y. Zhang. Secure name resolution for identifier-to-locator mappings in the global internet. In *22nd International Conference on Computer Communications and Networks (ICCCN)*, 2013.

[208] R. Lychev, M. Schapira, and S. Goldberg. Rethinking security for internet routing. *Communications of the ACM*, 59(10):48–57, 2016.

[209] P. Mahadevan, E. Uzun, S. Sevilla, and J. Garcia-Luna-Aceves. CCN-KRS: a key resolution service for CCN. In *Proceedings of the 1st international conference on Information-centric networking*, 2014.

[210] M. Mahdian, S. Arianfar, J. Gibson, and D. Oran. Mircc: Multipath-aware icn rate-based congestion control. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 1–10. ACM, 2016.

[211] M. Mangili, F. Martignon, and S. Paraboschi. A cache-aware mechanism to enforce confidentiality, trackability and access policy evolution in content-centric networks. *Computer Networks*, 76:126–145, 2015.

[212] E. Mannes, C. Maziero, L. C. Lassance, and F. Borges. Assessing the impact of cryptographic access control solutions on multimedia delivery in information-centric networks. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 427–435. IEEE, 2016.

[213] P. Martinez-Julia and A. F. Gomez-Skarmeta. Using identities to achieve enhanced privacy in future content delivery networks. *Computers & Electrical Engineering*, 38(2):346–355, 2012.

[214] P. Martinez-Julia, A. F. Gomez-Skarmeta, J. Girao, and A. Sarma. Protecting digital identities in future networks. In *Future Network & Mobile Summit (FutureNetw), 2011*, pages 1–8. IEEE, 2011.

[215] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM 2.0: A new version of the NDN simulator for NS-3. Technical report, NDN-0028, 2015.

[216] G. Mauri, R. Raspadori, M. Gerlay, and G. Verticale. Exploiting information centric networking to build an attacker-controlled content delivery network. In *Ad Hoc Networking Workshop (MED-HOC-NET), 2015 14th Annual Mediterranean*, pages 1–6. IEEE, 2015.

[217] G. Mauri and G. Verticale. Distributing key revocation status in named data networking. In *Advances in Communication Networking*. 2013.

[218] J. McCann, J. Mogul, and S. E. Deering. RFC 1981: Path MTU discovery for IP version 6, 1996.

[219] S. Misra, R. Tourani, and N. E. Majd. Secure content delivery in information-centric networks: Design, implementation, and analyses. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, pages 73–78. ACM, 2013.

[220] S. Misra, R. Tourani, F. Natividad, T. Mick, N. E. Majd, and H. Huang. Accconf: An access control framework for leveraging in-network cached data in the icn-enabled wireless edge. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[221] A. Mohaisen, H. Mekky, X. Zhang, H. Xie, and Y. Kim. Timing attacks on access privacy in information centric networks and countermeasures. *IEEE Transactions on Dependable and Secure Computing*, 12(6), 2015.

[222] A. Mohaisen, X. Zhang, M. Schuchard, H. Xie, and Y. Kim. Protecting access privacy of cached contents in information centric networks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013.

[223] K. Moriarty and A. Morton. Effect of Pervasive Encryption on Operators. Internet-Draft draft-mm-wg-effect-encrypt-11, Internet Engineering Task Force, Apr. 2017. Work in Progress.

[224] D. R. Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[225] M. Mosko, I. Solis, and C. A. Wood. CCNx Messages in TLV Format. Internet-Draft draft-irtf-icnrg-ccnxmessages-06, Internet Engineering Task Force, Oct. 2017. Work in Progress.

[226] M. Mosko, I. Solis, and C. A. Wood. Content-centric networking-architectural overview and protocol description. *arXiv preprint arXiv:1706.07165*, 2017.

[227] M. Mosko, E. Uzun, and C. A. Wood. CCNx Key Exchange Protocol Version 1.0. Internet-Draft draft-wood-icnrg-ccnxkeyexchange-02, Internet Engineering Task Force, July 2017. Work in Progress.

[228] M. Mosko, E. Uzun, and C. A. Wood. Mobile sessions in content-centric networks. *IFIP Networking*, 2017.

[229] M. Mosko and C. Wood. The CCNx URI Scheme. Internet-Draft draft-mosko-icnrg-ccnxurischeme-01, Internet Engineering Task Force, Apr. 2016. Work in Progress.

[230] M. Mosko and C. A. Wood. Secure fragmentation for content centric networking. In *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*, pages 506–512. IEEE, 2015.

[231] S. Mukherjee, A. Baid, I. Seskar, and D. Raychaudhuri. Network-assisted multihoming in the mobilityfirst future internet architecture. Technical report, Rutgers University, 2013.

[232] S. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195, May 2005.

[233] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in information centric networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 26–31. ACM, 2011.

[234] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560: Online certificate status protocol - OCSP. 1999.

[235] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra. Verifying and enforcing network paths with ICING. In *Proceedings of the 7th Conference on emerging Networking EXperiments and Technologies*, 2011.

[236] T. Narten, R. Draves, and S. Krishnan. RFC 4941: Privacy extensions for stateless address autoconfiguration in IPv6. 2007.

[237] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[238] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the s in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 133–140. ACM, 2014.

[239] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 199–212. ACM, 2015.

[240] S. C. Nelson, G. Bhanage, and D. Raychaudhuri. GSTAR: generalized storage-aware routing for mobilityfirst in the future mobile internet. In *Proceedings of the 6th international workshop on MobiArch*, 2011.

[241] Netflix. Internet Connection Speed Recommendations. `https://help.netflix.com/en/node/306`.

[242] T. Nguyen, R. Cogranne, and G. Doyen. An optimal statistical test for robust detection against interest flooding attacks in ccn. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 252–260. IEEE, 2015.

[243] T. N. Nguyen, R. Cogranne, G. Doyen, and F. Retraint. Detection of interest flooding attacks in named data networking using hypothesis testing. In *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[244] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, Oct. 2015.

[245] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman. Serval: An end-host stack for service-centric networking. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[246] I. O. Nunes, G. Tsudik, and C. A. Wood. Namespace tunnels in content-centric networks. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 35–42, Oct 2017.

[247] PARC. ccns3Sim: CCNx Module for NS3, 2017.

[248] H. Park, I. Widjaja, and H. Lee. Detection of cache pollution attacks using randomness checks. In *Communications (ICC), 2012 IEEE International Conference on*, pages 1096–1100. IEEE, 2012.

[249] C. Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[250] D. Perino, M. Varvello, L. Linguaglossa, R. Laufer, and R. Boislaigue. Caesar: A content router for high-speed forwarding on content names. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 137–148. ACM, 2014.

[251] S. Peter, U. Javed, Q. Zhang, D. Woos, T. Anderson, and A. Krishnamurthy. One tunnel is (often) enough. In *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014.

[252] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange. Minimalt: minimal-latency networking through better security. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 425–438. ACM, 2013.

[253] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):12–19, 2002.

[254] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170. ACM, 2002.

[255] J. Postel. RFC 768 user datagram protocol, 1980.

[256] J. Postel. RFC 791: Internet protocol, 1981.

[257] J. Postel. RFC 792: Internet control message protocol, 1981.

[258] J. Postel. RFC 793: Transmission control protocol, 1981.

[259] T. Przygienda. RFC 3359: Reserved type, length and value (tlv) codepoints in intermediate system to intermediate system, 2002.

[260] W. Quan, C. Xu, A. V. Vasilakos, J. Guan, H. Zhang, and L. A. Grieco. Tb2f: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking. In *Networking conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[261] M. Raykova, H. Lakhani, H. Kazmi, and A. Gehani. Decentralized authorization and privacy-enhanced routing for information-centric networks. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 31–40. ACM, 2015.

[262] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1), 1998.

[263] R. Rembarz, D. Catrein, and J. Sachs. Private domains in networks of information. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[264] Y. Ren, J. Li, S. Shi, L. Li, and X. Chang. An interest control protocol for named data networking based on explicit feedback. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, pages 199–200. IEEE Computer Society, 2015.

[265] E. Renault, A. Ahmad, and M. Abid. Toward a security model for the future network of information. In *Proceedings of the 4th International Conference on Ubiquitous Information Technologies & Applications (ICUT)*, 2009.

[266] E. Renault, A. Ahmad, and M. Abid. Access control to objects and their description in the future network of information. *JIPS*, 6(3):359–374, 2010.

[267] M. Rennhard and B. Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 91–102. ACM, 2002.

[268] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-14, Internet Engineering Task Force, July 2016. Work in Progress.

[269] G. F. Riley and T. R. Henderson. The ns-3 network simulator. *Modeling and tools for network simulation*, pages 15–34, 2010.

[270] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[271] C. E. Rothenberg, P. Jokela, P. Nikander, M. Sarela, and J. Ylitalo. Self-routing denial-of-service resistant capabilities using in-packet bloom filters. In *Computer Network Defense (EC2ND), 2009 European Conference on*, pages 46–51. IEEE, 2009.

[272] N. Rozhnova and S. Fdida. An extended hop-by-hop interest shaping mechanism for content-centric networking. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 1–7. IEEE, 2014.

[273] L. Saino, C. Cocora, and G. Pavlou. Cctcp: A scalable receiver-driven congestion control protocol for content centric networking. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3775–3780. IEEE, 2013.

[274] H. Salah and T. Strufe. Evaluating and mitigating a collusive version of the interest flooding attack in ndn. In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pages 938–945. IEEE, 2016.

[275] H. Salah, J. Wulfheide, and T. Strufe. Lightweight coordinated defence against interest flooding attacks in ndn. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 103–104. IEEE, 2015.

[276] S. Schiffner and S. Clauß. Using linkability information to attack mix-based anonymity services. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 94–107, Berlin, Heidelberg, 2009. Springer-Verlag.

[277] T. C. Schmidt, S. Wölke, N. Berg, and M. Wählisch. Let's collect names: How panini limits fib tables in name based routing. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 458–466. IEEE, 2016.

[278] S. C. Seo, T. Kim, and M. Jang. A privacy-preserving approach in content centric. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 866–871. IEEE, 2014.

[279] A. Serjantov. On the anonymity of anonymity systems. *University of Cambridge, Computer Laboratory, Technical Report*, (UCAM-CL-TR-604), 2004.

[280] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri. Mobilityfirst future internet architecture project. In *Proceedings of the 7th Asian Internet Engineering Conference*, 2011.

[281] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[282] W. Shang, Y. Yu, T. Liang, B. Zhang, and L. Zhang. Ndn-ace: Access control for constrained environments over named data networking. Technical report, NDN Project, Tech. Rep. NDN-0036, Revision 1, 2015.

[283] J. Shao and Z. Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography*, volume 5443, pages 357–376. Springer, 2009.

[284] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 213–226. ACM, 2015.

[285] K. Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, volume 1991, pages 93–99, 1991.

[286] Skype. `http://www.skype.com/`.

[287] D. Smetters, P. Golle, and J. Thornton. CCNx access control specifications. Technical report, Palo Alto Reserach Center, 2010.

[288] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. *ACM SIGCOMM Computer Communication Review*, 31(4), 2001.

[289] W. So, A. Narayanan, and D. Oran. Named data networking on a router: Fast and dos-resistant forwarding with hash tables. In *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 215–226. IEEE Press, 2013.

[290] W. So, A. Narayanan, D. Oran, and M. Stapp. Named data networking on a router: forwarding at 20gbps and beyond. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 495–496. ACM, 2013.

[291] Sodium. Public-key authenticated encryption. `https://download.libsodium.org/doc/public-key_cryptography/authenticated_encryption.html`, 2017.

[292] S. Son and V. Shmatikov. The hitchhiker's guide to dns cache poisoning. *Security and Privacy in Communication Networks*, pages 466–483, 2010.

[293] H. Song, F. Hao, M. Kodialam, and T. Lakshman. Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards. In *INFOCOM 2009, IEEE*, pages 2518–2526. IEEE, 2009.

[294] J. Song, R. Poovendran, J. Lee, and T. Iwata. The aes-cmac algorithm. Technical report, 2006.

[295] T. Song, H. Yuan, P. Crowley, and B. Zhang. Scalable name-based packet forwarding: From millions to billions. In *Proceedings of the 2nd International Conference on Information-centric Networking*, pages 19–28. ACM, 2015.

[296] J. W. Stewart III. *BGP4: inter-domain routing in the Internet*. Addison-Wesley Longman, 1998.

[297] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In *USENIX Security Symposium*, volume 21, page 114, 2000.

[298] K. Suksomboon, Y. Ji, M. Koibuchi, K. Fukuda, S. Abe Nakamura Motonori, M. Aoki, S. Urushidani, and S. Yamada. On incentive-based inter-domain caching for content delivery in future internet architectures. In *Proceedings of the Asian Internet Engineeering Conference*, pages 1–8. ACM, 2012.

[299] P. Syverson. A taxonomy of replay attacks [cryptographic protocols]. In *CSFW*, 1994.

[300] P. Syverson, R. Dingledine, and N. Mathewson. Tor: the second-generation onion router. In *Usenix Security*, 2004.

[301] W. Szpankowski. Patricia tries again revisited. *JACM*, 1990.

[302] J. Tang, Z. Zhang, Y. Liu, and H. Zhang. Identifying interest flooding in named data networking. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 306–310. IEEE, 2013.

[303] F. Tao, X. Fei, L. Ye, and F. J. Li. Secure network coding-based named data network mutual anonymity communication protocol. In *Proceedings of International Conference on Electrical, Computer Engineering and Electronics (ICECEE)*, pages 1107–1114, 2015.

[304] S. Tarkoma, M. Ain, and K. Visala. The publish/subscribe internet routing paradigm (psirp): Designing the future internet architecture. In *Future Internet Assembly*, pages 102–111, 2009.

[305] R. Tourani, T. Mick, S. Misra, and G. Panwar. Security, privacy, and access control in information-centric networking: A survey. *arXiv preprint arXiv:1603.03409*, 2016.

[306] R. Tourani, S. Misra, J. Kliewer, S. Ortegel, and T. Mick. Catch me if you can: A practical framework to evade censorship in information-centric networks. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 167–176. ACM, 2015.

[307] C. Tschudin and C. A. Wood. File-Like ICN Collection (FLIC). Internet-Draft draft-irtf-icnrg-flic-00, Internet Engineering Task Force, June 2017. Work in Progress.

[308] C. Tsilopoulos, G. Xylomenos, and Y. Thomas. Reducing forwarding state in content-centric networks with semi-stateless forwarding. In *INFOCOM, 2014 Proceedings IEEE*, pages 2067–2075. IEEE, 2014.

[309] G. Tsudik, E. Uzun, and C. A. Wood. AC3N: Anonymous communication in content-centric networking. In *13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2016.

[310] W.-G. Tzeng and Z.-J. Tzeng. A public-key traitor tracing scheme with revocation using dynamic shares. In *Proc. PKC 2001*, pages 207–224, Feb. 2001.

[311] A. Vahdat, D. Becker, et al. Epidemic routing for partially connected ad hoc networks. 2000.

[312] M. Varvello, D. Perino, and J. Esteban. Caesar: A content router for high speed forwarding. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 73–78. ACM, 2012.

[313] A. Venkataramani, A. Sharma, X. Tie, H. Uppal, D. Westbrook, J. Kurose, and D. Raychaudhuri. Design requirements of a global name service for a mobility-centric, trustworthy internetwork. In *5th International Conference on Communication Systems and Networks (COMSNETS)*, 2013.

[314] M. Virgilio, G. Marchetto, and R. Sisto. Pit overload analysis in content centric networks. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 67–72. ACM, 2013.

[315] M. Wählisch, T. C. Schmidt, and M. Vahlenkamp. Backscatter from the data plane–threats to stability and security in information-centric network infrastructure. *Computer Networks*, 57(16):3192–3206, 2013.

[316] C. Wang and J. W. Byers. Incentivizing efficient content placement in a global content oriented network. Technical report, Technical Report BUCS-TR-2012-012, Boston University, 2012.

[317] J. Wang and B. Lang. An efficient kp-abe scheme for content protection in information-centric networking. In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pages 830–837. IEEE, 2016.

[318] K. Wang, J. Chen, H. Zhou, and Y. Qin. Content-centric networking: Effect of content caching on mitigating dos attack. *International Journal of Computer Science Issues*, 9(6):43–52, 2012.

[319] K. Wang, J. Chen, H. Zhou, Y. Qin, and H. Zhang. Modeling denial-of-service against pending interest table in named data networking. *International Journal of Communication Systems*, 27(12):4355–4368, 2014.

[320] K. Wang, H. Zhou, H. Luo, J. Guan, Y. Qin, and H. Zhang. Detecting and mitigating interest flooding attacks in content-centric network. *Security and Communication Networks*, 7(4):685–699, 2014.

[321] K. Wang, H. Zhou, Y. Qin, J. Chen, and H. Zhang. Decoupling malicious interests from pending interest table to mitigate interest flooding attacks. In *Globecom Workshops (GC Wkshps), 2013 IEEE*, pages 963–968. IEEE, 2013.

[322] K. Wang, H. Zhou, Y. Qin, and H. Zhang. Cooperative-filter: countering interest flooding attacks in named data networking. *Soft Computing*, 18(9):1803–1813, 2014.

[323] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen. Scalable name lookup in ndn using effective name component encoding. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 688–697. IEEE, 2012.

[324] Y. Wang, T. Pan, Z. Mi, H. Dai, X. Guo, T. Zhang, B. Liu, and Q. Dong. Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters. In *INFOCOM, 2013 Proceedings IEEE*, pages 95–99. IEEE, 2013.

[325] Y. Wang, N. Rozhnova, A. Narayanan, D. Oran, and I. Rhee. An improved hop-by-hop interest shaper for congestion control in named data networking. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 55–60. ACM, 2013.

[326] Y. Wang, D. Tai, T. Zhang, J. Lu, B. Xu, H. Dai, and B. Liu. Greedy name lookup for named data networking. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):359–360, 2013.

[327] Y. Wang, M. Xu, Z. Feng, Q. Li, and Q. Li. Session-based access control in information-centric networks: Design and analyses. In *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, pages 1–8. IEEE, 2014.

[328] W. Wong, F. Verdi, and M. F. Magalhães. A security plane for publish/subscribe based content oriented networks. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 45. ACM, 2008.

[329] C. A. Wood. CCN eavesdropper simulator, 2017.

[330] C. A. Wood. Protecting the long tail: Transparent packet security in content-centric networks. In *IFIP Networking*, 2017.

[331] C. A. Wood and E. Uzun. Flexible end-to-end content security in CCN. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 858–865. IEEE, 2014.

[332] S. Wood, J. Mathewson, J. Joy, M.-O. Stehr, M. Kim, A. Gehani, M. Gerla, H. Sadjadpour, and J. Garcia-Luna-Aceves. Iceman: A practical architecture for situational awareness at the network edge. In *Logic, Rewriting, and Concurrency*, pages 617–631. Springer, 2015.

[333] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

[334] D. Wu, Z. Xu, B. Chen, and Y. Zhang. What if routers are malicious? mitigating content poisoning attack in ndn. In *Trustcom/BigDataSE/I SPA, 2016 IEEE*, pages 481–488. IEEE, 2016.

[335] M. Xie, I. Widjaja, and H. Wang. Enhancing cache robustness for content-centric networking. In *INFOCOM, 2012 Proceedings IEEE*, pages 2426–2434. IEEE, 2012.

[336] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A case for stateful forwarding plane. *Computer Communications*, 36(7):779–791, 2013.

[337] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive forwarding in named data networking. *ACM SIGCOMM computer communication review*, 42(3), 2012.

[338] Y. Yu, A. Afanasyev, and L. Zhang. Name-based access control. *Relatório Técnico TR NDN-0034, University of California, Los Angeles, Los Angeles*, 2015.

[339] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang. An endorsement-based key management system for decentralized ndn chat application. *Technical Report NDN-0023*, 2014.

[340] H. Yuan, T. Song, and P. Crowley. Scalable ndn forwarding: Concepts, issues and principles. In *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pages 1–9. IEEE, 2012.

[341] B. Zantout and R. Haraty. I2p data communication system. In *ICN 2011, The Tenth International Conference on Networks*, pages 401–409, 2011.

[342] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al. Named data networking (NDN) project. Technical report, NDN-0001, Xerox Palo Alto Research Center-PARC, 2010.

[343] L. Zhang and Y. Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *ICDCS*, 2008.

[344] X. Zhang, K. Chang, H. Xiong, Y. Wen, G. Shi, and G. Wang. Towards name-based trust and security for content-centric network. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pages 1–6. IEEE, 2011.

[345] Y. Zhang, H. Zhang, and L. Zhang. Kite: A mobility support scheme for ndn. In *Proceedings of the 1st international conference on Information-centric networking*, pages 179–180. ACM, 2014.

[346] Q. Zheng, G. Wang, R. Ravindran, and A. Azgin. Achieving secure and scalable data access control in information-centric networking. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5367–5373. IEEE, 2015.

[347] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-oriented dns to improve privacy and security (extended). *USC/Information Sciences Institute, Tech. Rep. ISI-TR-2015-695, Feb*, 2015.

[348] Z. Zhu and A. Afanasyev. Let's chronosync: Decentralized dataset state synchronization in named data networking. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

Glossary

anonymizing router An application that encrypts (decrypts) and forwards content (interests). 148

application-layer name A data name assigned by producers and used by consumers. 239

CBAC Access control enforced by content payload encryption.. 36

circuit A series of anonymizing routers. 150

consumer An entity that fetches content with interest messages. 25, 320, 321

content A CCN piece of data addressed using a unique name. 25, 26, 320, 321

FLIC A specific type of manifest whose payload contains collections of pointers to other contents, along with additional metadata about pointer collection. 31

Forwarding Information Base (FIB) A data structure that maps name prefixes to router interface identifiers using longest-prefix match. 27

gateway An entity that processes and translates messages from one domain to another, e.g., by encrypting plaintext interests to encrypted and tunneled interests. 169

IBAC Access control enforced by interest verification. 39

interest A message sent by CCN consumers to request content. 26, 320, 321

manifest A content whose payload contains pointers, e.g., (name, KeyId, ContentId) tuples, to other contents. 31, 320

name A URI-like hierarchical identifier. 25, 321

network name A wire-encoded name carried in interests and content objects. 239

path A sequence of routers over which specific messages traverse. 149

Pending Interest Table (PIT) A cache for interests that stores interest messages, their arrival interfaces, and additional auxiliary information. 27

principal An entity associated with an identity, e.g., consumer node, consumer application, or person. 36

producer An entity that produces (publish) content under a given routable prefix. 25, 321

replay attack A network attacks where adversaries replay legitimate and valid packets in order to gain access to restricted resources or leak private information. 39

routable prefix A minimal name prefix needed to reach a specific producer. 25, 321

router An entity that forward interests and content objects between consumers and producers. 26

trace A sequence of router IDs corresponding to a network path. 77, 82

tunnel An encrypted channel between two endpoints through which messages are totally encapsulated. 169