UNIVERSITY OF CALIFORNIA,
IRVINE

Security and Privacy Issues in Content-Centric Networking

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Cesar Ghali

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

2016

DEDICATION

To Elie ...
May your soul rest in peace

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

Although my name is the only one that appears on this dissertation, I would not have made it so far without the guidance and support of many great people.

My deepest gratitude goes to Gene Tsudik. I can not find the words to thank you enough for always being there during my Ph.D. years. Gene taught me how to do real research, how to always question and better shape my ideas, and how to significantly improve my writing skills. He always knew when to provide the right feedback and at the right time. Gene, thank you for being a great adviser and an amazing teacher, I could not have achieved all of this without your continuous guidance.

I would also like to thank my defense committee members, Gene Tsudik, Marco Levorato and Ersin Uzun, for their support, and for giving me a great defense experience with all the constructive feedback and questions. Thank you very much for serving on my committee.

Special thanks goes to Ersin Uzun, a great mentor, who provided me with the feedback I needed to improve my work. Ersin always made the time for meetings and discussions even when he was on vacations or during holidays, like Thanksgiving dinner. Thank you very much for all your guidance and for flying to Irvine to serve on my advancement and defense committee.

I am also very grateful to my parents who were always there with me in every step I took and constantly encouraged me towards higher education. Thank you very much for your unlimited love and support.

I would like to express my gratitude to Michelle Deville, my best friend, my guardian angel, and my love. You continuously encouraged and supported me to overcome many crisis situations. Thank you very much for always being there for me!

Special thanks goes to all the people I worked with during my Ph.D. (chronological order): Paolo Gasti, Gergely Acs, Mauro Conti, Ersin Uzun, Alberto Compagno, Marc Schlosberg, and Moreno Ambrosin. This dissertation would not have seen the light without your great contributions.

My years at UC Irvine would not have been as exciting without all members of the SPROUT research group (chronological order): Paolo Gasti, Kasper Rasmussen, Mishari Almishari, Quan Nguyen, Naveen Nathan, Sky Faber, Gergely Acs, Mauro Conti, Filipe Beato, Ekin Oguz, Marc Roeschlin, Ronald Petrlic, Jaroslav Sedenka, Michael Steiner, Tyler Kaczmarek, Christopher Wood, Alberto Compagno, Luca Ferretti, Tatiana Bradley, Norrathep Rattanavipanon, and Xavier Carpent. Special thanks to Christopher Wood, who co-authored many of my recent papers. Thank you for being a great colleague and friend. Many thanks goes to Sky and Ekin (amazing friends and roommates) for all the long nights we spent working on different projects or even playing Super Mario.

This dissertation is the outcome of several years of research at UC Irvine. I was supported during these years by fellowships from the Bren School of Information and Computer Science, and the National Science Foundation (NSF) award: "CNS-1040802: FIA: Collaborative Research: Named Data Networking (NDN)".

CURRICULUM VITAE

Cesar Ghali

EDUCATION

Doctor of Philosophy in Networked Systems — **2016**
University of California, Irvine — *Irvine, California*

Master of Science in Networked Systems — **2016**
University of California, Irvine — *Irvine, California*

Master of Engineering in Electrical and Computer Engineering — **2010**
American University of Beirut — *Beirut, Lebanon*

Bachelor of Engineering in Electrical Engineering — **2007**
University of Aleppo — *Aleppo, Syria*

RESEARCH EXPERIENCE

Graduate Research Assistant — **2012–2016**
University of California, Irvine — *Irvine, California*

Research Assistant — **2010**
Deutsches Zentrum für Luft- und Raumfahr — *Oberpfaffenhofen, Germany*

Graduate Research Assistant — **2008–2012**
American University of Beirut — *Beirut, Lebanon*

TEACHING EXPERIENCE

TA for Network & Distributed Systems Security (ICS 203) — **Winter 2015**
University of California, Irvine — *Irvine, California*

TA for Network & Distributed Systems Security (ICS 203) — **Winter 2014**
University of California, Irvine — *Irvine, California*

TA for Computer & Network Security (ICS 134) — **Spring 2013**
University of California, Irvine — *Irvine, California*

PAPERS IN SUBMISSION OR UNDER REVIEW

Privacy-Aware Caching in Information-Centric Networking **2016**
IEEE Transactions on Dependable and Secure Computing (TDSC)

Network Names in Content-Centric Networking **2016**
ACM Conference on Information Centric Networking (ICN'16)

REFEREED JOURNAL PUBLICATIONS

G-Route: An Energy-Aware Service Routing Protocol for Green Cloud Computing **2015**
Cluster Computing

Network-Layer Trust in Named-Data Networking **2014**
ACM Computer Communication Review (CCR)

ServBGP: BGP-Inspired Autonomic Service Routing for Multi-Provider Collaborative Architectures in The Cloud **2014**
Future Generation Computer Systems (FGCS)

REFEREED BOOK CHAPTERS

Reputation as a Service: A System for Ranking Service Providers in Cloud Systems **2014**
Security, Privacy and Trust in Cloud Systems

REFEREED CONFERENCE PUBLICATIONS

BEAD: Best Effort Autonomous Deletion in Content-Centric Networking **2016**
IFIP Networking Conference (NETWORKING'16)

Practical Accounting in Content-Centric Networking **2016**
IEEE/IFIP Network Operations and Management Symposium (NOMS'16)

Interest-Based Access Control for Information Centric Networks **2015**
ACM Conference on Information Centric Networking (ICN'15)

Secure Fragmentation for Content-Centric Networks **2015**
IEEE International Symposium on Network Computing and Applications (NCA'15)

To NACK or not to NACK? Negative Acknowledgments in Information-Centric Networking **2015**
International Conference on Computer Communications and Networks (ICCCN'15)

Needle in a Haystack: Mitigating Content Poisoning in Named-Data Networking **2014**
NDSS Workshop on Security of Emerging Networking Technologies (SEND'14)

Cache Privacy in Named-Data Networking **2013**
International Conference on Distributed Computing Systems (ICDCS'13)

Accountable Energy Monitoring for Green Service Routing in the Cloud **2013**
International Conference on Communications and Information Technology (ICCIT'13)

ABSTRACT

Security and Privacy Issues in Content-Centric Networking

By

Cesar Ghali

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2016

Professor Gene Tsudik, Chair

Content-Centric Networking (CCN) is a networking paradigm alternative to today's IP-based Internet Architecture. One fundamental goal of CCN is to include security and privacy as part of its design. CCN adheres to a simple request and response protocol. Consumers issue interests for named content objects. Routers forward these interests toward content producers. Once the desired content is located, it is returned to the consumer along the same path, in reverse, of corresponding interests. CCN routers can unilaterally cache content to reduce end-to-end latency and bandwidth consumption for future duplicate interests. In this dissertation, we study several security and privacy issues introduced by opportunistic caching in CCN. Specifically, we investigate the influence of caching on consumer and producer privacy, content poisoning attacks, and accounting. For each issue, we describe its root causes, discuss potential countermeasures, and present some experimental results. We conclude that, despite its networking benefits, router caching triggers some important security and privacy problems.

Chapter 1

Introduction

In this dissertation, we focus on security and privacy issues in Content-Centric Networking (CCN) – a prominent instance of the Information-Centric Networking paradigm. We first overview Internet evolution. Then, we present the motivation and goals of CCN and describe how it differs from today's Internet architecture and why it might become a potential replacement. Next, we introduce several security and privacy issues in CCN that are prompted by router caching and propose countermeasures. The contributions are:

- We show attacks on consumer privacy that rely on cached content in routers, and propose countermeasures based on delaying cache-hit responses to mimic cache-miss scenarios. My contribution involved implementing cache privacy attacks and proposed countermeasures as well as running experiments and analyzing results.
- We identify root causes of content poisoning attacks and propose trust management rules at the network layer to overcome such attacks. My contribution consisted of identifying content poisoning causes and developing proposed countermeasures.
- We postulate requirements and design a framework for practical and secure accounting for cache hits and content requests. My role involved studying different accounting

types and requirements in CCN as well as implementing proposed countermeasure and analyzing results.

- We propose a secure cut-through content fragmentation scheme that provides integrity guarantees and optional authentication. My role involved identifying requirements for content fragmentation, implementing the proposed fragmentation scheme (FIGOA), and evaluating results.

- We discuss implications of Negative Acknowledgments (NACKs) and show that supporting secure NACKs at the network-layer opens the door for DoS attacks against content producers. My activities included studying requirements of securing NACKs as well as demonstrating, through simulations and experiments, that producer-based DoS attacks are possible in practice.

1.1 Communication History

Communication is part of natural human behavior that enables transfer of information in a one-to-one, one-to-many, or many-to-many fashion. Inter-human communication started with oral messages. This required close physical proximity in order to convey information, thus potentially necessitating long-distance travel and/or indirect communication (i.e., via intermediaries). To overcome the limitations of oral communication, pigeons, drums and smoke signals were used by many civilizations. This defined early stages of telecommunication.

It was not until early 19th century that electrical telecommunication systems appeared. The electrical telegraph was a revolutionary invention compared to its electromagnetic counterpart [49]. It enabled, later in the same century, installation of the first successful transatlantic cable, making telecommunication between Europe and North America possible.

Audio telecommunication became possible when the telephone was invented in late 19th century by Alexander Graham Bell [81]. This opened the door for the first commercial audio telecommunication services across the Atlantic. In the longer run, this resulted in the deployment of telephone networks managed by human operators.

Early research in wireless communication [75] laid the ground for radio systems – the first devices capable of wireless transmission of human voice. This allowed communication systems to be deployed in more ad-hoc environments such as unpopulated or inhospitable terrains and in war zones. However, society's ambition did not stop at simply transmitting human voices over a wireless channel. In the first half of the 20th century, the invention of television allowed transmission of images and videos over the same channels.

1.2 The Internet of Today

Circuit-switched networking is based on establishing a dedicated channel between two parties [107]. This technology was initially used to transmit simple numerical data and is still used in today's telephone networks. However, the scalability of circuit-switched networking was a challenge when it came to transmitting data over the network. In late 1960s and early 1970s, packet-switched networking was, at the time, considered to be more suitable for data transmission. Data is divided into packets transmitted between computers over a medium shared with other computers. Table 1.1 summarizes circuit and packet switching.

ARPANET, the Advanced Research Projects Agency Network funded by the Department of Defense of the United States [42], was the first implementation of packet-switched networking in late 1960s. In its early stages, ARPANET supported only a few simple applications. The first was email [198, 162]. Email messages constituted the majority of ARPANET traffic in early 1970s.

Table 1.1: Differences between circuit switching and packet switching [2]

	Circuit Switching	**Packet Switching**
Path	Dedicated	Not dedicated
Setup	Per connection	Per packet
Delays	Setup delays	Packet transmission and queuing delays
Overload	Block call setup	Increase packet delays and loss rate
Overhead	No additional headers after setup	Each packet has its own header
Throughput	Fixed	Dynamic

In 1981, two RFCs (Request for Comments), the Internet Protocol v4 (IPv4) [166] and Transmission Control Protocol (TCP) [165], were published. Together, they represented the TCP/IP protocol suite that forms the backbone of today's Internet. The IP protocol is responsible for routing packets, called datagrams, from source to destination host. (We use the terms IP packets and IP datagrams interchangeably.) From the IP perspective, the Internet is viewed as a collection of interconnected networks and Autonomous Systems (AS-s).

Every network entity, e.g., host or router, is identified by an IP address.[1] Each IP address consists of two parts: (1) the network prefix, and (2) the host identifier. This design allows the IP addressing scheme to scale with the number of network hosts.

IP datagrams contain source and destination addresses along with other fields that convey control information. Actual data is carried in a payload field of the datagram. When a packet is received by a router, the latter searches its forwarding table to identify the next hop where the packet should be forwarded. Forwarding tables contain a list of destination network prefixes along with the router's outgoing interfaces and next hop IP addresses. This information allows routers to perform longest-prefix-matching on packet destination addresses to identify next hops. If a packet can not be forwarded, it is dropped and an error message is generated according to the Internet Control Message Protocol (ICMP) [164].[2]

[1] Usually IP addresses are assigned to entity interfaces.

[2] The ICMP protocol is also used for sending control messages, such as routing redirect for networks and hosts.

One important feature provided by IPv4 is fragmentation. Whenever the size of an IP packet is larger than the forwarding interface's Maximum Transmission Unit (MTU) [47], the packet must be divided into smaller chunks, called fragments. Destination hosts must reassemble fragments to recover the original packet. Other network entities, such as Network Address Translation Tables (NAT) [85, 185][3] and in-network firewalls *might* also assemble fragments.

Since IP's longevity and popularity were not foreseen, security and privacy were not supported by design. Therefore, IPsec [177] was designed to provide data-origin authentication, integrity, and confidentiality of IP datagrams. The first two are attained via Authentication Header (AH) protocol [99], while Encapsulating Security Payload (ESP) protocol [100] provides all three security features. IPsec supports two modes of operation:

- *Transport mode*: provides end-to-end communication, e.g., client-server communications. Only packet payloads are encrypted and authenticated in transport mode. Transport and application layers of packets are secured by a hash, thus, they can not be modified, e.g., using NAT. NAT-Traversal (NAT-T) [101] is developed to overcome this issue.

- *Tunnel mode*: is usually used between gateways to provide a secure connection between networks, e.g., different sites of the same organization. Furthermore, tunnel mode is also used to provide secure host-to-gateway communications. IP packets (as a whole) are encrypted and then encapsulated into new IP packets. One application of tunnel mode is Virtual Private Networks (VPN) [126].

IPv6 [62] is a newer version of IP developed to overcome some limitations of IPv4. IPv6 has 128-bit addresses. This significantly increases its address space. However, IPv6 does not support in-network fragmentation. A host that wishes to send an IP datagram must first find

[3]NAT operating on TCP and UDP packets (e.g., fragmented FTP control packets [186]) needs to reassemble IP fragments in order to correctly calculate higher-level checksums and perform the required translation.

the smallest MTU on the path to the destination and fragment the said datagram accordingly. Therefore, a Path MTU Discovery protocol [131] was designed and implemented. Moreover, the IPv6 design takes into consideration security and privacy by implementing some features similar to IPsec, such as AH and ESP, as extension headers [7].

TCP complements IP by providing reliable, ordered communication with flow control, congestion control, and error-free delivery. Arbitrary length application data is split into multiple fixed-sized TCP packets that are encapsulated into IP datagrams. Each TCP packet includes both the source and destination port numbers. Thus, the combination of source and destination IP addresses and port numbers identify a unique flow. Moreover, each TCP packet in a specific flow is assigned a unique sequence number to allow the destination to (1) sort received packets, in case of out-of-order delivery, and (2) request missing packets due to network loss. In addition, TCP implements flow control algorithms that maximize bandwidth utilization and avoid network congestion.

For the rest of this dissertation, we use the term IP to refer to both IPv4 and IPv6, unless otherwise specified.

1.3 The Internet of the Future

The original Internet intended to support a few thousand connected users, mainly in North America, accessing shared resources using terminals. Nowadays, the Internet connects over 3 billion users with a variety of applications ranging from simple web browsing to video conferencing and content distribution. These extreme changes in Internet usage highlighted limitations of the current IP-based architecture and prompted research into next-generation architectures.

To this end, the National Science Foundation (NSF) launched the Future Internet Architecture (FIA) program [16] in 2010. The FIA program originally included four research efforts to fulfill the project goals: Named-Data Networking (NDN) [206], MobilityFirst [178], eXpressive Internet Architecture (XIA) [86], and Nebula [27].

One of the main goals of the FIA program is to consider security and privacy as part of the design of any future Internet architecture. In this section, we summarize three of the above architectures; MobilityFirst, XIA, and Nebula. NDN and its commercial counterpart, Content-Centric Networking (CCNx) [93, 183, 142], are described in Chapter 2.

1.3.1 MobilityFirst

MobilityFirst architecture aims to overcome the inefficiencies and limitations of today's Internet. It focuses on scenarios where wireless connections are *ubiquitous* and *pervasive*. To this end, MobilityFirst has been designed around the concepts of *mobility* and *trustworthiness*. All endpoints must be able to seamlessly switch network connection, and the network must be resilient to compromised endpoints and routers.

MobilityFirst treats *principals* – devices, content, interfaces, services, human end-users, or a collection of identifiers – as primary addressable network entities. To promote mobility, the (constant) identity of a principal and its (dynamic) network location are strictly separated. This requires a distributed Global Name Service (GNS) to bind principal identities to network addresses. Furthermore, identity and network address separation (1) facilitates service implementation and deployment, and (2) supports designing routing protocols that overcome link fluctuation and disconnections [147].

We now briefly describe MobilityFirst's network layer and its Global Name Service.

Network Layer

Two types of identifiers are used to differentiate between principal identities and their physical locations.

- **Global Unique Identifier (GUID):** a flat self-certifying identifier that uniquely identifies a principal. GUIDs can be generated using multiple methods depending on the provided service type. For instance, they can be derived from the public key of a host or a service principal or the hash of a content principal. For the sake of usability, a human readable name can be assigned to a principal and later resolved (by GNS) to the corresponding GUID.
- **Network Address (NA):** a flat address that identifies a *network* to which a particular principal (GUID) is connected. MobilityFirst networks are equivalent to AS-s on today's Internet. NAs can be used to identify finer-grained networks such as subnets or organizations. In cases where principals are connected to multiple networks (e.g., using 3G and WiFi simultaneously), multiple NAs can correspond to the same principal.

As a consequence of this addressing scheme, MobilityFirst defines a new packet type called Packet Data Unit (PDU). PDUs contain source and destination GUIDs, lists of source and destination NAs, payload, and other control fields.

In order to communicate with a specific GUID, endpoints need to query GNS to obtain the corresponding NA. The retrieved tuple (GUID, NA) is then carried in the PDU header as a routable destination identifier. PDUs are first delivered to their corresponding destination NAs (using inter-domain routing), and then to the destination GUIDs (using intra-domain routing). In case of delivery failure, the packet is stored inside the network (in routers) and GNS is periodically queried for a new or updated GUID-NA mapping.

Multihoming, anycast, and multicast are supported by multicast GUIDs (MIDs). MID has the same format as a regular GUID, except its resolution results in a *set of NAs* (instead of, at most, one). Technically, GNS associates one MID with several GUIDs (the ones belonging to the multicast group). Resolving all of them results in one or more elements of the output NAs set.

MobilityFirst can also support content distribution networks. In this case, GUIDs are composed of two parts:

- Content GUID (CID): uniquely identifies the content and is generated by computing the hash of the corresponding content.
- Publisher GUID (PID): points to the network entity providing the content. Such an entity can be the actual content provider, or a third-party content repository.

A router may be equipped with a cache. This opportunistic caching feature facilitates content distribution at the network layer by reducing end-to-end latency and bandwidth consumption.

Global Name Service

GNS is an essential part of the MobilityFirst architecture. Its main task is to map endpoint identifiers (GUIDs or human readable names) to a set of attributes including the endpoint network address. GNS relies on the following two services:

- *Name Certification Service (NCS):* is equivalent to a Certificate Authority (CA). Its purpose is to (1) assign GUIDs to human-readable names and (2) attest this mapping by generating certificates. MobilityFirst allows multiple NCSs without a global root of trust. Moreover, if GUID space is large enough, the need for coordination between different NCSs is eliminated.

- *Global Name Resolution Service (GNRS):* a distributed naming service similar to Domain Name System (DNS) that stores the mapping between GUIDs and NAs [143, 118, 194].[4] Two GNRS implementations are evaluated: (1) a distributed hash table maintained among all AS-s of the Internet (DMap [195]), and (2) a number of replica-controllers that migrate data (GUID-NA mappings) between a variable number of active replicas (Auspice [179]).

Regardless of its implementation, GNRS clients interact with the service by issuing the following requests to the GNRS resolver:

- `insert`: register a new GUID-NA mapping when a principal joins the network.
- `update`: keep the GUID-NA mapping up-to-date when the corresponding principal migrates to a new network location.
- `query`: retrieve the list of NAs associated with a specific GUID.

In [118], a secure version of the above three GNRS request types is proposed. The secure `insert` and `update` requests adopt a two-step approach to check validity of a GUID-NA mapping. Four network entities are involved in this process: (1) the user issuing the new GUID-NA mapping, (2) the local router to which the user is connected, (3) the border gateway router that connects the user's AS to the rest of the Internet, and (4) the DHCP server which assigns the user's address.

The user generates and signs the request containing the GUID-NA mapping. Local and border routers are in charge of verifying validity of the announced mapping. This is achieved by verifying that the announced NA is the network connected to the user (and the local router), and querying the DHCP server to ensure that the returned NA corresponds to the

[4]GNRS is the actual GNS service that is responsible for maintaining GUID-NA mappings.

announced GUID. If the NA matches the one contained in the `update` or the `insert` request, the mapping is accepted and added or updated in the GNRS table.

In the secure `query` request, the protocol involves three entities: the user, the border gateway, and GNRS. The user issues an authenticated request and the border router checks its validity. The router then forwards the request to the appropriate GNRS replica. On receipt, the GNRS satisfies the request with a signed GUID-NA mapping response.

There are several differences [179] between GNRS and DNS [137]. First, GNRS does not restrict the structure of the names, while DNS only supports hierarchical names. Second, scalability of GNRS does not rely on TTL-based caching, which has been proven to be ineffective in the presence of high mobility. Third, GNRS does not statically give the authority to a replicated server for a specific set of names. Active and on-demand replication reduce reliance on passive caching and ensure that mapping replicas are always accessible close to clients.

1.3.2 eXpressive Internet Architecture

eXpressive Internet Architecture (XIA) is another research effort aiming to design a new architecture. XIA is based on three types of principals. *Host*-centric networking can support end-to-end communication, such as video conferencing and file sharing. *Service*-centric networking allows users to access various network services such as printing and data storage services. Meanwhile, *content*-centric networking can support Web browsing and content distribution. However, XIA's design is extensible in that it can adaptively provide network evolution and support any new principal type that might emerge in the future.

A core architectural property of XIA is *intrinsic security* of all principals. Any entity should be able to authenticate the principal it is communicating with, without trusted third parties.

This can be achieved by binding one or more security properties with principal names. For instance, using the hash of a service (or a host) public key as its name allows entities to verify that they are communicating with the desired principal. Similarly, binding content with its name can be achieved using the hash of the content as its name, allowing users to verify the integrity of a requested content.

XIA defines three main design requirements:

1. All network entities must be capable of clearly expressing their intent. This is achieved by designing the network to be *principal*-centric and allowing in-network optimization. Routers can perform principal-specific operations when receiving, processing, and forwarding packets.

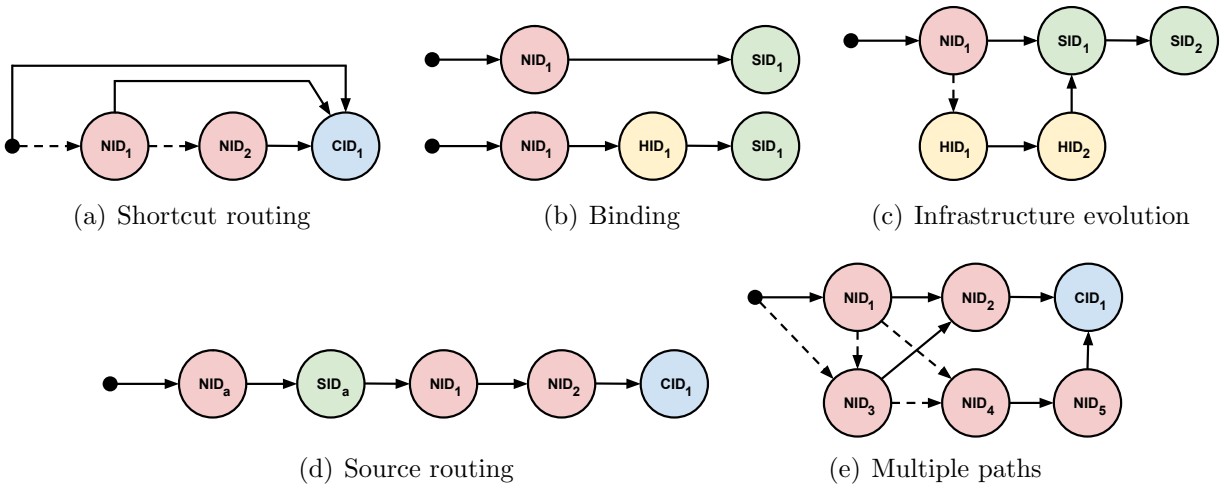
2. The network must be able to adapt to new types of principals. This is essential to support network evolution.

3. Principal identifiers must be intrinsically secure. This depends on the principal type, e.g., authenticating hosts in *host*-centric networking is different than verifying content integrity in *content*-centric networking.

Principal identifiers are denoted as XID, where X defines the type of principle. For instance, HID identifies a host, SID a service, NID a network, and CID a content.

eXpressive Internet Protocol

In order to comply with the aforementioned requirements, eXpressive Internet Protocol (XIP) is designed. XIP defines packet format, addressing schemes, and behavior of all nodes while processing incoming and outgoing packets from/to various principal types. One of the main features of the XIP addressing scheme is flexibility of defining multiple (fallback) paths to destinations. This prevents downtime and service interruption, especially while gradually



(a) Shortcut routing (b) Binding (c) Infrastructure evolution

(d) Source routing (e) Multiple paths

Figure 1.1: XIP addressing styles [86]

deploying new principal types. An XIP address is a directed acyclic graph (DAG) with several properties:

- Each address is a single connected component.
- Each DAG starts with an untyped entry node and ends with one or multiple "sink" nodes. Thus, each node in the address graph has a unique XID except for the entry node.
- Edges define next hops in the path.
- Multiple outgoing edges of a single node are processed in the order they are listed.
- Out-degree of each node is upper bounded to restrict performance overhead.

Using DAGs as a basis for XIP addresses allows applications to build several "styles" of addresses, such as:

- **Shortcut routing** – this style, shown in Figure 1.1(a) is best suitable for requesting content principal. Each node has a direct edge to the destination principal CID_1, which

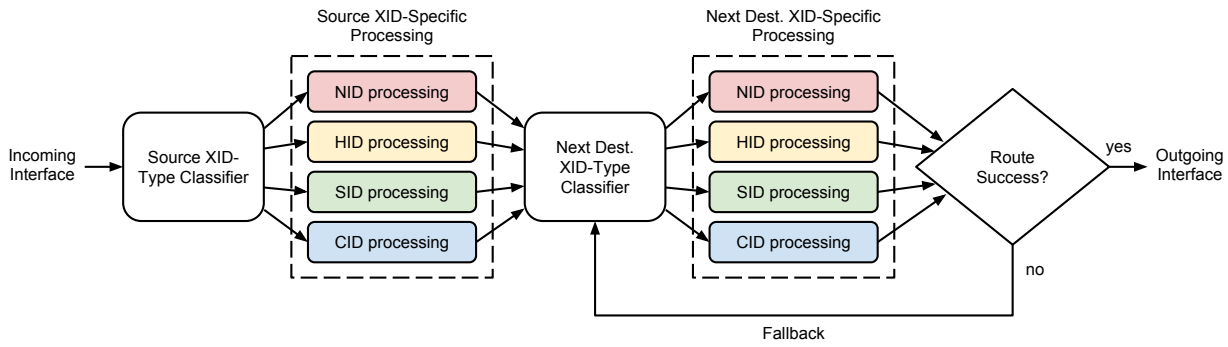


Figure 1.2: XIA router diagram

enables in-network caching. If a node does not have the content cached, the fallback path is processed and the packet is forwarded to the next hop.

- **Binding** – some services require that communication is bound to a specific source or destination. For instance, a service hosted in multiple geographical locations. Users can establish a session with the closest host providing this service. Then, all further communications must be directed to this particular host. Figure 1.1(b) shows an example of this addressing style. The first packet is destined to SID_1, i.e. the closest host, while the second packet is destined to SID_1 provided by a specific host HID_1.

- **Infrastructure evolution** – as mentioned above, XIA supports gradual network evolution for emerging principal types using fallback paths. Figure 1.1(c) shows an example of this style. Assume that NID_1 is gradually deploying service SID_1. All NID_1 routers that are not yet updated to recognize and process SID_1 use the fallback path through HID_1 and HID_2.

- **Source routing** – Figure 1.1(d) gives an example of this addressing style, in which the source routes the packet to the destination through a third party domain and service, NID_a and SID_a, respectively.

- **Multiple paths** – this supports recovery from link failures. An example of this style is shown in Figure 1.1(e).

Figure 1.2 shows a high level overview of an XIA router. Its modular design allows efficient multi-principal processing and supports network evolution. Each router contains two main XID-specific processing modules:

- **Source XID-specific processing:** necessary for certain XID types. For instance, in case of a reply to a CID request, the "CID processing" unit can implement in-network content caching.
- **Next Destination XID-specific processing:** invoked by the *Next Destination XID-specific Classifier* which determines the appropriate forwarding action. Similar to source processing, this module consists of several units that carry on XID-specific operations right before forwarding the packet.

If all outgoing DAG edges of a node lead to unrecognizable XIDs, the packet is dropped and an unreachable destination error is generated. It is the responsibility of user applications to provide appropriate fallback paths to avoid forwarding failures at any router. Usually fallback paths are built using well-supported principals, e.g. HID and NID.

Principals

As mentioned above, principals in XIA support emerging communication paradigm on the current Internet. When introducing a new principal, the following issues arise:

- What does it mean to communicate with a principal of this type?
- How is the principal's unique XID generated and how does it map to intrinsic security properties?
- What are the source and next destination XID-specific processing actions that routers should perform and how can such actions be implemented?

We now describe several principal types and discuss their addressing schemes, in-router processing behaviors, and security properties.

Network and Host. Network and host principal identifiers are denoted as NID and HID, respectively. They are generated by using the public key hash of the network or the host. Unlike hosts on current Internet, each XIA host has a unique HID regardless of the interface it is communicating through. This feature helps support host mobility. In order to support fallback paths, all XIA routers should implement NID and HID processing modules.

As mentioned above, the fact that the network and host addresses are derived from their corresponding public keys allows users to verify the identity of entities with whom they are communicating. Furthermore, this security requirement helps defend against address spoofing, Denial of Service (DoS), and cache poisoning attacks.

Service. Services in XIA represent applications in today's Internet. Users communicating with a service SID can use a destination address of the form NID:HID:SID. In today's terminology, this is analogous to sending a packet to a specific host in a specific network and indicating the associated protocol and port number.

Since different services might require different specialized processing, implementing in-router source and next destination processing modules is a challenge. Therefore, routers are only required to perform default processing, routing, and forwarding of SID packets. All other specialized processing should be handled by end-nodes.

SIDs are generated by computing the hash of the service public key. This inherits security properties similar to NIDs and HIDs.

Content principals. This principal type signifies user's intent to retrieve content. Packets carrying content identifiers (CID) as destination addresses will be routed all the way to the node hosting the content. Routers can use a cached version of the content as a reply to such packets. As mentioned above, caching is implemented by routers source XID-specific processing module.

CIDs are generated based on the cryptographic hash of the content they address. This binds the content to its name, forming a self-certifying name.

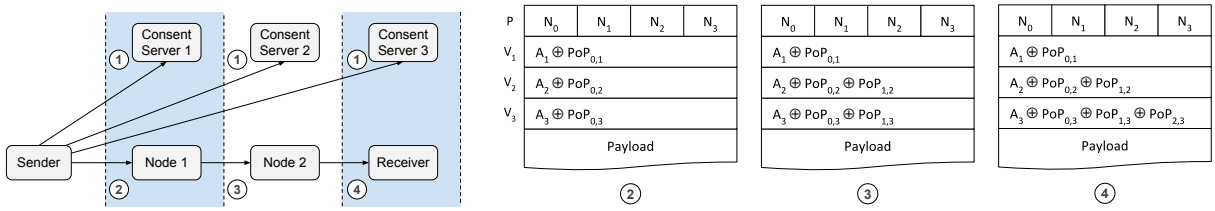
1.3.3 NEBULA

NEBULA [27, 28, 29] is an FIA project focused on providing a secure and cloud-oriented networking infrastructure. Its architecture is composed of three tiers:

- *Network core* (NCore) is a collection of routers and interconnections that provide reliable connectivity between routers and data centers. NCore is based on high-performance core routers and rich interconnected topologies [116].
- *NEBULA Virtual and Extensible Networking Techniques* (NVENT) represents the control plane of NEBULA. NVENT helps in establishing trustworthy routes based on policy routing [33] and service naming [149].
- *NEBULA Data Plane* (NDP) is responsible for routing packets along the paths established by NVENT. To guarantee confidentiality, availability, and integrity, NDP ensures that packets for a specific communication can only be carried when all parties, i.e., end nodes and routers in between, have agreed to participate.

NEBULA Network Layer

The original design of NEBULA specifies different candidate network layer stacks for NDP



(a) Packet forwarding with ICING.

(b) ICING packet high-level structure.

Figure 1.3: ICING architecture

[28], e.g., ICING [145], TorIP [117], and Transit-as-a-Service (TaaS) [157]. From this list, ICING was picked as the most suitable candidate and was included in the Zodiac NEBULA prototype implementation [29]. We now provide a brief overview of ICING.

ICING provides a new primitive, called *Path Verification Mechanism* (PVM), which guarantees the following two properties:

- *Path Consent* – every entity in a path between two hosts *consents* to the use of the whole path before the communication starts.
- *Path Compliance* – ability of each node in a path between two hosts to verify that a received packet (1) follows the approved path and (2) has been "correctly" forwarded by all the previous nodes on that path, according to a specific pre-established policy.

ICING can be deployed either at the network layer or as an overlay on top of IP. In the former case, service providers can deploy ICING nodes as ingress gateways to their networks. However, in the latter case, ICING nodes may become *waypoints*, interconnected using IP, providing waypoint-level path guarantees.

To start communication, a sender must first establish a complete path. Such a path can be provided by DNS with policy enforcement [145]. Figures 1.3(a) and 1.3(b) show how forwarding works in ICING and a high-level representation of how the ICING header evolves.

Once a path is selected, the sender requests a *Proof of Consent* (PoC_j), for each node j on the path (action ① in Figure 1.3(a)). PoCs are cryptographic tokens created by each node transit provider, which attest to the provider's consent to carry packets along the specified path. Each PoC certifies that the corresponding network provider consents to (1) the full path, and (2) a specific policy-based set of local actions (e.g., forwarding) to be performed on packets traversing the path. PoCs are generated by a *consent server*, which is owned by the transit provider or acts on its behalf. Such servers share secret keys with each node (router) in their corresponding providers. Once all PoCs are received, the path is established and packet transmission can begin.

A sender builds a packet header as follows:

1. *Proof of Provenance* (PoP) token, one for each node on the path (action ② in Figure 1.3(b)), is generated using a PoP key k_j shared with the corresponding node j. In Figure 1.3(b), PoPs are denoted as $\text{PoP}_{i,j}$, where i is the index of the node generating the PoP and j is the index of the node for which PoP is generated. Specifically, $\text{PoP}_{0,j}$ is computed by node 0 using k_j, path P, and message M itself.
2. Authenticator A_j is computed for each node j using PoC_j, P and M.
3. Verifiers V_j, one per node, are computed by XORing the corresponding A_j and $\text{PoP}_{i,j}$.

PoP tokens are used by each node on the path to prove that downstream nodes have handled the received packets based on the established policies. When an intermediate node N_i receives a packet, it performs the following actions:

1. Computes the corresponding PoC_i.
2. Computes $\text{PoP}_{j,i}$ using k_j, for each downstream node N_j.

3. Verifies that the received PoC_i and $\text{PoP}_{j,i}$ match the two values computed in the previous two steps. If this verification fails, N_i drops the packet.

4. Derives a shared PoP key k_l, for each upstream node N_l, and computes $\text{PoP}_{i,l}$ as described above.

5. Modifies the verifiers to include the computed PoP, and forwards the packet upstream (actions ③ and ④ in Figures 1.3(a) and 1.3(b)).

The previous steps allow any node to guarantee that all packets are forwarded by all the consenting nodes while establishing the path.

NEBULA Resolution Service

Service resolution in NEBULA is provided by NVENT, using declarative networking [120, 121], and allows administrators to provide high-level specifications of their routing policies. NVENT also involves special interfaces, called *service interfaces*, that enable service access and specify the required level of availability. For instance, an emergency service can request high availability, which can be provided by multi-path interdomain routing. A distributed resolution service is used for discovery of other NVENT services. This service is populated by service providers, e.g., NCore data centers [27].

Serval [149] is a means of service resolution in NEBULA. Serval is a control and data plane that implements service-centric networking [132, 84], which decouples service instances (e.g., web or email services) from their physical locations (i.e., IP address and port). Serval introduces a new layer, the Service Access Layer (SAL), between the (unmodified) network layer and the transport layer. Figure 1.4 presents Serval's protocol stack and highlights its main operations.

With Serval, each service is identified by a *serviceID*, a unique identifier that applications use to communicate with the service. In addition, each local traffic flow, representing a

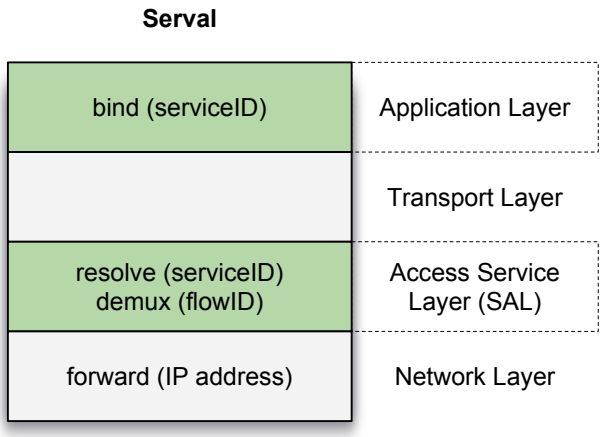


Figure 1.4: Serval protocol stack [149]

connection between two hosts, is identified by a unique *flowID*. The request is handled by SAL, which uses local control plane policies to map the *serviceID* to a service instance. SAL eventually creates a new *flowID* that identifies the established connection. This *flowID* is delivered to the destination host during connection setup, and used by both parties for connection identification. Finally, SAL routes the packet based on specific control plane rules contained in its *SAL table*. For instance, a host application that wants to connect to a specific service might direct the first request to a default Serval router (using its IP address). The SAL of the router then might process the request and take further decisions based on its *SAL table* (e.g., forward to another router or send directly to a known service instance). Moreover, Serval does not allow clients to learn *serviceID*s. It simply suggests the use of directory services or search engines [149].

Figure 1.5 presents a view of how all NEBULA components integrate to allow a user to negotiate a custom end-to-end path to a specific data center and send the desired packets. First, the user (either the mobile phone or the laptop in the figure) contacts NVENT to request a path to NCore. NVENT determines a suitable path that complies with each transit network's policies and contacts the corresponding consent servers to obtain the necessary PoCs. Once the path and all PoCs are delivered to the user, the latter generates appropriate

packet headers and forwards them, using the NDP forwarders network, to the nearest NCore router. This router ensures that all header fields are valid (as described above) and verifies that the negotiated path has actually been traversed. Once verified, the core router forwards received packets to the correct data center using its NCore links.

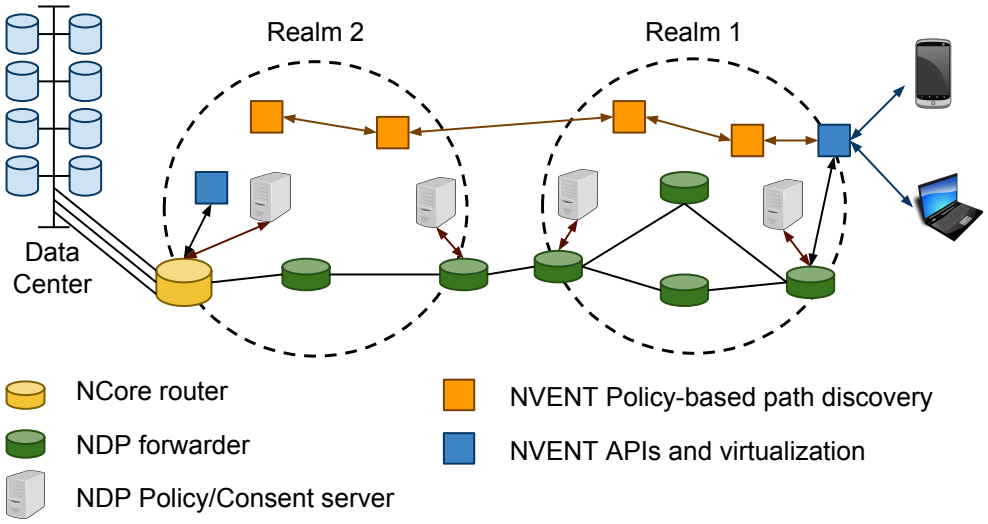


Figure 1.5: High-level view of NEBULA components integration

1.4 Information-Centric Networking

As mentioned above, Internet usage has dramatically shifted in the last decade to new applications and communication models. It is anticipated that, by 2019, almost two thirds of Internet traffic [91] will be handled by Content Distribution Networks (CDNs) [112, 158, 148]. Although such networks are deployed as IP overlays, e.g., Akamai [151], content distribution benefits can not be fully achieved. This is mainly because the network layer is not aware of packets being transmitted through the overlay network.

Information-Centric Networking (ICN) [26] is a recent paradigm for the future Internet. It emerged as an alternative to the host-based communication approach of the current IP-based Internet. While IP traffic consists of packets sent between communicating end-points,

ICN traffic is comprised of explicit requests for, and responses to, *named content objects*. A request refers to the desired content by name and is forwarded by routers (using content name) towards the content producer. Such a request is satisfied by either the producer itself or a cached copy in some router. This independence of data (content) from its location is what enables ICN routers to opportunistically cache content to satisfy future requests. The idea of in-network caching is incentivized by end-node mobility and networks with disruptions and disconnections. In such networks, nodes are not guaranteed to be continuously available. Therefore, the store-and-forward technique is used to transfer data between end-hosts. When a packet is received by a node and can not be forward due to the current lack of a route to the destination, that packet is stored by said node until it can be forwarded. In ICN, however, responses are cached in routers to satisfy future requests, thus, reducing end-to-end latency and bandwidth consumption.

Several projects have adopted the ICN paradigm. They include NDN, CCN, the Data-Oriented (and Beyond) Network Architecture (DONA) [104], and Network of Information (NetInf) [60]. We now overview DONA and NetInf while postponing CCN/NDN description to Chapter 2.

1.4.1 Data-Oriented (and Beyond) Network Architecture

DONA is a networking architecture that lives above the IP layer. It separates data and services from their physical locations. It also provides data persistence, availability, and authentication. By using principal-based networking, DONA replaces DNS domain names with flat self-certifying names, and DNS resolution with a name-based routing protocol which provides support for different types of discovery services, mobility, and data replication.

DONA principal owners can authorize other hosts to serve their own principals. For instance, data owners could host their own websites or delegate this process to a third party. Even

if the hosting location changes, the name of the data remains intact, providing mobility by design. In addition, data owners might authorize multiple third-party hosts to serve their data. In this case, DONA's name-based routing provides access to the closest replica.

Naming

Names follow the format P:L where P is the public key digest of the principal's provider and L is a label set by that provider to ensure uniqueness of the name. For instance, the cryptographic hash of a piece of data can be used as L's value of its corresponding principal. To provide data and origin authentication, data principals are digitally signed by their provider. When users request a specific piece of data with a name P:L, they can verify that the signature is valid, and data origin is authentic, i.e., P is the digest of the principal provider's public key. However, trust management is not supported by DONA's network layer and is left to the application.

Routing

DONA uses two primitives for principal reachability, Find(P:L) and Register(P:L), facilitated by Resolution Handler (RH) nodes. RHs are installed in a hierarchical fashion forming a tree-like topology. Leaf nodes are directly connected to hosts serving specific principals, while the root node contains reachability information for all nodes. Each domain should contain one logical RH that can be implemented using several collaborating physical nodes. Domain granularity in DONA can vary from AS size to a group of friends forming a small social network. Moreover, RH's behavior varies depending on the primitive:

- Find primitive: When an RH receives a Find(P:L) packet, it looks up its registration table to learn where to forward this packet downstream. If the packet can not be forwarded, RH sends the packet to its parent RH. This process repeats until the packet reaches the root RH, where it will be routed correctly down the tree branch towards its destination. In addition, RH might forward the Find(P:L) packets to peer (not

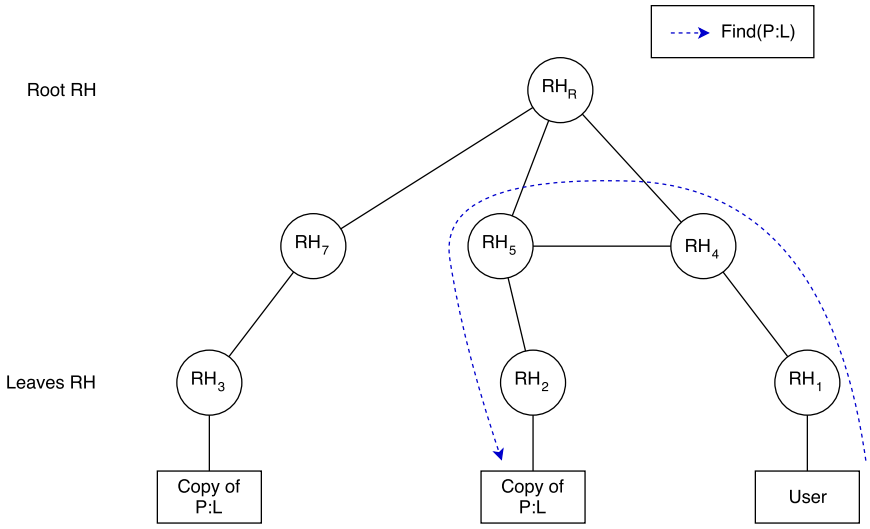


Figure 1.6: Find primitive forwarding in DONA

parent) nodes if doing so provides a shorter path to their destination. For instance, in Figure 1.6, the user sends a Find(P:L) packet to RH_1, its local RH. This packet is then forwarded towards the closest copy of P:L through RH_4, RH_5 and RH_2. Note that name matching in DONA is longest-prefix-based, where P:L is a longest prefix for P:*. Moreover, Find can also be issues for *:L, i.e., Find(*:L). In this case, the user accepts the data labeled L from any provider.

- Register primitive: If RH receives a Register(P:L) packet, the decision of forwarding it to parent and peer RHs depends on whether a similar (or a matching longest-prefix, e.g., P:*) entry exists in its registration table and local policies. The latter makes it possible to apply policy routing, similar to the Border Gateway Protocol (BGP) [87]. For instance, RHs not wishing to serve as a transport AS for their RH peers drop all Register packets received from those peers.

RHs can implement local caches. This allows them to respond to Find packets (requesting popular data) from their cache, reducing end-to-end latency and bandwidth consumption.

Security

Like any other Internet architecture, DONA raises some security concerns. Although packet

rate limitation mechanism (as DoS attacks countermeasure) is deferred to the IP layer, RHs can implement their own methods to rate-limit Find and Register requests.

Moreover, RHs establish trust relationships between users and peer RHs using public cryptography. This authenticates the origin of all received packets, and prevents adversaries from injecting bogus Find and Register packets. However, malicious RHs can still refuse to forward legitimate traffic causing denial of service. Fortunately, DONA provides users with a method of requesting data from further (not the closest) sources in order to evade such on-path malicious RHs.

Key management is deferred to the application. This subsumes revocation of compromised or expired keys. Moreover, when connecting to the network or generating new keys, providers should always ensure that their keys are not already in use. This is achieved by sending a Find(P:*) packet, expecting no response.[5]

1.4.2 Network of Information

NetInf is another ICN architecture that considers data, called Named Data Objects (NDOs), to be the primary network primitive. NetInf is designed to grow to global-scale networks such as the Internet and data centers, as well as small and infrastructure-less ad hoc networks. NetInf provides scalability, data persistency and availability, security, and supports mobility of both clients and servers.

Naming

Each NDO has a unique name called Named Information (NI). NIs identify corresponding NDOs and contain: the hash function used to calculate the name itself and the output value of this hash function, e.g., `ni:///sha-256;gh04...Lwnj9`. The semantics of the hash value depends on the NDO type. For instance, for static NDOs, the hash of the entire

[5] A Find(P:*) packets can be satisfied by any data produced by provider P.

data can be used, whereas in dynamic NDOs, the hash of the data owner's public key is used. Unlike hierarchical naming, NetInf name flatness provides better data persistency and facilitates mobility since the NDO name remains unchanged even if its owner is moving across domains. However, this comes at a higher cost of complicating route aggregation to maintain manageable-size routing tables.

Messages

Data communication is achieved using three types of messages [67]:

- `GET`: used to express a request for a specific NDO. The name of the requested NDO must be included in the `GET` message.
- `PUBLISH`: used to announce an NDO name to the rest of the network. It can optionally include a copy (or the metadata) of the NDO itself.
- `SEARCH`: includes a set of keywords to search for a specific NDO name. A response might contain one or more possible matching NDO names. `SEARCH` messages could be used to translate human-readable keywords into NIs.

Routing

NetInf routing is based on a combination of Name-Based Routing (NBR) and name resolution. NBR allows routers to forward `GET` messages based on their names and routing tables. On the other hand, name resolution enables translation of NDO names into network or host identifiers (called routing hints).[6] Such identifiers are used in making routing decision. For instance, a Name Resolution Server (NRS) can return the IP address where a specific NDO is hosted. In this case, said NDO will be requested using the same methodology utilized in today's Internet. Routing hints could also indicate a network that serves the requested NDO. Therefore, the corresponding `GET` message is forwarded to that network which, in turn, forwards the message internally.

[6]Routing hints are included in `GET` messages as part of the requested NDO names.

End-users are not the only network entities that interact with NRSs. When routers receive a `GET` message that can not be forwarded further, they can issue a resolution request to NRSs to receive routing hints. However, this operation is not mandatory and depends on local policies.

Routing NDOs is different from routing `GET` messages. This is because the latter do not contain any source addresses. NetInf does not specify a single NDO routing method. Routers can keep state of processed `GET` messages, and later use it to forward NDOs back to their requesters. Alternatively, labels can be appended to `GET` messages to serve the same purpose. Since labels are only used by routers to determine the downstream interface for forwarding NDOs, their format can be locally significant.

Although name flatness provides seamless global-scale mobility, routing (and possibly name resolution) information needs to be changed to ensure data availability. However, this only applies to mobile servers and not clients.

Security

Dannewitz et al. studied in [59] the security properties of NetInf names. Data integrity is inherently provided due to the format of NDO names. Since static NDO names contain the hash of the data itself, any network entity can verify the integrity of received NDOs. Moreover, dynamic NDOs include the public key hash of their owner, therefore, requesters can verify NDOs origin. However, NetInf does not provide any secure method for retrieving NDO names. [59] also suggests separating NDO self-certification from owner and NDO authentication. This is achieved by using two different keys: one is included in dynamic NDO names to identify the owner, and the other is used for verifying NDO's authentication, e.g., via signature.

Owner pseudonymity is also available to protect NDO owner identities. However, full anonymity can only be achieved if an unlimited number of pseudonyms can be used. NDO

owners can use their pseudonyms, instead of names, and gradually build required trust levels with their clients.

Furthermore, NetInf uses secure communication sessions. This protects against eavesdropping, packet manipulation, and replay attacks [170]. Also, to prevent the effect of malicious executable files uploaded by users, NetInf recommends the use of sandboxes to avoid issues during execution.

Caching

NetInf provides on-path and off-path caching to increase the effectiveness of content distribution and retrieval. Caches allow routers to respond to `GET` messages with stored copies of requested NDOs. Routers can redirect `GET` messages, off their path to the destination, towards a nearby cached copy. This, requires collaboration between neighboring routers to exchange NDO names they cache.

NetInf also supports peer-caching. End-devices contain local caches and inform their local NRS about caching or eviction of NDOs. Neighboring clients can contact the local NRS to learn the location of peer-cached NDOs. Then, clients send their requests to their neighbors which are satisfied from the latter caches. However, peer-caching jeopardize clients' privacy. This is because end-devices providing peer-caching can learn what other neighboring devices request.

Chapter 2

Content-Centric Networking

CCN [93] is an example of the ICN paradigm. Named content is the main network abstraction in CCN. A content name is composed of one or more variable-length components opaque to the network. Component boundaries are delimited by "/" in the usual URI-like representation. For instance, the name of a CNBCs news' homepage for February 5, 2016, might be: `/ccn/cnbc/news/02-05-2016/index.htm`. Large content can be split into intuitively named segments, e.g., chapter 16 of the Netflix movie "Groundhog Day" could be named: `/ccn/netflix/movies/groundhog_day.mp4/ch16`.

In the rest of this chapter, we overview two instances of CCN: CCNx [142, 3], a project developed at the Xerox Palo Alto Research Center (PARC), and its academic counterpart NDN [206]. Unless otherwise specified, we use the term CCN to refer to both CCNx and NDN.

2.1 CCN Elements

We now overview CCN entities and packet formats.

2.1.1 CCN Roles

CCN involves three types of entities:

- *Consumers*: end-users that request content by sending interest messages.
- *Producers*: entities that produce (publish) and disseminate content.
- *Routers*: entities that forward interests and content to/from consumers and producers.

As mentioned above, CCN uses two types of packets: interest messages and content objects. We now describe packet formats and the differences between CCNx and NDN [141, 13]. Since both architectures use different names for the same fields, for the rest of the dissertation, we use the CCNx naming terminology when referring to common header fields.

2.1.2 Content Objects

Content objects, called data packets in NDN, carry the actual application data representing content. CCN content objects are encoded using the Type-Length-Value (TLV) scheme [167]. Content header includes the following fields (we only list those relevant to our discussion):

- `Name`: content name that is being requested.
- `ContentObjectHash`: optional field that might carry hash of the content object. If not present, the hash is recomputed at every hop for verification purposes.
- `Payload`: contains an arbitrary length of binary data representing the actual content. This field is called `Content` in NDN.
- `PayloadType`: identifies the type of content, e.g., `DATA`, `KEY`, or `NACK`. This field allows the content payload to have different semantics, and is called `ContentType` in NDN.

- `ExpiryTime`: time when the payload of the corresponding content expires. In this case, such a content should not be used by producers or caches to satisfy interests. This field is called `FreshnessPeriod` in NDN.

- `Signature`: signature generated by the content producer which covers the entire object, including all explicit components of the name.

- `Signature` supporting fields: fields containing supporting information for signature verification, e.g., CCN name of the verifying public key, its digest, or the key itself. This information is carried in CCNx content header fields: `KeyId`, `PublicKey`, `Certificate` and `KeyName`, and a single field in NDN: `KeyLocator`.

Each content producer must have at least one public key, represented as named content of `PayloadType` set to `KEY` and signed by its issuer, e.g., a Certification Authority (CA).[1] The naming convention for a public key content object is to contain "key" as its last explicit component, e.g., `/ccn/netflix/movies/key`. In order for content signature to be valid (not just verifiable), the name of the public key (used to verify it) without the last explicit component must form a prefix of the content name. For instance, this holds for content named `/ccn/netflix/movies/groundhog_day.mp4/ch16` and the key named as above. However, the same is not necessarily the case for the key named: `/ccn/netflix/movies/key`, which could be verifiable by its issuer's public key with the name: `/ccn/verisign/key`.

2.1.3 Interest Messages

CCN interest message headers contain the name of the content being requested in the `Name` field. A TTL is also optionally present to specify the interest expiration time. All other fields have different format and semantics in CCNx and NDN; we present them separately.

[1] CCN is agnostic as far as trust management, aiming to accommodate peer-based, hierarchical and hybrid PKI approaches.

CCNx Interest

- `KeyId`: hash of the public key expected to verify the content signature. This field is optional.
- `ContentObjectHash`: optional hash of the requested content.
- `Payload`: optional data "pushed" by consumers to producers along with the interest.

NDN Interest

- `MinSuffixComponents` and `MaxSuffixComponents`: minimum and maximum number of name components, beyond those specified in the name, that are allowed to occur in matching content.
- `Exclude`: a list of name components that **must not** appear in the name of a returned content. This field can be used to exclude certain content by referring to its hash, which, as noted above, is considered to be an implicit last component of content name.
- `PublisherPublicKeyLocator`: either the name or the digest of the public key used to verify requested content. This field is optional.
- `Nonce`: a 32-bit random value that, along with the name, uniquely identifies the interest to allow loop detection.

An older version of CCN interest header specifications suggested the use of the `Scope` field. It specifies how far the interest message will be transmitted. `Scope` can take one of these values:

- 0 – do not propagate beyond the local CCN forwarder.
- 1 – only propagate to the application layer of the current node.

- and 2 – do not propagate beyond the next hop node. This guarantees that the interest will either be satisfied or dropped by the node that is only one hop away.

2.2 Node Components

All CCN nodes (consumers, routers and producers) maintain the following three components [93]:

- *Content Store* (CS): an optional cache. The size of this cache is determined by local resource availability. Even though content `Lifetime` field is used to suggest the duration of content to be cached, CCN nodes may unilaterally decide what content to cache and for how long. From here on, we use the terms CS and cache interchangeably.
- *Forwarding Interest Table* (FIB): routing table that maps name prefixes to outgoing interfaces. Unlike IP routing tables, FIB can specify one or more outgoing interfaces for a specific prefix.
- *Pending Interest Table* (PIT): a table of not-yet-satisfied (i.e., pending) interests, and, for each entry, a set of corresponding incoming interfaces. PIT entries might also store additional metadata.

Upon receiving an interest with name N, a router first checks its cache for a local copy of a matching content. If no local copy is found and no other interests for names matching to N are pending in the PIT, the router forwards the interest to the next hop(s) according to its FIB and forwarding strategy. For each forwarded interest, a router creates a new PIT entry with state information, including the name and the interface on which the interest arrived. If an interest with the name N arrives while there is already an entry for the same name in the PIT, the router collapses the new interest and only stores the interface on which

it was received. This process is called interest collapsing. When content is returned, the router forwards it to all incoming interfaces in the corresponding PIT entry and flushes this entry. Since no additional information is needed to deliver content, interests do not carry any source addresses.

2.3 Content Matching

As mentioned above, consumers request content objects by issuing interests containing their names. Although this operation is the same in both CCNx and NDN, the process of satisfying interests differs between the two architectures. NDN employs a longest-prefix content name matching scheme. An interest for `/icn/netflix/movies/ groundhog_day.mp4/` can be satisfied by content named `/icn/netflix/movies/groundhog_day.mp4/ch16`. However, the reverse does not hold. CCNx utilizes an exact-prefix content matching scheme, whereby interests can only be satisfied by content with the exact same name (or a hash digest).

CCN content can be requested at different granularity.

- Content names only: such interests can be satisfied by any content with a matching name (depending on the matching scheme used). This content can be published by any producer and is consider valid as long as its signature is valid.
- Content names and hash of the public key expected to verify the content signature: consumers limit the number of content objects that can satisfy corresponding interests. Only a producer possessing the corresponding private key can respond with a matching content. The public key hash is carried in CCNx's `KeyId` or NDN's `KeyLocator` field.
- Content names and hash of the requested content object: this combination forms a Self-Certifying Name (SCN). Such names limit the possible matching content to exactly

one. Content hash can be carried in interest headers as: `ContentObjectHash` in CCNx or the last component of the name in NDN. We will discuss this in more detail in Chapter 4.

Neither CCNx nor NDN mandate the use of any of the above methods. The choice is left to the application.

2.4 Routing and Forwarding

CCN routing differs for content objects and interest messages. The PIT is used to route content by following the reverse path of its corresponding interest. Interests are routed similarly to today's IP packets. The main difference is that they are forwarded based on content names rather than IP addresses. This has several advantages and drawbacks.

- Address space limitation: Unlike fixed-length IP addresses, CCN names are arbitrary long sequence of human-readable components. This solves address exhaustion without using NAT. However, unlimited address space makes maintaining large routing tables a challenge.
- Mobility support: IP-based networks do not natively support mobility. When hosts move between networks, their IP addresses must be changed to allow continuous connectivity. In CCN, content names published by mobile producers do not have to be changed, since names are unique. However, this is problematic if the producer is moving between domains. Assume a producer P is publishing content C with name `/uci/some_content`. If P moves to a neighboring domain, e.g., `/ucla/`, C's name should be changed to reflect that (e.g., `/ucla/some_content`) in order to maintain hierarchical routing. Even if P moves only within the `/uci/` domain, internal forwarding tables must be updated to reflect the change.

CCN can take advantage of hierarchical routing protocols used in today's Internet, replacing IP prefixes with name prefixes [206, 196].

IP only supports one forwarding strategy, i.e., best route strategy. In contrast, CCN FIBs can specify multiple outgoing interfaces for the same name prefix, which allows new types of multicast forwarding strategies.

- Parallel: interest is forwarded on all specified FIB interfaces at the same time and either the same or various timeout values are set for each interface. Once a matching content arrives on one of these interfaces, it is used to satisfy the corresponding interest.
- Sequential: interest is forwarded on one interface at a time. In the event of a timeout, another interface is used. This process is repeated as needed.

2.5 CCN Security and Privacy

According to NSF, one of the guiding principles for a new Internet architecture is security and privacy by design. Although CCNx and NDN offer many networking benefits, they trigger various security issues. Below we overview security and privacy features of both CCNx and NDN as compared to IP and IPsec.

2.5.1 Trust

IPsec defines trust as a one-way relationship between two or more entities (hosts or networks). This relationship is represented using a Security Association (SA). SAs contain a set of information that can be considered as a "contract" between the involved entities. Such information describes security services and contains security information needed by hosts to protect the communication.

Entities involved in secure communication in IPsec establish SAs via the ISAKMP [129] protocol and exchange necessary cryptographic material using the Internet Key Exchange (IKE) protocol [97]. Host authentication in ISAKMP and IKE can be achieved via either digital signatures, or pre-shared keys. Digital signatures require the use of certificates to bind entity identities to their public keys. This implies the existence of a CA to create, sign, and properly distribute certificates.

Unlike IP, the notion of trust in CCN is not directly associated with hosts and networks, but rather with content. Trust in content can be expressed at different levels of granularity, from a single content to an entire namespace. Recall that a content object is signed by its producer which allows anyone to verify its origin and authenticity. Origin verification refers to content producer rather than whoever stores a copy of that content. In order to authenticate a content and its origin, its signature must be verified. To do so, the verification (public) key must be retrieved and trusted.[2] However, trust management is not specified at the network layer and is left to applications. We discuss CCN network-layer trust management in Chapter 4.

2.5.2 Authentication

The IP protocol (IPv4 in particular) does not provide any form of authentication. A separate add-on method, IPsec, provides entity authentication via AH and ESP protocols.[3] In transport mode, two hosts securely negotiate a shared secret key. This key is later used to generate a Message Authentication Code (MAC) [106] for each packet. Successful MAC verification ensures authenticity of received packets and their origin. In case of gateway-to-gateway communication, gateways can only verify that the received data originated by *any* (not a specific) host connected to the network at the other end of the tunnel. In host-

[2]Keys in CCN are distributed in content objects with type `KEY`. Such objects (keys) are signed by their issuer, e.g., CA.

[3]Recall that IPv6 implements both AH and ESP as extension headers.

to-gateway communication, the gateway can actually verify that the data originated by the involved host, while the latter can only verify that received data is originated by the network located behind the gateway. This partial authentication opens the door for insider attacks.

CCN provides origin and data authentication via content signatures. Before consuming content, consumers are required to verify its signature [206]. However, this operation is optional for routers because signature verification is an expensive operation at line speed and comprehensive trust management is not viable at the network layer. Even if we assume that routers know all possible application trust models, establishing trust in content is complicated and expensive. For instance, traversing a PKI hierarchy requires routers to fetch and verify public key certificates until a trusted anchor is reached.

2.5.3 Accounting

Accounting is a means of reporting packet-related statistics. Statistics granularity differs depending on the application. For instance, a content publisher might only be interested in learning the number of requests users issue for each content, whereas a medical records provider might be required, by law, the collect information about the actual users accessing its records.

IP allows routers and hosts to collect various packet- and communication-related information. This is because every IP router has access to source and destination addresses of each packet. IPsec does not inhibit accounting when used in transport mode since the IP header is unchanged. However, tunnel mode IPsec does not preserve original source and destination addresses.

Accounting in CCN can only be done at the level of content. Network entities inspecting interest and content names can keep track of request frequency and volume. Unfortunately,

in-network caching prompts accounting challenges. Since interests can be satisfied by any router, it is hard for producers to collect accurate and timely information about content requests and cache hits. This is discussed in Chapter 5.

2.5.4 Data Confidentiality

The natural way to achieve data confidentiality at the network layer is by using encryption.

IP does not provide data confidentiality. This is done by using IPsec. The level of confidentiality depends on the mode of operation. In transport mode, ESP only encrypts the IP packet payload and data confidentiality is host-to-host. Tunnel mode extends confidentiality to the entire encapsulated IP packet, including both payload and header. However, data confidentiality can only be achieved in host-to-gateway or gateway-to-gateway scenarios. Also, ESP confidentiality is not generally effective against active adversaries. It has been demonstrated that achieving confidentiality without a strong integrity mechanism, or even applying integrity before encryption, can only protect against passive adversaries [39, 105, 63]. Thus, even though IPsec provides confidentiality, poor usage practices can negate its benefits.

Data confidentiality in CCN can be attained by encrypting content payload. However, this is not supported by CCNx and NDN architectures and is left to the application.

2.5.5 Traffic Flow Confidentiality

It is well known that encryption does not protect against statistical traffic analysis – attacks that monitor traffic in order to extract properties, such as volume and timing [168].

IPsec provides some traffic flow confidentiality by padding packet payloads to hide their size patterns. However, according to IPsec specifications, this is not mandatory and, therefore, may not be supported in all IPsec implementations.

Both CCNx and NDN are susceptible to traffic analysis. Fortunately, padding can be used to provide traffic and flow confidentiality.

Another architecture-agnostic alternative is to add artificial delays to communications to better hinder time-based attacks. This, however, comes at the expense of increasing end-to-end latency and reducing overall network performance, especially for time-sensitive traffic.

2.5.6 Privacy and Anonymity

IP (with or without IPsec) does not support anonymous communication. This is mainly because source and destination addresses are in the clear in packet headers. However, partial anonymity can be achieved using the tunnel mode of IPsec along with ESP. This is because tunnel mode allows the ESP protocol to encrypt the original IP packet along with the source and destination addresses, and it encapsulates that packet into a new one with a new header reflecting gateway addresses. This combination hides end-host identities among the set of other hosts connected to respective end-networks. However, this is only effective if the adversary is eavesdropping on the link between the two gateways and is not located inside one of the end-networks. Furthermore, in case of host-to-gateway tunnel mode, only anonymity of hosts located behind the gateway is preserved.

Crowds [169] is one of the first proposals to achieve user anonymity. In it, a message is randomly forwarded between group members before it reaches its destination. Therefore, none of the group members nor the end recipient learn the actual source of the message. The Onion Router (TOR) [189] is another method that provides anonymous communication

through a "circuit." Circuits are multi-hop encrypted communication channels established using at least three TOR nodes. Theoretically, TOR guarantees anonymity with respect to an adversary controlling, at most, two TOR nodes. However, flawed TOR implementations can reduce its provided anonymity level [44].

Unlike IP, CCN has some features that facilitate anonymous communication. A PIT allows interest messages and content objects to only carry the requested content name without any consumer-related information. However, Compagno et al. show in [53] that adversaries with enough knowledge of the network can determine consumer's location. DiBenedetto et al. proposed ANDāNA [65], a tool that provides a level of anonymity similar to TOR, while requiring only two intermediate nodes, instead of three.

2.6 Attacks on CCN

Most network-layer attacks on CCN fall into the general domain of DoS or Distributed DoS (DDoS). Some key CCN features, e.g., caching, trigger new types of DoS/DDoS attacks. Gasti et al. [77] presented a range of DoS/DDoS attacks in NDN. In this section, we describe some prominent attack types and discuss some countermeasures proposed in the literature.

2.6.1 Interest Flooding

As mentioned above, a PIT is one of the main router components that enables content delivering without requiring any form of consumer source addresses. A PIT is also used for interest collapsing in order to reduce bandwidth due to a burst of closely-spaced interests for the same content. However, the fact that a PIT is a limited and valuable resource makes it susceptible to malicious exhaustion. Adversaries can send a large number of interests attempting to fill the PIT. To avoid collapsing, such interests can refer to *nonsensical* content.

Once the PIT is full, a router can either: (1) drop incoming interests, or (2) remove old PIT entries to make space for new ones. Both options, however, can adversely impact past or future interests. This type of attack is called Interest Flooding (IF).

Unfortunately, there is no comprehensive remedy for IF attacks. Although several countermeasures have been proposed, they are ineffective against smart adversaries and only manage to lower the volume of IF attacks [54, 22]. One possible remedy for IF attacks is to eliminate the PIT – its root cause. For this reason, [78] suggests a modified CCN architecture without router PITs.

2.6.2 Cache Privacy

In-network caching opens the door for side-channel privacy attacks targeting producers, consumers or both. Measuring the delay to deliver content allows adversaries to determine whether said content was satisfied by a nearby cache or by its producer. This reveals the fact that other consumers have recently requested the same content, jeopardizing privacy of these consumers.

Moreover, the combination of NDN longest-prefix matching on content names, and the `Exclude` and the old `Scope` fields could be abused for the purpose of malicious cache *harvesting*. Suppose that an adversary Adv sends an interest to router R with the name `/` and `Scope` set to 2. This ensures that R will satisfy the interest from its cache with a content object C_1 with name prefixed `/`. Since this criteria holds for *any* cached content, R will always reply from its cache with some content object. Adv can then send another interest with the name `/` and `Scope` of 2 which excludes C_1. This causes R to respond with another content from its cache $C_2 \neq C_1$. Adv can easily repeat this process indefinitely to receive all content in R's cache and, as a result, *harvest* that cache without doing any timing measurements.

Fortunately, the `Scope` field is discarded in the newest NDN's interest header. Therefore, we do not discuss harvesting further and only focus on time-based attacks in Chapter 3.

2.6.3 Content Poisoning

Since routers may cache content they receive without being forced to verify its authenticity, so-called content poisoning attacks become possible. In such attacks, adversaries inject fake content into router caches. This poisoned content can be used to satisfy future interests and cause a denial of service. We discuss this further in Chapter 4.

2.6.4 Cache Pollution

In cache pollution attacks, adversaries attempt to manipulate reference locality of caches, causing incorrect decisions by cache eviction strategies. This causes routers to possibly evict popular content, reducing overall content distribution efficacy.

Conti et al. discuss this attack in [57]. It is shown that with even limited adversarial resources, a highly effective cache pollution attack can be mounted. In fact, even small cache locality manipulation can cause a significant content distribution disruption [64]. It is also shown in [57] that launching pollution attacks on large networks is relatively easy, and smart adversaries reduce the effectiveness of proposed countermeasures.

Chapter 3

Cache Privacy

Despite its obvious benefits, content caching in CCN raises a major privacy issue that we summarize below and then discuss in more detail in Section 3.1. Suppose an adversary Adv wants to determine whether a consumer (Alice) recently requested certain content C. Assume Adv and Alice share a first-hop CCN router R, e.g., they are neighbors served by the same ISP, and Adv can measure the round-trip time (RTT) to R (Adv $\leftrightarrow$ R). Adv issues an interest for C and measures the corresponding RTT. By comparing the expected and actual RTTs after retrieving C, Adv can determine whether C was retrieved from R's cache.[1] Specifically, if the two RTTs are approximately equal, Adv can conclude that C must have been served by R.

Similarly, suppose that Adv wants to learn whether a content producer (Bob) has been recently asked for, and subsequently distributed, C. Assuming that Adv and Bob are separated by at least one router, Adv measures or estimates the RTT between itself and Bob and then requests C. If the latter RTT is lower than the former, Adv concludes that C was fetched from some CCN router cache and, therefore, at least one consumer recently

[1] Clearly, there might be other users besides Alice and Adv. However, this would not change the nature of the attack.

requested it. Furthermore, a combination of these two attacks can be used to learn whether two parties (Alice and Bob) have been recently, or still are, involved in two-way interactive communication, e.g., voice communication or SSH.

These attacks do not require Adv to have any special privileges: the interaction between Adv and routers is normal. It might appear that Adv can learn the same information by simply eavesdropping on Alice or Bob. However, eavesdropping requires real-time presence by Adv who must also be physically near the victim (e.g., the same Ethernet segment, wired or wireless), whereas aforementioned attacks require neither real-time presence nor proximity. Moreover, using encrypted links between consumers and their first-hop routers obviates Adv's need to eavesdrop.

At first glance, CCN seems inherently safe against cache privacy attacks: if $k > 1$ consumers share the same router's cache, Adv can not determine exactly *which* or *how many* requested particular content (if the content does not expose consumer-identifying information). Thus, consumers seem to benefit from some form of k-anonymity. However, for many types of traffic, e.g., email, instant messaging, and voice, consumer identities can be determined from the content or its name. Moreover, k-anonymity may be insufficient if Adv possesses any auxiliary information about neighboring consumers, or it simply wants to determine whether *any* consumer requested a particular piece of content.

Note that content encryption is not sufficient to mitigate these attacks. Encryption conceals the payload of a content object and, optionally, part of its name. However, content names can not be fully encrypted, since doing so would prevent content objects from being routed.

Based on the above discussion, there is an inherent conflict between utility of in-network caching and privacy for consumers and producers. This chapter focuses on the resulting privacy problem and potential countermeasures. Our contributions are:

- Identification of root causes of the cache privacy attack.
- Countermeasures based on routers simulating consumer-to-producer RTT.
- Optimizations to increase private content distribution performance.

3.1 Cache Privacy Attacks

In this section we describe in more detail the aforementioned cache privacy attacks. Our goal is to show that Adv can learn whether a specific content C was recently requested by a consumer Cr by probing their shared first-hop router's (R) cache. To do so, Adv issues an interest for C and measures the delay τ_1 required to retrieve it. It then picks another (existing) content C' and requests it twice in succession. The first time, C' might be fetched from its original producer P or from R's cache. However, the second time, C' is certainly fetched from R's cache. Let τ_2 denote the retrieval delay for the latter. If $\tau_1 \approx \tau_2$, Adv concludes that Cr recently requested C. Otherwise, if $\tau_1 > \tau_2$, Adv decides that C has not been recently requested by anyone from its side of R.[2]

To illustrate the ease of carrying out these timing attacks, we ran some experiments using the topology shown in Figure 3.1. It includes: (1) a consumer Cr, (2) a router R, (3) a producer P, and (4) an adversary Adv. P is reachable by Cr and Adv only through R. We also assume that only Cr and Adv are served by R as their fist-hop router, though we will later relax this assumption. In all of our experiments, P publishes 1000 content objects with different names. Also, the type and characteristics of the links connecting Cr, Adv, and P to R differ based on each experiment setup. We describe the detailed setup and experiment outcomes below.

[2]Note that $\tau_1 < \tau_2$ is not possible, since we assume that R is the first-hop router for both Cr and Adv.

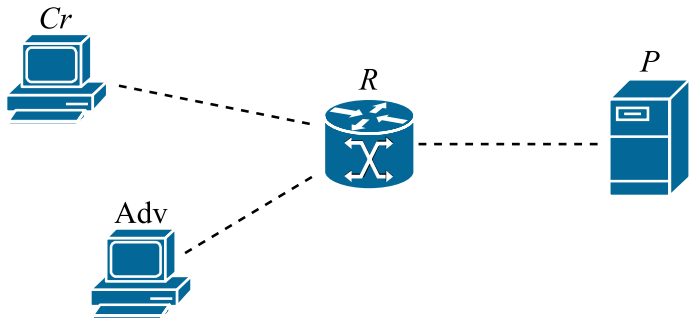


Figure 3.1: Cache privacy attack experimental setup

3.1.1 Consumer Privacy in LAN Environment

In this environment, Cr, Adv and P are all connected to R via Ethernet. Cr starts by requesting all published content objects, causing them to be cached by R. Then, Adv requests the same content objects which are promptly fetched from R's cache. We measure average delays for retrieving content from P and R. The results, illustrated in Figure 3.4(a), show the Probability Distribution Function (PDF) for these delays. It is clear that Adv can determine, with probability over 0.99 whether C is retrieved from R's cache.

3.1.2 Consumer Privacy in WAN Environment

We also run similar experiments in a WAN topology using the NDN testbed [14]. In this case, Cr and Adv both connect to the same first-hop router R' via Ethernet. R' is 3 hops away from R via a WAN link. Similarly, P is also 3 hops away from R over WAN. Figure 3.2 shows the setup of this experiment and Figure 3.4(b) shows the results. Clearly, the presence of additional hops increases delay and introduces some variance. However, Adv can still determine whether C is retrieved from R's cache with probability over 0.99.

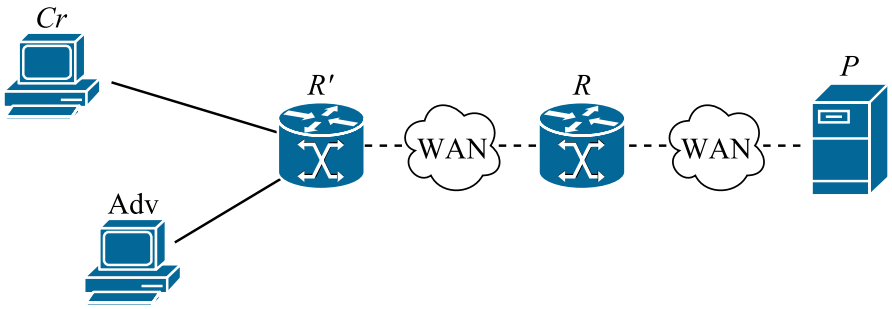


Figure 3.2: Consumer privacy in WAN environment topology

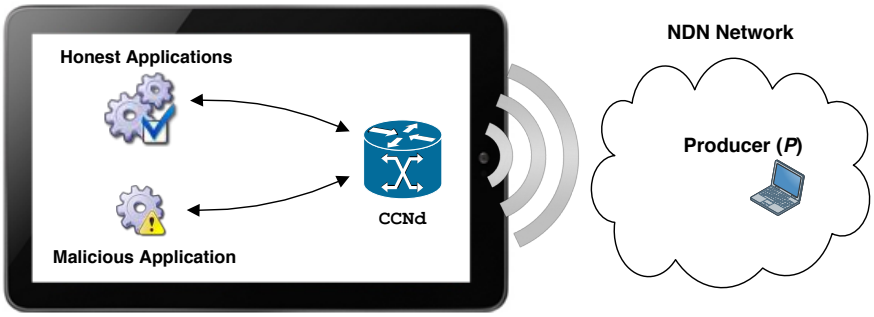


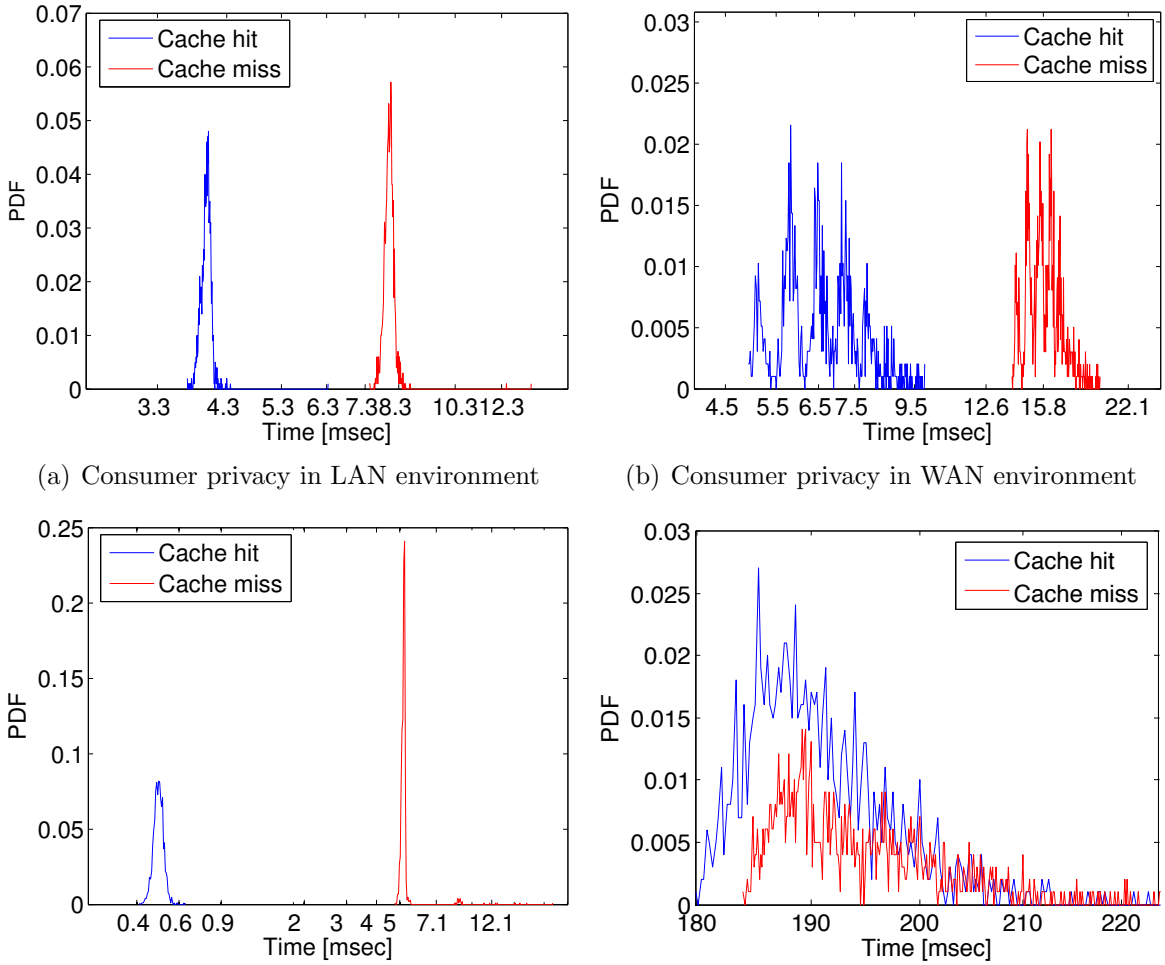
Figure 3.3: Consumer privacy in local environment topology

3.1.3 Consumer Privacy in Local Environment

The two attacks described thus far (in LAN and WAN environments) are also applicable to the local cache of a specific CCN node, e.g., a laptop or Android smartphone [1], which is shared among several applications, as shown in Figure 3.3. A malicious application can employ the same probing techniques described above. Figure 3.4(c) summarizes the results obtained in the localhost settings. The difference between cache hits and cache misses is even more evident than in previous experiments; Adv easily learns information about cached content requested by other applications.

3.1.4 Producer Privacy in WAN Environment

We now turn to producer's privacy. We consider the topology shown in Figure 3.5, where P is directly connected to R, while Cr and Adv are three hops away over WAN. Adv's



(a) Consumer privacy in LAN environment

(b) Consumer privacy in WAN environment

(c) Consumer privacy in local environment

(d) Producer privacy in WAN environment

Figure 3.4: Timing attack results

goal is to determine whether C was recently served by P and, as a result, is cached by R. Figure 3.4(d) shows that Adv can distinguish whether C is served from R with over 0.59 probability by probing a single content object. However, large content is typically split into several (smaller) pieces, called segments, and transmitted as multiple content objects. The correlation between such objects can be exploited to improve Adv's cache privacy attack accuracy. Adv only needs to determine whether *one* of the correlated content objects has been served by P.

Let **success** denote the event where Adv successfully determines whether a single content object is fetched from the cache, and let **fail** denote failure to do so. Since **fail** and **success**

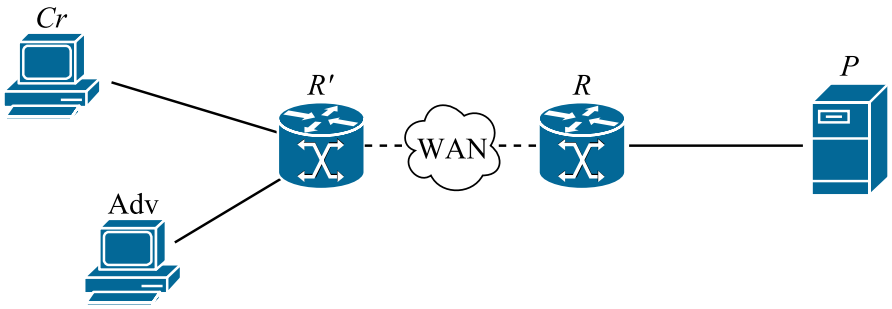


Figure 3.5: Producer privacy in WAN environment topology

are independent for all correlated content objects, overall probability of failure (**FAIL**) can be expressed as

$$\Pr[\textbf{FAIL}] = (\Pr[\textbf{fail}])^n$$

And, similarly,

$$\Pr[\textbf{success}] = 1 - (\Pr[\textbf{fail}])^n$$

In the above experiment, $\Pr[\textbf{success}] = 0.59$ and $\Pr[\textbf{fail}] = 0.41$. Thus, if a large content is split into eight content objects, then the probability of a successful cache privacy attack can be increased to almost 1.

$$\Pr[\textbf{SUCCESS}] = 1 - 0.41^8 \approx 0.999$$

3.2 System, Adversary and Privacy Model

In this section we introduce our system, privacy, and adversary models.

3.2.1 System Model

We assume the following system model. Let $\mathbb{N}$ and $\mathbb{C}$ denote the universes of all names and content objects, respectively. As before, let R be a router. The internal state of R is

represented by a function $S : \mathbb{C} \to \mathbb{Z}^+$ that, for a given content, represents the number of times it has been forwarded. $S(C) = 0$ for all C not in R's cache. We assume that C can appear in R's cache only if it has been previously forwarded by R.

We define a cache management algorithm $\mathcal{CM}$ that uses R's internal state to determine what content forwarded by R needs to be cached. $\mathcal{CM}$ also controls how R responds to interests that correspond to cached content.[3] Without loss of generality, we assume that consumers have access to content only through R, i.e., R is their only choice as a first-hop router. We make no assumptions about how $\mathcal{CM}$ responds to interests that match content in its cache, e.g., $\mathcal{CM}$ is free to ignore the cache altogether for some incoming interests. Finally, we assume that $\mathcal{CM}$ can hide cache hits (e.g., by simply not using its cache) but can not hide cache misses.

By interacting with R, consumers (and adversaries) are allowed to determine, with some probability, whether a specific content has been forwarded by R using probing attacks. We model this by a probabilistic algorithm $Q_S : \mathbb{N} \to \{0, 1\}$ with access to the router's internal state S. Q_S outputs 1 if cached content C matches the input name. Otherwise, Q_S outputs 0. After each invocation of Q_S, S transitions to S' such that $S'(C) = S(C) + 1$ and, for all other $C' \neq C$, $S'(C') = S(C')$. In other words, $S(C)$ indicates the number of times C has been served from R's cache.

3.2.2 Adversary Model

The goal of Adv is to learn information about content forwarded by, and likely still cached in, R. Since $\mathcal{CM}$ is not secret, Adv can use Q_S to learn private information. In particular, Adv can test whether C has been recently forwarded by querying $Q_S(n)$, for any $n \in \mathbb{N}$.

[3]The need for this will become apparent in Section 3.6.

In our adversary model, Adv can be any CCN entity that requests and receives content. Adv is not allowed to compromise any honest (intended victim) consumers or R. Also, Adv can not eavesdrop on communication between R and honest consumers. This restriction can be enforced in practice by using an encrypted channel between each consumer and its closest router.

Note that this model does not assume knowledge of any consumer adjacent to R. However, Adv's success in learning what content is cached in R can be used to help identify these consumers.

3.2.3 Privacy Model

We now turn to privacy definitions. We take advantage of the concept of (ε, δ)-probabilistic indistinguishability [83, 123] – a standard notion to measure indistinguishability of two distributions in privacy-oriented applications.

Definition 3.1 ((ε, δ)-probabilistic indistinguishability)**.** *Two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are (ε, δ)-probabilistically indistinguishable, if we can divide the output space $\Omega = Range(\mathcal{D}_1) \cup Range(\mathcal{D}_2)$ into Ω_1 and Ω_2 such that:*

- *for all $O \in \Omega_1$, $e^{-\varepsilon} \leq \frac{\Pr(\mathcal{D}_1=O)}{\Pr(\mathcal{D}_2=O)} \leq e^{\varepsilon}$*
- $\Pr(\mathcal{D}_1 \in \Omega_2) + \Pr(\mathcal{D}_2 \in \Omega_2) \leq \delta$

Two distributions are "close" if both ε and δ are small. This definition is stronger than the widely used *statistical indistinguishability* since it requires similar probabilities for *each* output in Ω_1. Ω_2 contains all "bad" outputs with probabilities in $\mathcal{D}_1$ and $\mathcal{D}_2$ that differ substantially; their ratios can not be bounded by e^{ε} or even $e^{-\varepsilon}$. Intuitively, if $\mathcal{D}_1$ and $\mathcal{D}_2$ represent output distributions of $\mathcal{CM}$ with two different states, then (ε, δ) measures the

information that $\mathcal{CM}$ leaks about those states. Any output from Ω_2 typically leaks "too much" information, e.g., occurrence of any $O \in \Omega_2$ such that $\Pr(Q_{S_1}(n) = O) > 0$ and $\Pr(Q_{S_2}(n) = O) = 0$, for the same name n and $S_1 \neq S_2$, may result in a privacy breach in practice, as S_1 and S_2 become distinguishable through $\mathcal{CM}$ in that case.

We now define *perfect privacy* with respect to forwarded content. Informally, $\mathcal{CM}$ provides perfect privacy if the way it responds to Q_S queries does not yield any information about the content of R's cache.

Definition 3.2 (Perfect privacy)**.** *For all names $\ell \in \mathbb{N}$, subset of content $M \subset \mathbb{C}$, and pairs of states S_0, S_1 such that $S_0(x) = S_1(x)$ for all $x \in \mathbb{C} \setminus M$ and $S_0(y) \neq S_1(y)$ for all $y \in M$, $Q_{S_0}(n)$ and $Q_{S_1}(n)$ are $(0,0)$-probabilistically indistinguishable.*

The above definition is strong since it implies that no information is revealed about any content previously forwarded by R if $\mathcal{CM}$ offers perfect privacy. We believe that this level of privacy may not be necessary in practice. For this reason, we use the concept of content popularity to relax the above definition. Specifically, there is no need to conceal the presence of popular content objects (e.g., Google's home page) in router caches since Adv can safely assume that these objects are cached *without* probing them. To this end, let k be the number of requests after which a content is considered *popular*. We allow the distributions of Q outputs under two states S_0 and S_1 to be non-indistinguishable with some probability that depends on k.

Definition 3.3 ((k, ε, δ)-privacy)**.** *For all names $n \in \mathbb{N}$, subset of content $M \subset \mathbb{C}$, and pairs of states S_0, S_1 such that $S_0(x) = S_1(x)$ for all $x \in \mathbb{C} \setminus M$ as well as $S_0(y) = 0$ and $0 < S_1(y) \leq k$ for all $y \in M$ (i.e., S_0 and S_1 only differ on content objects in M); $Q_{S_0}(n)$ and $Q_{S_1}(n)$ are (ε, δ)-probabilistically indistinguishable.*

3.3 Which Content is Private?

When a content object is considered or requested as private, the network should provide (at least) (k, ε, δ)-privacy based on Definition 3.3. However, since not all content is private, the question remains: how does the network determine which content is private? Unfortunately, there is no universal policy. All three CCN entities (consumers, routers, and producers) must individually or collectively participate in this decision. In the rest of this section, we present advantages and drawbacks of individual privacy decisions. We also describe the involvement of each entity. We then discuss privacy decision techniques that require collaboration between network entities.

3.3.1 Router-Driven

Since cache management is part of a normal router's standard operating procedure, it might seem obvious that this router involvement in protecting cache privacy is required. However, as we show below, it is possible to achieve privacy without network (router) awareness. Routers solely deciding what content is private is problematic. This is because routers do not have the means to differentiate between requested content. Any attempt in doing so increases overhead and negatively impacts network throughput. Even if routers do not attempt to differentiate between requested content, achieving cache privacy requires treating all content as private, hence providing unnecessary perfect privacy as per Definition 3.2. This adds complexity and overhead to router operations, thus inhibiting efficient content distribution.

However, consumer-facing routers can provide privacy as a service for consumers. When consumers connect to the network, they can specify whether their traffic should be treated as private. For instance, Alice might want all her content requests to be considered private

when connecting to a public Wi-Fi access point (AP) in a coffee shop, which might not be the same policy when connecting to the AP in her home. Although not all consumer traffic is considered private, all of Alice's traffic would be treated as such. Unfortunately, this approach suffers from the "selfish consumer" phenomena, whereby some consumers always demand privacy by default without regards for the overall network performance.

3.3.2 Consumer-Driven

In this case, consumers are responsible for specifying what content is private. Both consumer and router involvement is required since the latter performs all cache management operations.

Communicating what content is private between consumers and routers can be done using one of the following (non-exclusive) methods:

1. Interest messages can carry a `NO-Cache` flag. When set, routers do not cache corresponding content objects. This clearly protects consumer privacy.

2. Interest messages can carry a `Privacy` bit. When set, corresponding content objects are marked accordingly and treated as private when cached by routers. The producer may or may not honor this `Privacy` bit, though.

The advantages of consumer-driven privacy decisions are: (1) finer granularity than routers treating all (or per-consumer) traffic as private, and (2) consumers know in advance whether requested content is private. However, this technique also suffers from the selfish consumer phenomena, e.g., consumers might always set `Privacy` bit in all interests.

3.3.3 Producer-Driven

In this case, producers specify which content is private. This approach requires producer and router involvement. Communicating content privacy to the network can be done using one of the methods mentioned above, i.e., content object header can include either a `NO-Cache` flag or a `Privacy` bit. A drawback of this technique is that consumers do not know in advance whether requested content will be treated as private.

One difference between producer-driven and consumer-driven methods is that the former does not suffer from the "selfish producer" phenomena. Selfish behavior by producers is nonsensical since it defeats producers' objective of efficient content delivery.

3.3.4 Collaborative Privacy Decisions

It is possible for both consumers and producers to determine the privacy level of a certain content. In some cases, the consumer might set the `Privacy` bit in an interest while the producer does not set that bit in the corresponding content C. All routers caching this content should respect the `Privacy` bit, i.e., C is considered private if it is either requested or served as such.[4]

Another alternative is to use unpredictable (secret) names for content. In other words, if C is private, both consumers and producers refer to it by a name that contains a random and a hard-to-guess component, ideally derived from some shared secret. One disadvantage of this approach is that both parties need to be involved and *a priori* agree on the random name component. On the other hand, an important advantage of this approach is its opaqueness since routers need not be involved. As discussed later, this approach is useful, and in fact recommended, for interactive traffic.

[4]We will show in Section 3.4.4 that it is possible not to consider C private even if it is requested or served as such.

3.4 Countermeasures

One trivial and effective countermeasure to all aforementioned attacks is to simply disable router caching altogether. However, this would immediately worsen the performance of content distribution, one of the key features of CCN. Another alternative would be to provide perfect privacy per Definition 3.2. Nevertheless, it is clear that this would result in significant performance degradation, especially since not all content is private. Section 3.3 presented some possible mechanisms that can be used to mark content as private. They are not mutually exclusive, e.g., even if C is not marked as private by its producer, it can be requested as private by a consumer. However, Section 3.3 did not specify any actions that need to be taken by CCN entities (particularly, routers) upon encountering private content. In the rest of this section we introduce techniques that inhibit Adv from extracting meaningful information about private content from router caches. These techniques provide various trade-offs between privacy and performance.

In designing countermeasures, we consider two types of network traffic: *interactive* and *content distribution*. The first represents synchronous communication between two or more parties, e.g., voice/video conferencing and remote shell. This type of traffic is characterized by low-latency and continuous interaction, i.e., communicating parties continuously play the roles of both producer and consumer. Conversely, multimedia data delivery, live broadcasts, and delivery of web pages are examples of content distribution traffic. Our rationale for distinguishing between these two traffic types is discussed below.

3.4.1 Interactive Traffic

While in-network caching mostly benefits content distribution, it also helps mitigate packet loss in interactive communication [206]. This is because interests re-issued for lost packets

can usually be satisfied by content cached closest to the location of the actual loss, thereby reducing the delay for re-requested content. For this reason, any privacy-enhancing caching mechanism for this class of traffic should not introduce additional delay.

At the same time, since interactive content tends to be time-sensitive, there are hardly any benefits from caching it in routers in the longer term. For instance, if several users take part in a video-conference, cached stale video frames are of no use to any of them after a short amount of time.

We choose to protect this class of traffic using unpredictable names, as described in Section 3.3.4. Consumers and producers use a random value *rand* as the last component of the name of each content they request and serve, respectively. This requires some coordination between the two (or more) parties involved in the interaction. Without loss of generality, the parties need to agree on a shared secret for seeding a pseudo-random function (e.g., a keyed cryptographic hash such as HMAC [106]) used to generate content-name-specific suffix *rand*.

We take advantage of our previous assumption that Adv can not eavesdrop on consumers or producers involved in interactive communication or on traffic over R's incident links (e.g., due to link encryption or lack of physical access). Unpredictable content names inhibit malicious probing of R's cache. However, NDN routers must not respond with content that include *rand* as a name component to interests that do not explicitly express it. For example, content named `/alice/skype/0/`*rand* should not be returned to interests for `/alice/skype/` even though it would be a longest-prefix match. This is not an issue in CCNx since the architecture uses exact content name matching. Moreover, in the event of packet loss, a consumer can re-issue an interest and still benefit from obtaining requested content from the router closest to the location of this loss.

3.4.2 Content Distribution Traffic

Unlike interactive traffic, content distribution does not require any coordination between producers and consumers. Also, it benefits a lot more from longer term router caching. Consequently, an ideal privacy approach for content distribution traffic would retain at least some benefits of caching beyond simple packet loss recovery.

In this setting, neither the consumers nor the timings of their requests are known in advance by the producer. Thus, using unpredictable content names, which seems well-suited for interactive traffic, is not viable for content distribution. More generally, all techniques described in Section 3.3.4 are not applicable, leaving only router-, producer-, or consumer-driven means of marking private traffic. In order to maximize flexibility, we allow any combination thereof.

Based on our discussion thus far, it appears that router involvement in handling private traffic is unavoidable for content distribution. Since low latency is not usually a primary requirement for this kind of traffic, routers can hide cache hits by introducing artificial delays before responding with privacy-sensitive cached content. Although this strategy increases end-to-end latency, it retains one important benefit of caching: reducing congestion and better bandwidth utilization. Moreover, if the overall delay introduced by routers is close to the RTT between Cr and P, the behavior of the network from Cr's perspective becomes similar to that of the current IP-based Internet.

We now need to consider *which* routers should introduce artificial delays. Since the objective of this countermeasure is to hide cache hits for privacy-sensitive content, it makes sense that only caching routers should introduce artificial delays. An interest might generate a cache hit in, *at most*, one router on the path between the consumer and the producer. Therefore, responding to interests should be delayed by, *at most*, one router – the router which satisfies the interest from its cache.

3.4.3 Artificial Delay Properties

Suppose that we introduce a constant delay γ such that, in case of a cache hit, R waits for γ before returning privacy-sensitive content. This procedure is shown in Algorithm 1. In case of a cache miss, the artificial delay at R must be the difference between γ and the actual delay for R to receive the requested content. Note that, in the latter case, the overall delay between the interest and content arrival times would still be γ.

Algorithm 1 Privacy-Aware-Forwarding

```
1: Input: Interest for C, privacy bit b
2: if C ∉ CS then
3:     Forward interest based on local forwarding strategy
4: else
5:     if b = 1 then
6:         Wait for some artificial delay based on C
7:     end if
8:     Forward C to downstream interface
9: end if
```

This approach is easy to implement and requires very little additional per-cache-entry state. However, it has a major drawback in that it either penalizes nearby content or sacrifices privacy for far-away content. The former happens if γ is set too high and content with nearby (with respect to R) consumers becomes unduly delayed. Whereas, the latter occurs when requested content is far away (or routed via slow and/or congested links) and the actual delay at R exceeds γ.

It is unclear how to determine the optimal value of γ that would avoid both problems. Therefore, we consider two alternatives:

- *Content-specific delay:* For each privacy-sensitive content C, R stores the original interest-in $\rightarrow$ content-out delay, γ_C. In other words, γ_C is the time it took R to obtain C, from either its producer or some other router's cache, the first time. If an interest for it arrives while C is in R's cache, R delays replying by γ_C.

- *Dynamic delay:* A router dynamically adjusts artificial delay to mimic current behavior of in-network caching for popular content. As the number of interests for a given content grows, so does the likelihood of it being cached at a nearby router. According to Definition 3.2, artificial delay must not drop below the actual delay for content located two hops from Adv. We describe this type of delay in Section 3.6.

The former is obviously the safer choice for privacy even though it imposes considerable delays for popular content that was originally fetched from far-away producers or routers and then cached at closer locations. Conversely, dynamic delay is more responsive to ephemeral traffic patterns at the cost of requiring routers to constantly monitor delay and popularity for all content.

In order for content to be perceived as satisfied by its producer, all routers on the path should take cache privacy into consideration. Consider the topology in Figure 3.6, which contains two consumer Cr_1 and Cr_2, two routers R_1 and R_2, and a producer P. Assume that both routers start with empty caches and C is a content object published by P. Propagation delays of the links in the network are shown in the same figure.

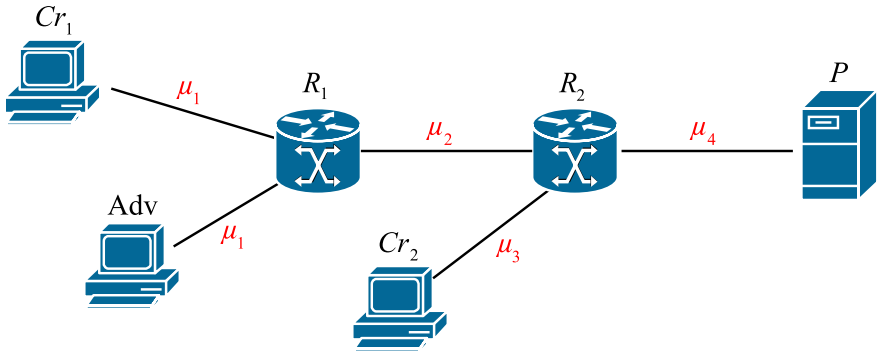


Figure 3.6: Artificial delay length

We have the following three cases:

1. Both routers introduce artificial delays when privacy-sensitive content is requested. If Cr_1 requests C, the latter is cached at both R_1 and R_2. Both routers also store the

RTT to retrieve C from P. R_2 stores $2 \times \mu_4$, and R_1 stores $2 \times (\mu_2 + \mu_4)$. If Adv requests C, R_1 delays with already observed RTT $= 2 \times (\mu_2 + \mu_4)$. In other words, RTT experienced by Adv would be $2 \times (\mu_1 + \mu_2 + \mu_4)$, which is the RTT to fetch C from P.

2. Both routers introduce artificial delays. If Cr_2 requests C, R_2 sets its retrieval RTT as $2 \times \mu_4$. When Cr_1 requests C, R_2 adds an artificial delay of $2 \times \mu_4$ before responding from its cache. This causes R_1 to store C retrieval RTT as $2 \times (\mu_2 + \mu_4)$ when caching C. If Adv requests C, R_1 delays $2 \times (\mu_2 + \mu_4)$ causing Adv to experience RTT of $2 \times (\mu_1 + \mu_2 + \mu_4)$, which is also RTT to fetch C from P.

3. Only R_1 is concerned about cache privacy, i.e., R_2 does not introduce any artificial delay when cache hits occur. If Cr_2 requests C then this content gets cached in R_2. If Cr_1 requests C, R_1 caches it and stores its retrieval RTT $= 2 \times \mu_2$. R_1 does not know that the request is satisfied from R_2's cache. Therefore, if Adv requests C, it experiences RTT of $= 2 \times (\mu_1 + \mu_2)$ due to R_1's introduced delay.

In Cases 1 and 2, all interests issued for C by Adv are perceived to be satisfied by P due to artificial delay introduced by both routers. However, this is not the case if at least one router on the path treats all content uniformly, regardless of whether it is privacy-sensitive or not, e.g., in Case 3. We thus conclude that if a router caches content then it should also introduce artificial delays. This requirement is stronger for edge routers and can be relaxed in the core. This is because network core routers serve a large amount of consumers which significantly increases the size of the anonymity set.

3.4.4 Artificial Delay Exceptions

We now consider how a router should handle all possible combinations of consumer- and producer-driven content marking. As mentioned above, if C is marked as private by its

producer *or* any consumer, this must be always honored by routers as long as C is cached. However, this rule has exceptions, especially when specific content is consecutively requested as private and non-private by different (or even the same) consumers while cached by R.

To demonstrate this, we assume the same topology as in Figure 3.6, both routers are privacy-aware and start with empty caches.

1. If Cr_1 requests C for the first time with the interest privacy bit set, the content will be delivered from P and cached by R_1. Cr_1 experiences $2 \times (\mu_1 + \mu_2 + \mu_4)$ RTT delay. If Adv then requests C twice as non-private, it experiences the same (artificial) RTT delay of $2 \times (\mu_1 + \mu_2 + \mu_4)$ in both previous requests. This reveals the fact that a consumer previously requested C as private.

2. If Adv requests C as non-private, the content is served by P and cached in R_1. First, after Adv's request, Cr_1 requests the same content as private, triggering R_1 to treat it as such in all subsequent requests. Also, stored delay will be $2 \times (\mu_2 + \mu_4)$ since that was the value recorded by R_1 when C was originally requested by Adv. Then, Adv requests C again as non-private. Adv sees an RTT delay of $2 \times (\mu_1 + \mu_2 + \mu_4)$. However, since Adv triggered caching C in its first request, this RTT delay is longer than expected. This again reveals that a consumer might have requested C as private in between two requests by Adv.

Although the previous two scenarios leak private information, Adv does not know whether privacy-sensitive content has actually been requested. This is because caching only happens for a limited time. In either scenario, Adv's second interest for C might have actually been satisfied by the producer because C was evicted from R_1's cache. However, with some additional information about R_1's traffic, i.e., knowledge about caching and eviction patterns at R_1, Adv can increase its success probability. We do not consider this case any further because it is outside the scope of this chapter.

One way to address this information leakage problem is as follows: once an interest for C is marked as non-private, the content must be treated as non-private as long as it remains in a router's cache. Furthermore, the effect of changing the privacy bit (or any other method of specifying content privacy) via an interest should always take effect after the interest is served. In particular, an artificial delay should not be introduced in response to the interest that sets the privacy bit. This delay should only be applied to all subsequent interests for the same content.

As mentioned in Section 3.3, requesting content as private encourages consumer selfishness. The general outcome would be detrimental for all consumers, who would experience high latency even when requested content is cached. However, we claim that consumers have at least one incentive for requesting content without privacy: *reduced delay* for re-transmitted interests in case of packet loss.[5] Requesting content with privacy precludes its re-transmission from router caches (if the original content object is lost) and hence results in higher delays. Thus, we believe that the rational choice for consumers is to request content with privacy only when actually needed.

3.5 Handling Distributed Adversaries

Techniques presented in Section 3.4 enable perfect-privacy. However, they incur potentially severe performance penalties. They are effective for preventing attacks by local adversaries. However, if an adversary is distributed and connected to (or controls) multiple upstream routers, artificial delay introduced by the first-hop forwarder can be easily bypassed. This type of attack and its mitigation are addressed in this section.

[5]Packet loss rate in today's Internet hovers around 4% [43].

3.5.1 Distributed Timing Attack

Consider a topology in Figure 3.6 where Adv is adjacent to both R_1 and R_2. We also assume that both routers honor private content requests and start with empty caches. The following attack shows how Adv can exploit access to both R_1 and R_2 to circumvent the artificial delay countermeasure proposed above.

1. Cr_1 asks for C as private.

2. Adv asks for C first from R_1 and then from R_2 as non-private.

The delay for Cr_1's interest is $2 \times (\mu_1 + \mu_2 + \mu_3)$. The RTTs for Adv's interests are $2 \times (\mu_1 + \mu_2 + \mu_3)$ and $2 \times (\mu_3 + \mu_4)$, respectively. The latter is because Cr_1's interest marked the content as private in R_2's cache. If Cr_1 did not ask for this content, then RTTs for both of Adv's interests would be $2 \times (\mu_1 + \mu_2 + \mu_4)$ and $2 \times \mu_3$. Since neither Adv interest is private, the corresponding content is cached in both routers (as a consequence to Adv's first request) as non-private.

This attack works because R_1 and R_2 do not share content-specific privacy status bits. In other words, regardless of C's privacy status in R_1's cache, R_2 makes a local decision when receiving requests for C.

3.5.2 Mitigating Distributed Adversaries

We propose a simple technique to deal with this type of attack. Whenever a router makes a privacy status change for any cached content, it notifies the upstream router through which that content was received. We do not mandate a specific form of notification, as long as it contains the name of the content in question and some form of authentication. A router treats these notifications as a signal to make a local privacy status change for the content in

question, if it is still cached. A router then forwards the notification to its upstream peers, and so on. This goes on until notifications reach the producer or a router where the content is not cached. We expect that notifications would not travel far since most caching will likely occurs at the edge of a network [74].

To see how the proposed technique works, consider the distributed attack example described above. Assume that Cr_1 performs step (1) to obtain C from R_1 privately. If R_1 notifies R_2 about its privacy status change for C before replying to Adv, R_2 marks its cached copy of C as non-private. Observed RTTs from Adv's interests are $2 \times (\mu_1 + \mu_2 + \mu_4)$ and $2 \times \mu_3$, the same as if Cr_1 did not ask for C. Therefore, Adv can not use these RTTs to determine whether Cr_1 asked for C.

The rationale behind this approach is as follows. Artificial delay works to prevent timing attacks because it hides the presence of cached content. If R_1 did not have a cache, Adv's first interest would be forwarded to R_2. In this case, Adv's timing attack would be unsuccessful. Thus, the notification supplements artificial delay to emulate this scenario.

Also, using notifications and artificial delay is much more efficient than forwarding interests without delays, because notifications (as described) only carry a content name and do not expect a subsequent content response.

3.6 Improving Privacy-Utility Trade-Off

So far we considered techniques where cache hits for private content are always delayed. Proposed techniques are secure according to Definition 3.2, i.e., perfectly private, for all privacy-sensitive content. This is a strong security notion which may not be required in practice. We believe that there are factors, such as content popularity, that allow us to avoid hiding cache hits for private content without significantly compromising privacy. In

this section, we discuss and analyze more practical techniques that relax the *perfect privacy* requirement in favor of better performance, i.e., higher utility. In general, such techniques randomly decide whether to mimic a cache hit or a cache miss for each content request. The distribution of observed output reflects the (local) popularity of requested content.

3.6.1 A Non-Private Naïve Approach

Let ρ_C denote the number of requests for particular content C. The algorithm always generates a cache miss, iff $\rho_C \leq k$, where k denotes the size of the anonymity set. A cache hit indicates that at least k requests have been generated for C.

This approach has a drawback of Adv being able to determine whether C was previously requested. To do so, Adv (knowing k) issues requests for C until it determines that the content is coming from R's cache. Let ρ'_C be the number of such requests. If $\rho'_C > 0$, Adv learns that exactly $k - \rho'_C$ requests have been issued for C.

3.6.2 Random-Cache

Security of the previous scheme depends on Adv's knowledge of k. Our next scheme – *Random-Cache* – selects a random k for each content. Thus, the index of the first cache hit in the output sequence is random, and should not leak information about the router's cache.

As shown in Algorithm 2, the scheme works as follows: the router maintains a counter ρ_C for each C. The first request for C is always a cache miss, and ρ_C is initialized to 0. Also, k_C is picked randomly from $[0, K)$ according to a distribution of domain $[0, K)$, described by a random variable $\mathbb{K}$. Upon receipt of a new request for C, the router increments ρ_C and checks whether $\rho_C \leq k_C$. If so, it generates a cache miss, and a cache hit otherwise.

Algorithm 2 Random-Caching

Input: Request for content C, Domain size K, Distribution of $\mathbb{K}$
Output: Cache hit or cache miss
$\mathbb{T}$:= set of received content
if $C \notin \mathbb{T}$ **then**
    Select k_C from $[0, K)$ with probability $\mathsf{Pr}(\mathbb{K} = k_C)$
    $\mathbb{T} := \mathbb{T} \cup \{C\}$
    $\rho_C := 0$
    Output cache miss
else
    $\rho_C := \rho_C + 1$
    if $\rho_C \leq k_C$ **then**
        Output cache miss
    else
        Output cache hit
    end if
end if

We define *utility* as the ratio of expected number of cache hits and the total number of requests for a given content, i.e., it represents the fraction of interests satisfied from the cache.

Definition 3.4 (Utility)**.** *Let $\mathcal{H}(\rho)$ denote the random variable describing the distribution of the number of cache hits depending on the total number of requests ρ ($\rho \geq 1$). The utility function $u : \mathbb{N} \to \mathbb{R}^+$ of a cache management scheme is defined as: $u(\rho) = \frac{1}{\rho}\mathbb{E}(\mathcal{H}(\rho))$.*

In practice, we derive u using the average number of cache misses, instead of cache hits, which is easier to compute. Let $\mathcal{M}(\rho)$ denote the random variable describing the distribution of the number of cache misses, based on the total number of requests ρ ($\rho \geq 1$). Then, $\mathcal{M}(\rho) + \mathcal{H}(\rho) = c$, and $u(\rho) = 1 - \frac{1}{\rho}\mathbb{E}(\mathcal{M}(\rho))$.

Specifically, for *Random-Cache*, we have:

$$\mathbb{E}(\mathcal{M}(\rho)) = \sum_{i=1}^{\rho} i \cdot \mathsf{Pr}(\mathbb{K} = i-1) + \sum_{i=\rho+1}^{K} \rho \cdot \mathsf{Pr}(\mathbb{K} = i-1), \qquad \text{if } 1 \leq \rho < K \tag{3.1}$$

$\mathbb{K}$ influences both privacy and utility. If cache misses occur with overwhelming probability, then we obtain (almost) perfect privacy with nearly no utility.

Uniform-Random-Cache

If $\mathbb{K}$ is uniform, then we obtain the best privacy among all distributions in terms of ε (which is 0). Also, as shown below, we can decrease δ (and improve privacy) by increasing K at the cost of degrading utility. We refer to this instantiation of Random-Cache as *Uniform-Random-Cache*.

Formally, let $\mathcal{U}(0, K)$ denote a discrete uniform random variable, i.e., $\mathsf{Pr}(\mathcal{U}(0, K) = r) = 1/K$, $0 \leq r < K$. *Uniform-Random-Cache* is an instantiation of Random-Cache (Algorithm 2) with $\mathbb{K} = \mathcal{U}(0, K)$.

Theorem 3.1 (Privacy)**.** *If all cached content is statistically independent, Uniform-Random-Cache is* $(k, 0, \frac{2k}{K})$*-private.*

Proof. Slightly abusing the notation, let $Q_0(\mathcal{C}, r)$ and $Q_1(\mathcal{C}, r)$ denote the output of Algorithm 2 in states S_0 and S_1, respectively, with C when $k_C = r$. (Recall that $S_0(C) = 0$ and $S_1(C) = x$, where $1 \leq x \leq k$). In addition, $Q_0^t(C, r)$ and $Q_1^t(C, r)$ denote the sequence of outputs obtained by executing Algorithm 2 with C consecutively t times, in states S_0 and S_1, respectively. Since all content is statistically independent: (1) it does not matter whether S_0 and S_1 differ in more than one content's count, and (2) Adv's best strategy is to request the same content multiple times in order to infer information about router state. Let $\mathcal{Q}_0^t$ and $\mathcal{Q}_1^t$ denote two random variables describing $Q_0^t(C, r)$ and $Q_1^t(C, r)$ when r is selected uniformly at random according to Line 5 of Algorithm 2.

We show that, for all content C, $\mathcal{Q}_0^t$ and $\mathcal{Q}_1^t$ are $(0, \frac{2x}{K})$-probabilistically indistinguishable. This implies that *Uniform-Random-Cache* is also $(0, \frac{2x}{K})$–probabilistically indistinguishable with any C, assuming all content is statistically independent.

The output of $\mathcal{Q}_0^t$ and $\mathcal{Q}_1^t$ is a sequence with length t consisting of two sub-sequences; the prefix, which is composed of consecutive cache misses (i.e., sequence of 0's), and the suffix with consecutive cache hits (i.e., a sequence of 1's).

We partition output space $\Omega = \mathit{Range}(\mathcal{Q}_0^t) \cup \mathit{Range}(\mathcal{Q}_1^t)$ into Ω_1, Ω_2 and Ω_3, for all t and C, as follows:

- $\Omega_1 = \mathit{Range}(\mathcal{Q}_1^t) \setminus \mathit{Range}(\mathcal{Q}_0^t)$: If $r \in [0, x)$, then all the t replies are cache hits in state S_1. However, this output can not appear with S_0 where the very first answer is always a cache miss (the router first needs to retrieve the content). Thus, $\nexists r'$ such that $Q_1^t(C, r) = Q_0^t(C, r')$.
- $\Omega_2 = \mathit{Range}(\mathcal{Q}_0^t) \cap \mathit{Range}(\mathcal{Q}_1^t)$: If $r \in [x, K - x)$, then $Q_1^t(C, r) = Q_0^t(C, r - x)$.
- $\Omega_3 = \mathit{Range}(\mathcal{Q}_0^t) \setminus \mathit{Range}(\mathcal{Q}_1^t)$: If $r \in [K - x, K)$, then the output with S_0 contains at least $K - x + 1$ cache misses, which is not possible with S_1. Hence, $\nexists r'$ such that $Q_0^t(C, r) = Q_1^t(C, r')$.

For output $O \in \Omega$, let $\mathit{prefix}(O)$ denote prefix length of O (i.e., # cache misses in O). Since k_C is selected uniformly at random, for all $O \in \Omega_2$, $\Pr(\mathcal{Q}_0^t = O) = \Pr(k_C = \mathit{prefix}(O) - 1) = \Pr(k_C = \mathit{prefix}(O) + x - 1) = \Pr(\mathcal{Q}_1^t = O)$. Hence, $\varepsilon = 0$. Moreover, if $O \in \Omega_1 \cup \Omega_3$, $\Pr(\mathcal{Q}_0^t = O) + \Pr(\mathcal{Q}_1^t = O) = \frac{1}{K}$. Since $|\Omega_1 \cup \Omega_3| = 2x$, we obtain $\delta = \Pr(\mathcal{Q}_0^t \in \Omega_1 \cup \Omega_3) + \Pr(\mathcal{Q}_1^t \in \Omega_1 \cup \Omega_3) = \frac{2x}{K} \leq \frac{2k}{K}$. □

This theorem states that the probability that Adv can determine whether a content has been requested zero or k times is $2k/K$. This is because observing any outcome (hit/miss) which can occur in state S_0, but not in S_1, (or vice-versa) occurs with probability $2x/K$. The analysis also shows that perfect privacy can not be achieved if a cache hit can be generated, with non-zero probability.

Theorem 3.2 (Utility)**.** *For Uniform-Random-Cache,* $u(\rho) = 1 - \frac{1}{\rho}\mathbb{E}(\mathcal{M}(\rho))$*, where*

$$\mathbb{E}(\mathcal{M}(\rho)) = \rho\left(1 - \frac{\rho - 1}{2K}\right), \qquad \text{if } 1 \leq \rho < K$$

Proof. The theorem follows from Equation 3.1. If $1 \leq \rho < K$, we have

$$\begin{aligned}
\mathbb{E}(\mathcal{M}(\rho)) &= \sum_{i=1}^{\rho} i \cdot \mathsf{Pr}(\mathbb{K} = i-1) + \sum_{i=\rho+1}^{K} \rho \cdot \mathsf{Pr}(\mathbb{K} = i-1) \\
&= \sum_{i=1}^{\rho} i \cdot \frac{1}{K} + \sum_{i=\rho+1}^{K} \rho \cdot \frac{1}{K} \\
&= \left(\frac{\rho(\rho+1)}{2}\right)\left(\frac{1}{K}\right) + (K-\rho)\left(\rho \cdot \frac{1}{K}\right) \\
&= \left(\frac{\rho(\rho+1)}{2K}\right) + \rho - \left(\frac{\rho^2}{K}\right) \\
&= \rho + \frac{\rho^2+\rho}{2K} - \frac{2\rho^2}{2K} \\
&= \rho - \frac{\rho^2-\rho}{2K} = \rho\left(1 - \frac{\rho-1}{2K}\right)
\end{aligned}$$

□

Theorems 3.1 and 3.2 show that, by increasing the size of domain K, resulting privacy increases at the cost of degraded utility.

Exponential-Random-Cache

One drawback of uniform distribution is that having only one parameter (K) gives limited flexibility for adjusting the privacy/utility trade-off. Hence, we also consider truncated geometric distribution as a candidate for $\mathbb{K}$. The shape of this truncated geometric distribution can be calibrated through an extra parameter other than K. Assigning exponentially larger probability to small values of k_C results in fewer cache misses on average, at the cost of additional privacy loss (ε will increase). The corresponding scheme is called *Exponential-Random-Cache*.

Consider a random variable $\mathcal{G}(\alpha)$ with geometric distribution, i.e., $\mathsf{Pr}(\mathcal{G}(\alpha) = k) = (1-\alpha)\cdot\alpha^k$, where $k \geq 0$ and $0 < \alpha \leq 1$. Its truncated counterpart denoted by $\widetilde{\mathcal{G}}(\alpha, x_1, x_2)$ has a conditional probability distribution defined as: $P(\widetilde{\mathcal{G}}(\alpha, x_1, x_2) = k) = \frac{\mathsf{Pr}(\mathcal{G}(\alpha)=k)}{\sum_{i=x_1}^{x_2} \mathsf{Pr}(\mathcal{G}(\alpha)=i)}$ if $x_1 \leq k \leq x_2$ and 0 otherwise, where $[x_1, x_2]$ is the truncation interval ($x_1, x_2 \in \mathbb{N}^0$). Therefore:

$$\mathsf{Pr}(\widetilde{\mathcal{G}}(\alpha, 0, K) = r) = \frac{(1-\alpha)\cdot\alpha^r}{1-\alpha^{K+1}} \tag{3.2}$$

Exponential-Random-Cache is an instantiation of *Random-Cache* (Algorithm 2) with $\mathbb{K} = \widetilde{\mathcal{G}}(\alpha, 0, K-1)$. Here, α and K are input parameters of the algorithm that can be calibrated to achieve the desired privacy/utility trade-off:

Theorem 3.3 (Privacy)**.** *If all cached content is statistically independent, Exponential-Random-Cache is* $(k, -k\ln(\alpha), \frac{1-\alpha^k+\alpha^{K-k}-\alpha^K}{1-\alpha^K})$*-private*

Proof. The proof is similar to that of Theorem 3.1. We assume the same annotations and show that, for all content C, $\mathcal{Q}_0^t$ and $\mathcal{Q}_1^t$ are $(-k\ln(\alpha), \frac{1-\alpha^k+\alpha^{K-k}-\alpha^K}{1-\alpha^K})$-probabilistically indistinguishable

We identically partition Ω into Ω_1, Ω_2, Ω_3 similar to Theorem 3.1. If $O \in \Omega_2$,

$$\begin{aligned}
\frac{\Pr(\mathcal{Q}_0^t = O)}{\Pr(\mathcal{Q}_1^t = O)} &= \frac{\Pr(k_{\mathcal{C}} = \mathit{prefix}(O) - 1)}{\Pr(k_{\mathcal{C}} = \mathit{prefix}(O) + x - 1)} \\
&= \frac{\Pr(\widetilde{\mathcal{G}}(\alpha, 0, K-1) = \mathit{prefix}(O) - 1)}{\Pr(\widetilde{\mathcal{G}}(\alpha, 0, K-1) = \mathit{prefix}(O) + x - 1)} \\
&= \frac{\frac{(1-\alpha)\cdot\alpha^{(\mathit{prefix}(O)-1)}}{1-\alpha^K}}{\frac{(1-\alpha)\cdot\alpha^{(\mathit{prefix}(O)+x-1)}}{1-\alpha^K}} \\
&= \frac{\alpha^{(\mathit{prefix}(O)-1)}}{\alpha^{(\mathit{prefix}(O)+x-1)}} \\
&= \alpha^{-x}
\end{aligned}$$

Similarly, $\frac{\Pr(\mathcal{Q}_1^t=O)}{\Pr(\mathcal{Q}_0^t=O)} = \alpha^x$. Since $\alpha < 1$, we have $\varepsilon \leq \ln \alpha^{-k} = -k\ln(\alpha)$. In addition, $\Pr(\mathcal{Q}_1^t \in \Omega_1) = \sum_{i=1}^{x} \frac{(1-\alpha)\alpha^{i-1}}{1-\alpha^K} = \frac{1-\alpha^x}{1-\alpha^K}$, and $\Pr(\mathcal{Q}_0^t \in \Omega_3) = \sum_{i=K-x+1}^{K} \frac{(1-\alpha)\alpha^{i-1}}{1-\alpha^K} = \frac{\alpha^{K-x}-\alpha^K}{1-\alpha^K}$. Since $\Pr(\mathcal{Q}_0^t \in \Omega_1) = \Pr(\mathcal{Q}_1^t \in \Omega_3) = 0$, we get $\Pr(\mathcal{Q}_0^t \in \Omega_1 \cup \Omega_3) + \Pr(\mathcal{Q}_1^t \in \Omega_1 \cup \Omega_3) \leq \frac{1-\alpha^k+\alpha^{K-k}-\alpha^K}{1-\alpha^K}$. □

Theorem 3.4 (Utility)**.** *For Exponential-Random-Cache,* $u(\rho) = 1 - \frac{1}{\rho}\mathbb{E}(\mathcal{M}(\rho))$*, where*

$$\mathbb{E}(\mathcal{M}(\rho)) = \frac{1-\alpha^\rho-\rho\alpha^K}{1-\alpha^K} + \frac{\alpha(1-\alpha^\rho)}{(1-\alpha^K)(1-\alpha)}, \qquad \text{if } 1 \leq \rho < K$$

Proof. Using the fact that $\sum_{i=1}^{\rho} i\alpha^{i-1} = \frac{d}{d\alpha}\sum_{i=1}^{\rho}\alpha^i = \frac{1-(\rho+1)\alpha^\rho}{1-\alpha} + \frac{\alpha(1-\alpha^\rho)}{(1-\alpha)^2}$ and $\sum_{i=\rho+1}^{K}\alpha^{i-1} = \frac{\alpha^\rho-\alpha^K}{1-\alpha}$ the theorem follows from Equation 3.1. If $1 \leq \rho < K$, and using the conditional probability in Equation 3.2, we have

$$\begin{aligned}
\mathbb{E}(\mathcal{M}(\rho)) &= \sum_{i=1}^{\rho} i \cdot \mathsf{Pr}(\mathbb{K} = i-1) + \sum_{i=\rho+1}^{K} \rho \cdot \mathsf{Pr}(\mathbb{K} = i-1) \\
&= \sum_{i=1}^{\rho} i \cdot \frac{(1-\alpha)\cdot\alpha^{i-1}}{1-\alpha^K} + \sum_{i=\rho+1}^{K} \rho \cdot \frac{(1-\alpha)\cdot\alpha^{i-1}}{1-\alpha^K} \\
&= \frac{1-\alpha}{1-\alpha^K} \cdot \sum_{i=1}^{\rho} i\alpha^{i-1} + \frac{\rho(1-\alpha)}{1-\alpha^K} \cdot \sum_{i=\rho+1}^{K} \alpha^{i-1} \\
&= \left[\frac{1-\alpha}{1-\alpha^K} \cdot \left(\frac{1-(\rho+1)\alpha^\rho}{1-\alpha} + \frac{\alpha(1-\alpha^\rho)}{(1-\alpha)^2}\right)\right] + \left[\frac{\rho(1-\alpha)}{1-\alpha^K} \cdot \left(\frac{\alpha^\rho-\alpha^K}{1-\alpha}\right)\right] \\
&= \frac{1-(\rho+1)\alpha^\rho}{1-\alpha^K} + \frac{\alpha(1-\alpha^\rho)}{(1-\alpha^K)(1-\alpha)} + \frac{\rho(\alpha^\rho-\alpha^K)}{1-\alpha^K} \\
&= \frac{1-\alpha^\rho-\rho\alpha^K}{1-\alpha^K} + \frac{\alpha(1-\alpha^\rho)}{(1-\alpha^K)(1-\alpha)}
\end{aligned}$$

□

3.6.3 Comparison of Proposed Schemes

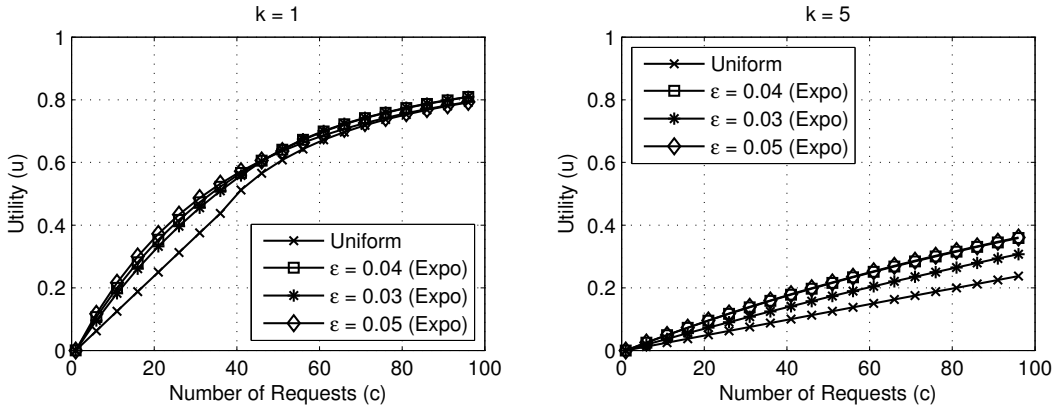
Increasing α in the Exponential-Random-Cache scheme results in better privacy (smaller ε). However, δ can not be made arbitrarily small and it is ultimately determined by α. In particular, $\delta = 1 - \alpha^k$ when $K = \infty$, which is the smallest possible δ. In contrast, δ of the uniform distribution can be arbitrarily decreased by sufficiently increasing K.

We compare the utility of proposed schemes in Figure 3.7. In Figure 3.7(a), we adjust the same value of δ, that is 0.05, for both schemes, and plot their utility for different values of k while varying ε. In Figure 3.7(b), we compute the maximum value of $\varepsilon = -\ln(1 - \delta)$ for various combinations of δ and k, and plot the difference between the utility functions of the two schemes for varying δ. Both figures show that the exponential scheme exhibits up to 12% performance gain over the uniform one. Figure 3.7(a) also shows that both schemes achieve better utility as the number of requests grows.

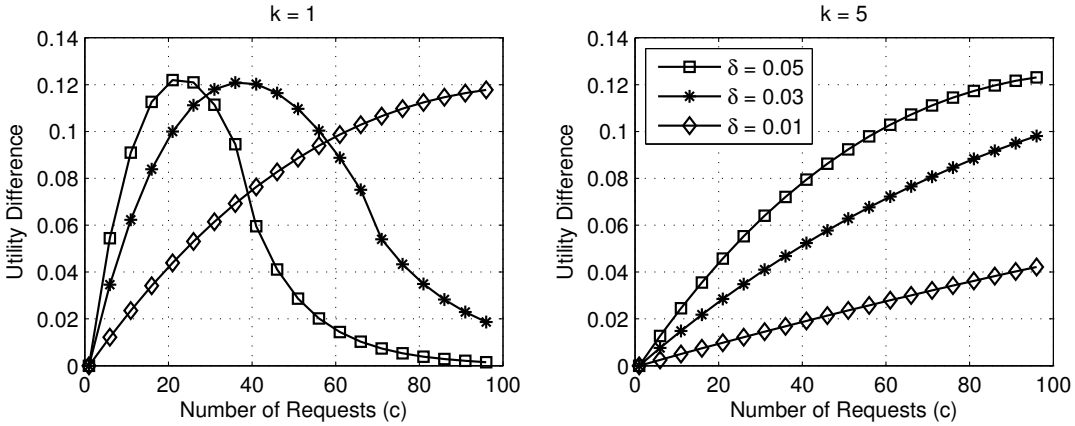
3.6.4 Addressing Content Correlation

Random-Cache requires statistically independent content in the cache, which is a very strong assumption. Multiple content objects may share the same name prefix, and their access patterns could be strongly correlated. Under this assumption, Random-Cache as described above becomes insecure since it allows Adv to sample multiple points under different k. By requesting a large number of related content objects, as soon as Adv receives one without any delay, it learns that, with overwhelming probability, the whole set of content has been requested before.

To alleviate this problem, correlated content must be grouped together. Algorithm 2 can then be applied to these groups rather than to individual content, i.e., using a single counter ρ_C and value of k_C.



(a) Utility depending on privacy ($\delta = 0.05$)



(b) Maximal utility difference between *Uniform-Random-Cache* and *Exponential-Random-Cache* when $\varepsilon = -\ln(1-\delta)$

Figure 3.7: *Uniform-Random-Cache* vs. *Exponential-Random-Cache*

Even the above extension can not be proven secure against all correlation-based attacks. In many cases, content correlation is even more subtle (e.g., semantically related content having different names such as linked webpages). This might be identified with appropriate background knowledge. As a possible countermeasure, content could be augmented with a content ID field. Producers would populate such field with identical values for correlated content. Routers could then determine how to handle such content by observing this field. However, a thorough analysis of these attacks and of the corresponding countermeasures is beyond the scope of this dissertation.

3.7 Experimental Evaluation

We now evaluate the actual impact of cache privacy techniques through simulations. We do not include the method that involves notification messages since we believe that caching will most likely take place at the edge [74]. Thus, notification messages will traverse only one hop (or few hops) before being dropped by the first non-caching router.

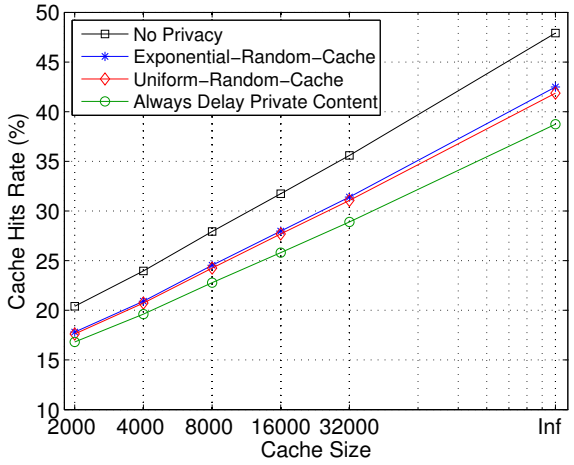
We experiment using HTTP traffic traces collected by IRCache [8], which is part of the National Laboratory for Advanced Network Research (NLANR) project [12]. Traces were collected on September 1, 2007 (over a 24-hour period), on a Web proxy located at Research Triangle Park, North Carolina. These traces reflect activity of 185 users, for approximately 3.2 million requests distributed over various destinations. We randomly divide these requests into two sets: private and non-private. Then, we "replay" them with the following algorithms:

1. **No Privacy.** Routers use no privacy-preserving techniques.

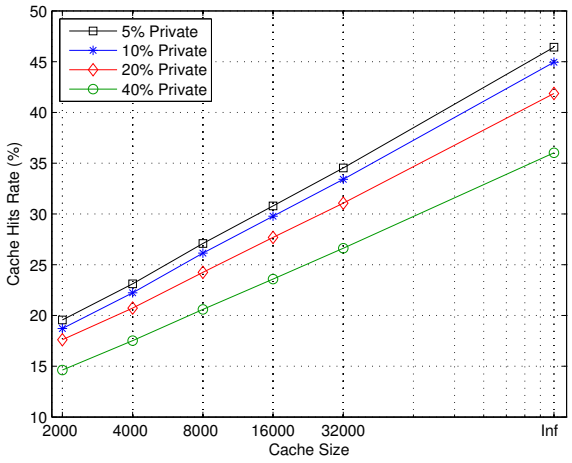
2. **Always Delay Private Content.** For each request of a (cached) *private* content, the router always generates a cache miss, while for *non-private* cached content the response is always cache hit. This implements the basic protocol in Section 3.4.2.

3. **Uniform-Random-Cache/Exponential-Random-Cache.** Requests for cached *private* content are handled according to Algorithm 2. Requests for *non-private* cached content always result in a cache hit.

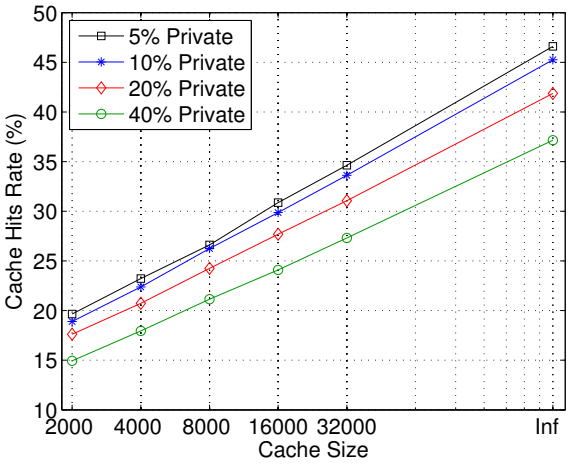
A router caches all content and removes elements from its cache (when full) according to the Least Recently Used (LRU) policy. In case of a cache hit, the corresponding cache entry becomes "fresh" even if the response is delayed.



(a) Comparison of our techniques



(b) *Uniform-Random-Cache* varying number of private requests



(c) *Exponential-Random-Cache* varying number of private requests

Figure 3.8: Cache hit rates: experimental evaluation results

For the algorithms that classify content into private and non-private, each incoming request is randomly marked as private with probabilities 0.05, 0.1, 0.2, and 0.4. Without loss of generality, we assume that all content is of the same size. We consider caches of size: $2,000$, $4,000$, $8,000$, $16,000$, and $32,000$ units (content objects). Furthermore, as a baseline, we also run the same algorithms with a cache of infinite size.

We set $k = 5$ and $\varepsilon = 0.005$. Results are plotted in Figure 3.8. Random caching algorithms have little impact on the percentage of cache hit rates. There is, at most, a 5% decrease in

hit percentages for nearly all observed cache sizes. Increasing the amount of private interests also had little effect on the cache hit percentage. There is, at most, a 10% drop in the hit percentage as the rate of private interests increased from 0.05 to 0.4.

3.8 Bypassing Cache Delays

Thus far, our goal has been perfect privacy which we attempted to attain by introducing artificial delays in routers when cache hits occur. We also described how to improve upon this deterministic delay with randomized delay algorithms. However, such strategies are problematic in certain scenarios:

1. If a consumer needs to resend an interest marked as private due to packet loss or transmission errors, the upstream router delays responding.

2. If a consumer is co-located with other trusted consumers then there is no reason the latter should be penalized by artificial delays if all parties request the same content. This is particularly true for the local application topology in Figure 3.3, where different applications running on the same consumer device may inherently trust one another since their common owner trusts all applications. In this scenario, if two applications request the same content through the local router, neither should be subjected to artificial delay.

In this section, we modify prior countermeasures so as to allow consumers to bypass the artificial delay. This does not require any trust relationship between consumers and routers. The basic idea is that each consumer creates a special cache-bypass token $y = H(x)$ which is enclosed with each interest, where H is a cryptographic hash function and x is a random l-bit string generated and stored for each request. The token is placed into a special per-hop

header field called `Token`. When R receives such interest Int for C on interface F with a non-empty `Token` field, it performs the following steps:

1. If C is not cached locally, R forwards Int upstream according to its local forwarding strategy.
2. When C is returned, R stores the value of `Token` and F in the cache alongside C and its retrieval RTT.

If a consumer wishes to bypass the artificial delay for C, it must present the token pre-image x in an interest. The pre-image is also included in the `Token` field. Upon receipt of such an interest from the same interface F, R performs the following steps:

1. R computes $y' = H(x)$.
2. If $y' = y$, R responds with C with no delays. Otherwise, R behaves according to the random cache algorithm described above.

This method does not introduce per-user state and requires no trust relationship between consumers and routers. However, one drawback is that routers are now required to store an additional l-bit value for every cached content and perform one hash computation per interest.

The use of the bypass token is limited to consumers connected to the same local network (including different applications on the same device). The token can be exchanged between neighboring consumers using ARP-like request and response queries [161]. This, however, implies that Adv must not be able to eavesdrop on these messages, which complies with the adversary model in Section 3.2. However, Adv connecting to an encrypted network, e.g., in coffee shops with protected public access points, can still eavesdrop on all packets. In this

case, there is no need for routers to obscure cache hits from cache misses since Adv learns what is requested by eavesdropping.

Chapter 4

Network-Layer Trust

As mentioned above, efficient content distribution is facilitated by caches. This reduces overall latency and improves bandwidth utilization for popular content. Despite its benefits, in-network caching opens the door for several types of Denial-of-Service (DoS) attacks [77]. One such attack is content poisoning, where an adversary injects fake content into router caches. When such content is used to satisfy subsequent interests, it results in amplified fake content distribution to other caches and consumers.

CCN mandates that each content must be verifiable using, for instance, a signature generated by its producer. Consumers are expected to verify content signatures in order to assert:

- *Integrity* – a valid signature (computed over a content hash) guarantees that signed content is intact.
- *Origin Authentication* – since a signature is bound to the public key of the signer, anyone can verify whether content originates by its claimed producer.

- *Correctness* – since a signature binds content name to its payload, a consumer can securely determine whether delivered content corresponds to what was actually requested.

However, mandating signature verification by routers involves two additional requirements:

1. A router must be aware of the specific trust model for each content-producing application. Given the wide range of possible applications, it is very unlikely that they will all use the same trust model. Some applications will use trust hierarchies while others might adopt a flat peer-based trust models or hybrid versions thereof. Furthermore, the set of CCN applications will change over time, and the trust model of a particular application might not be static in the long term.

2. Depending on the trust model of an application associated with a particular content, a router needs access to – and thus might need to fetch – a chain of public key certificates in order to make a trust decision.[1] For instance, if an application uses a hierarchical PKI, an entire leaf-to-root certificate might have to be traversed and all intermediate certificates would need to be separately verified. This would also need to include expiration and revocation checking for each such certificate.

One way to deal with these issues is to make content signature verification by routers optional. Unfortunately, this leaves CCN vulnerable to the content poisoning attacks. To make matters worse, CCN does not provide any mechanism for consumers to request genuine content. For instance, in NDN, a consumer that receives fake content can explicitly exclude it (by referring to its hash) in subsequent requests. This does not guarantee eventual success, due to the potentially unbounded number of fake content objects, sharing the same name, that can be injected into the network.

[1]The alternative of carrying the entire collection of certificates as part of each content is clearly undesirable.

This undesirable state-of-affairs serves as the main motivation for this chapter. We first analyze CCN's susceptibility to content poisoning attacks. Next, we postulate some requirements for routers to support trust management and content validation. We then present simple rules that allow CCN parties (consumers, producers, and routers) to deterministically mitigate content poisoning while minimizing trust-related complexity for routers. These rules require no changes to the fundamentals of the CCN architecture.

As discussed later, proposed rules might not be applicable for some routers. Therefore, we also propose, in this chapter, a lightweight ranking algorithm for cached content. Its goal is to assign higher rank values to valid content objects than to fake ones. Although it represents a probabilistic approach, we show that our ranking algorithm can be effective in mitigating content poisoning attacks in some attack scenarios.

4.1 Content Poisoning

Since CCN routers are not required to verify signatures, delivered content is not guaranteed to be authentic. However, consumers are required to verify signatures of all received content objects, allowing them to detect fake ones. To do so, consumers need to have the application-specific trust context.

CCN has no means for consumers to ask routers to *flush* fake content from their caches. In NDN, consumers detecting fake content can only exclude said content in subsequence interests by specifying its hash in interest exclusion filters. However, excluding content does not necessarily imply that it is fake since the same feature is also used to exclude stale content. Furthermore, even if the exclusion technique were to be used strictly for flagging poisoned content, the result would be undesirable. The entire notion of consumers (i.e, end-

systems or hosts) informing routers about poisoned content is problematic for the following reasons:

1. Suppose a consumer complains to a router about a specific content. If this is done without consumer authentication, the router has two choices: (1) immediately flush referenced content or (2) verify the content signature and flush content only if verification fails. The first option (1) is problematic since it opens the door for anyone to cause easy removal of popular content from router caches – a type of a DoS attack. Furthermore, as noted in [77], the adversary mounting a content poisoning attack could continue *ad infinitum* to feed new invalid content in response to interests that exclude previously consumer-detected invalid content. The second option (2) is equally unattractive. Not only content signature verification is costly (which can lead to a DoS attack by itself), it brings back the problem of routers having to understand various and potentially complex trust semantics of content-producing applications.

2. Suppose that consumers are required to authenticate themselves when complaining about poisoned content. This would entail signing the interest that complains about allegedly fake content. However, signing interests would violate consumer privacy by exposing their identities to the network, i.e., to routers.[2] Also, corresponding verification (public) keys would have to be communicated with each complaint message, along with auxiliary information that routers would need to trust the keys. This approach is undesirable since it can be abused to mount DoS attacks on routers by flooding them with invalid complaints, forcing expensive signature verification operations.[3] Note that, even if the router successfully authenticates a consumer complaint, this does not guarantee that the accused content is fake. In order to be sure, the router would have to

[2]Recall that consumer anonymity is a feature of CCN. It is a consequence of interests having no source or destination addresses.

[3]The same attack does not work with flooding routers with fake content since content can not be sent unsolicited. A router would only attempt signature verification of incoming content for which it has an entry in its PIT.

verify the content signature as well. Moreover, authentication of consumers by routers would require identity management and verification systems at the network layer, thus adding more complexity and overhead.

Finally, since CCN only recommends, and does not mandate, content caching. It is entirely *normal* for a router not to cache some, or all, content that it forwards. If a router does not cache C, then complaining about C being fake is clearly useless.

4.1.1 Injecting Fake Content

Thus far, we mentioned content poisoning attacks without describing how they take place. Before going into details, we define the terminology used in the rest of this chapter.

- A *fake* content is a content object that contains one of the following:
  - An invalid signature, i.e., the signature verification algorithm returns an error.
  - A valid signature generated with the wrong key, i.e., not the key of the purported producer.
  - A signature field that is somehow malformed, e.g., formatted badly.
- A content object is *valid* if it contains a valid signature verifiable with the correct public key. (The meaning of *correct* key is discussed in Section 2.1.)
- *Adversary*, Adv, is any CCN entity (or a collaborating group thereof) capable of injecting fake content into the network.
- *Content poisoning* is an attack whereby Adv injects fake content into router caches.

We consider an Adv that anticipates interests for content C with name n and injects fake content with the same name into router caches. Fake content can be injected into the network

via malicious routers or end-nodes. For example, consider an Adv consisting of a malicious consumer Cr_m and a malicious producer P_m that target a specific router R_v. Assume that Cr_m and P_m are (directly or indirectly) connected to different interfaces of R_v. When Cr_m sends an interest for n, and this interest is received by R_v, the latter adds a new entry to its PIT. Next, P_m sends a fake content to R_v which is promptly cached and its corresponding PIT entry is flushed. Consequently, R_v is pre-polluted with fake content, ready for arrival of genuine interests. To maximize longevity of the attack, P_m sets the `ExpiryTime` field of fake content to its maximum value. The same attack can be mounted if P_m is replaced by a malicious upstream router R_m, i.e., R_m is on the path between R_v and the actual content producer.

4.1.2 Problem Definition

Based on the above, we conclude that CCN has a major security problem, since it offers: (1) no way to prevent fake content from being delivered to consumers and cached by routers, and (2) no way to reliably flush invalid content from router caches. There are two reasons for this:

1. **Ambiguous interests:** CCN requires each interest to carry the name of desired content. However, neither the content digest, nor the `KeyId` is a required field in an interest. In other words, an interest for a content name can be satisfied by multiple content objects, including those with untrusted or unverifiable signatures.

2. **No unified trust model:** even if routers could verify signatures at line speed, CCN does not provide a trust model enforceable at the network layer. Although the two aforementioned selector fields can be used to communicate content-specific trust context to the network layer, the initial design of CCN has no mechanism for a consumer

to securely pre-acquire the hash of a given content or the specific public key that should be used to verify a content signature.

4.1.3 Goals

Before we address the content poisoning problem, it is necessary to state the obvious, **network-layer trust management and content poisoning are inseparably conjoined**. Since content is the basic unit (currency) of network-layer in CCN, trust in content (and not in its producers or consumers) is the central issue at the network layer. Our main goal is to **minimize or eliminate fake content delivery to benign consumers**. This can be achieved using the following sub-goals:

1. Trust-related complexity (activities, state maintenance, etc.) must be minimized at the network layer. Specifically, as part of validating content, a router should not fetch public key certificates, perform expiration and revocation checking of certificates, maintain its own collection of certificates, or be aware of trust semantics of various applications.[4]

2. A router should verify at most one signature per content. This upper-bounds the heavier part of content-related cryptographic overhead. Ideally, a router would not perform any signature verification at all. As discussed below, this might be possible for some, yet not all, content. Also, although verifying a signature given an appropriate public key is a mechanical operation, a router would still need to support multiple signature algorithms since uniformity across all applications is improbable.

The above discussion implies that CCN entities other than routers, i.e., producers and consumers, *should* bear the brunt of trust management of content at the network layer.

[4] This is only related to trust management for content and is separate from trust management for routing protocols.

4.2 Interest-Key Binding Rule

Ghodsi et al. [79] informally argue that, for each content, at least two out of the three possible bindings (producer-key, name-key, producer-name) must be present. The third binding is transitively inherited from the other two, since, due to the use of human-readable names in CCN, producer–name binding can be easily inferred.[5] Our approach to network-layer trust adheres to all goals outlined above. It is based on the binding between a name and the public key used to verify content signature. We call it the *Interest-Key Binding (IKB)* rule:

IKB: An interest must reflect the public key of the producer.

Recall that CCN interest format (Section 2.1) includes an optional field `KeyId` in CCNx (or `PublisherPublicKeyLocator` in NDN) which serves exactly this purpose. Our approach makes it mandatory without any substantive changes to the architecture.

As discussed in Section 2.1, CCN public keys are a special type of content. They are distributed in the form of a certificate signed by their corresponding issuing CA. Each certificate contains a list of all name prefixes that the associated public key is authorized to sign. As mentioned in Section 2.1, a content name must share a prefix with the public key used to verify its signature. However, this is not the case if the content itself is a key (certificate). Specifically, the name of the key used to verify a certificate (CA public key) and the name of the key contained in a certificate are not required to have any specific relationship, e.g., signed content C can be verified with public key PK, with C and PK having no common prefix. This is part of CCN's philosophy of leaving trust management up to the application. Applications are free to impose all kinds of restrictions as long as routers remain oblivious.

[5] If we assume that names are clear and unambiguous.

4.2.1 IKB Implications for Producers and Routers

We now examine IKB implications for content producers, routers, and consumers.

Producers

For content producers, IKB has very few consequences. In fact, it simplifies content construction by asking the producer to include the public key itself in the `PublicKey` field of content. In other words, IKB obviates two other current CCN options: (1) referring to a verification key by its name, or (2) including it in a form of a certificate. Moreover, producers should include the digest of the public key expected to verify the content signature in CCNx's `KeyId` field (or NDN's `PublisherPublicKeyLocator`). Recall that for a content to be considered a match, its `KeyId` field value should match that (if any) of the corresponding interest.

Routers

For routers, IKB implications are overwhelmingly positive. First, routers do not need to perform any fetching, storing or parsing of public key certificates, as well as no revocation or expiration checking. All such activities are left to consumers.

Upon receiving a content and identifying the corresponding PIT entry, a router simply (1) ensures that `KeyId` values in both interest and content headers match[6], and (2) verifies the content signature using the key provided in `PublicKey` field. If either check fails, the content is discarded. Otherwise, the content is forwarded and optionally cached. One optimization of PIT lookup for each incoming content is to use both content name and `KeyId`.

These implications would be even more beneficial for routers with the use of SCNs as discussed in Section 4.3 below. In this case, inclusion of key information and signature checking

[6] `KeyId` values in interests header are stored in corresponding PIT entries.

could be avoided for most content objects, thus further reducing both bandwidth and computation overhead.[7]

We believe that not all routers might support all signature schemes for different key sizes. For instance, governmental agencies might be using proprietary signature schemes. Behavior of intermediate routers receiving a content with an unsupported signature depends on local policies and configurations. A router might either (1) forward the content *without* caching it, only if its `KeyId` value matches that in the corresponding interest, or (2), more strictly, drop the content. In other words, routers *should not* cache content without verifying its signature. This requirement will be relaxed later in this chapter.

Consumers

For consumers, IKB does not increase complexity. It actually prompts us to codify desired consumer behavior – something that has been left unspecified in the CCN architecture.

The most immediate IKB consequence for a consumer is the need to obtain and validate the producer's public key *before* issuing an interest for any content originated by that producer. At the first glance, this might appear to be an example of the proverbial "chicken-and-egg" problem. However, we show below that this is not the case.

A consumer that wants to fetch certain content C is doing so as part of some CCN application, APP_C. We assume that a consumer must have already installed this application. APP_C must have a well-defined trust management architecture that is handled by its consumer- and producer-side software. However, the remaining question is: how to bootstrap trust and how to obtain initial public keys?

We consider three non-exclusive alternatives:

[7] The use of SCNs does not rule out the use of signatures. See Section 4.3 for details.

1. APP_C client-side software comes with some pre-installed root public key(s), perhaps contained within self-signed certificates. Without loss of generality, we assume that there is only one such key – PK_{root}. Armed with it, a consumer can request lower-level certificates, by issuing an interest referencing the hash of PK_{root} in the `KeyId` field.[8]

2. A global Key Name Service (KNS) [124], somewhat akin to today's DNS. In response to consumer-issued interests referencing public key names and/or name prefixes, KNS would reply with signed content containing one or more public key certificates (i.e., as embedded content) corresponding to requested names.

3. A global search-based service, i.e., something resembling today's Google. A consumer would issue a search query (via an interest) to the search engine which would reply with signed content representing a set (e.g., one page at a time) of query results. One or more of those results would point to content corresponding to the public key certificate of interest to the consumer.

In cases (2) and (3), consumers would still need to somehow securely obtain the root public keys for KNS and the search engine, respectively. This can be easily done via (1).

4.2.2 Security Arguments

We now return to the original motivation – mitigation of content poisoning attacks. We need to show that global adherence to the IKB rule leads to security against content poisoning.

We assume that:

1. Routers and consumers abide by the IKB rule and act as described in Section 4.2.1.

[8] If APP_C comes with several root public keys, the consumer would need to issue multiple simultaneous interests referencing the hash of each root key in `KeyId`.

2. The consumer is not malicious.

3. Each router one hop away from the consumer is not compromised.

4. The link between a consumer and adjacent router is not compromised.

We argue security by contradiction. Suppose that a consumer issues an interest *Int* and receives a fake content C_f from router's (R) cache. According to IKB, *Int* must contain the digest of a public key of producer P in its `KeyId` field. Let PK denote this public key. Since R adheres to IKB, it must have checked that: (1) `KeyId` in C_f matches that of *Int*, and (2) the content signature itself is valid, i.e., verified using the public key PK' provided in C_f. Also, since R and the consumer are not malicious and all communication between them is secure, the only remaining possibility is a hash collision. In other words, $H(PK') = H(PK)$ for $PK' \neq PK$.[9] Since $H(\cdot)$ is assumed to be a suitable cryptographic hash function, collisions occur with negligible probability.

This does not yet conclude our security discussion. As noted in [77], content poisoning attacks can originate with malicious routers. What happens if a malicious router R' feeds poisoned (fake) content C_f to its non-malicious next hop neighbor R, towards some consumer(s)? Since R is honest and adheres to IKB, before forwarding and (optionally) caching C_f, it verifies, as mentioned above, that C_f is valid, i.e., its signature is successfully verifiable using PK', which matches `KeyId` in *Int*. Consequently, R will detect that C_f is fake and discard it.

We now prove that IKB rule mitigates content poisoning attacks.

Definition 4.1 (Second pre-image resistant hash functions)**.** *A hash function $\mathcal{H}$ is* second pre-image resistant*, if for any given x, no probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ can find a value $x' \neq x$ such that $\mathcal{H}(x) = \mathcal{H}(x')$. In other words,* $\Pr[\mathcal{H}(x) = \mathcal{H}(x')] \leq \epsilon(\kappa)$*, where $\epsilon(\kappa)$ is negligible and κ is the security parameter.*

[9] $H(PK')$ is included in `KeyId` in content header, and $H(PK)$ is included in `KeyId` in interest header.

A formal definition of probabilistic polynomial-time adversaries and negligible functions can be found in [96].

Definition 4.2 (Unforgeable signature schemes)**.** *A signature scheme* Π *is* unforgeable *if for any message* m*, no PPT adversary* $\mathcal{A}$ *(given a public key* PK*) can generate a valid signature without knowing the corresponding private key. We denote the success of* $\mathcal{A}$ *as* $\mathcal{A}^{\mathbf{forge}}(m) = 1$*, i.e., if* Π *is unforgeable, there exists a negligible function* $\epsilon(\kappa)$ *such that:* $\Pr\left[\mathcal{A}^{\mathbf{forge}}(m) = 1\right] \leq \epsilon(\kappa)$*.*

Definition 4.3 (CCN cache poisoning experiment)**.** *For any interest message Int with* $\mathcal{H}(PK)$ *(the digest of the verifying public key for the corresponding content) assigned to the* `KeyId` *field, and for any* $\mathcal{A}$*, the* CCN cache poisoning experiment *is defined as follows:*

Given Int as input to $\mathcal{A}$*, it outputs a content object* C' *containing: (1) public key* PK' *in the* `PublicKey` *field, (2) digest* $\mathcal{H}(PK')$ *in* `KeyId`*, and (3) signature* σ' *in the* `Signature` *field. The output of this experiment is* 1 *if one of the following holds:*

- $PK \neq PK'$ *and* $\mathcal{H}(PK) = \mathcal{H}(PK')$*.*
- $PK = PK'$ *and* σ *is valid.*

In other words, $\mathcal{A}$ *can either violate second pre-image resistance of* $\mathcal{H}$ *(we denote this event as* **collision** *which occurs with some probability* p_c *and succeeds with probability* $\Pr[\mathcal{H}(x) = \mathcal{H}(x')]$*), or forge the content signature (we denote this event as* **forge** *which occurs with some probability* p_f *and succeeds with* $\Pr\left[\mathcal{A}^{\mathbf{forge}}(m) = 1\right]$*). We denote the success of* $\mathcal{A}$ *as* $\mathcal{A}^{\mathbf{pois}}(Int) = 1$*.*

Theorem 4.1 (IKB Security)**.** *Given* $\mathcal{H}$ *and* Π *(as per Definitions 4.1 and 4.2, respectively),* $\mathcal{A}$ *succeeds in injecting a fake content object* C' *into a network that abides by the IKB rule with a negligible probability* $\epsilon(\kappa)$*, i.e.,* $\Pr\left[\mathcal{A}^{\mathbf{pois}}(Int) = 1\right] \leq \epsilon(\kappa)$*.*

Proof. (By contradiction) Assume that $\mathcal{A}$ succeeds in injecting C' with a non-negligible probability. Then, we can construct a reduction $\mathcal{A}'$ (another PPT adversary), that uses $\mathcal{A}$ to break second pre-image resistance of $\mathcal{H}$, or unforgeability of Π, as follows.

Adversary $\mathcal{A}'$

- Given a value x.
- Creates *Int* and sets $\mathcal{H}(x)$ as its `KeyId`.
- Runs $\mathcal{A}(Int)$ to obtain C'.
- Extracts from C' and outputs:
  - PK' as a collision with x, if $x \neq PK'$,
  - or σ' as a forged signature for C', if $x = PK'$.

We now determine the probability of success of $\mathcal{A}'$. Whenever either **collision** or **forge** event occurs, $\mathcal{A}'$ succeeds. Therefore,

$$\begin{aligned}\Pr\left[\mathcal{A}' \text{ succeeds}\right] &= \Pr\left[\textbf{collision} \cup \textbf{forge}\right] \\ &= p_c \cdot \Pr\left[\mathcal{H}(x) = \mathcal{H}(PK')\right] \\ &+ p_f \cdot \Pr\left[\mathcal{A}'^{\textbf{forge}}(C') = 1\right] \\ &> \epsilon(\kappa)\end{aligned}$$

The last inequality holds because $\mathcal{A}'$ succeeds with the same probability as $\mathcal{A}$, which is non-negligible. If the result of adding two functions is non-negligible, at least one of them must be non-negligible [96]. Moreover, since both p_c and p_f can not be exponential functions, then either $\Pr\left[\mathcal{H}(x) = \mathcal{H}(PK')\right] > \epsilon(\kappa)$ or $\Pr\left[\mathcal{A}'^{\textbf{forge}}(C') = 1\right] > \epsilon(\kappa)$. This contradicts Definitions 4.1 and 4.2, which concludes our proof. □

4.2.3 Optimization

IKB rule implies that routers perform only one signature verification using the public key provided (by the producer) in the content and specified (by the consumer) using `KeyId` in the interest. An alternative to including the public key in the content, is to directly include it in interests. This, however, requires storing the entire key in the corresponding PIT entry, to be used later for signature verification. Since cache entries have longer lifetime than PIT entries, including keys in interests can be beneficial in terms of storage. The main drawback is that the current interest format would need to be modified to include the requested content public key.

Also, in high-speed routers, performing even a single signature verification per packet might incur appreciable overhead. One way to address this issue is to take advantage of the network structure. The current Internet is composed of Autonomous Systems (AS-s), each representing an administrative entity. In such an architecture, only border routers of consumer-facing AS-s could implement IKB rule and verify signatures of all content. Each router inside an AS might probabilistically verify signatures. Nonetheless, fake content could still be cached by routers that do not perform (or probabilistically perform) signature verification. However, since border routers of consumer-facing AS-s adhere to IKB, *most* fake content can be detected and discarded before reaching the consumer. The reason why not *all* fake content objects can be detected is because some of them can be generated from within the consumer-facing AS, i.e., between the consumer and the nearest IKB-abiding router. Applying this optimization in practice is discussed in Section 4.6.

4.3 Self-Certifying Names

Another way of handling content trust at the network layer is to use SCNs [80, 73, 130, 77, 38]. According to [199], a content name can only have at most two out of the following three properties: security, uniqueness, and human-readability. As suggested in [79, 80], SCNs can be formed by appending to the producer's public key digest a label that uniquely identifies the content. While this approach guarantees security and uniqueness, it lacks human-readability and the means of verifying the binding between the content and its name [181]. Our approach involves forming SCNs by specifying the hash of the content itself along with its name.[10] This provides name-content, producer-name, producer-key, and name-key security bindings described in Section 4.2. Although this does not yield fully human-readable names, it provides required uniqueness and security properties [79].

If a benign consumer uses SCNs in interests, the network guarantees delivery of "valid" content. The main advantage of using SCNs is that routers are no longer required to verify signatures. Instead, they only recompute the hash of the received content and check that it matches that in the corresponding PIT entry. The remaining question is: how can a consumer obtain the hash of a content beforehand?

For the type of communication where most content is requested using SCNs, we advocate the use of so-called *catalogs*. A catalog is basically an authenticated data structure that includes a set of SCNs. This set can consist of references to content objects containing data, public keys, or even other catalogs. The structure of catalogs is application-specific and might vary from a simple list of SCNs, to multiple sets forming a Merkle tree [134] or some similar data structure. To securely fetch an initial catalog consumers would fall back to the IKB rule as discussed earlier.

[10] CCNx provides this hash using the `ContentObjectHash` field of both interest and content, while NDN suggests appending the hash as the last component of the content name [92].

One obvious corollary of using SCNs in interest messages is that consumers (not just routers) are no longer required to verify content signatures, as long as the SCN is trusted, i.e., obtained from a (consumer-verified) catalog. This reduces:

1. publishing overhead, since producers now sign catalogs containing several SCNs rather than signing the individual content objects referenced by these SCNs,

2. network overhead, since there is no need to add the public key in the content, as discussed in Section 4.2.[11]

The only time a signature is required is whenever a content is requested using IKB. Routers (prior to serving content from cache or forwarding it) and consumers (prior to accepting) *must* verify the content signature. We believe that it should be left up to the producer to decide whether a content should be requested by IBK, SCN, or both.

Using SCNs in conjunction with catalogs brings up the issue of unsigned content objects. In other words, a content C, which is indirectly signed as part of a catalog, can be fetched by SCN. This does not rule out C being separately signed by its producer. However, signing catalog-ed content increases overhead for the producer and content size. A sensible approach is not to sign catalog-ed content objects at all. This would imply that such objects can *only* be fetched via SCN. However, the original design of both CCNx and NDN architectures requires each content to be *individually* verifiable. Thus, existence of unsigned objects conflicts with a basic tenet of CCN. This is not the case for the latest version of CCNx. CCNx 1.0 adopts secure catalogs (called manifests), and its packet format supports unsigned content objects [142].

Using SCNs as described above might not be compatible with negative acknowledgement (NACK), especially Content-NACK (cNACKs) [55]. These are a special type of empty content

[11] Even if a content is fetched by SCN, it can still be signed.

objects sent by producers in response to interests requesting non-existing content. Although it might sound counterintuitive to request a content object using SCN and receive a cNACK in return, this can indeed happen. Even if a consumer retrieves the SCN of a specific content from a legitimate source, e.g., a catalog, the content in question might no longer be available. For instance, a YouTube video's catalog can be delivered to consumers from a router's cache after YouTube has deleted the video.

If an interest containing an SCN results in a cNACK, this cNACK will be dropped by the first router. The is because the router expects a content object with a hash value matching the one indicated in the corresponding SCN. Fortunately, the issue can be dealt with via a minor modification: interests bearing SCNs should also (as a backup) adhere to IKB, i.e., reflect the producer public key, thus allowing routers to verify cNACKs.

4.4 Content Ranking

As mentioned above, some routers might not adhere to IKB. This can result in fake content being cached.[12] As discussed in Section 4.2.3, if border routers of consumer-facing AS-s always abide by IKB, most fake content will not be delivered to benign consumers. However, fake content in router caches can still be problematic. Therefore, we explore other approaches for mitigating content poisoning attacks in routers not adhering to the IKB rule. We present a lightweight ranking algorithm for cached content objects. Its objectives are: (1) probabilistically distinguish between valid and fake content objects based on observing consumers behavior, and (2) prioritize valid content when responding to consumer interests from the cache. Note that this approach can only be implemented in NDN for reasons that will become apparent below.

[12]Assuming that SCNs are not used to request content.

Our content ranking approach is premised on the fact that a consumer that detects a fake content, issues a new interest that excludes that fake content. Our hypothesis is that analyzing exclusion information could allow routers to rank cached content objects such that valid ones end up ranked higher. We achieve this by assigning each cached content object a rank and routers selecting the highest-ranked object in response to an interest. The rank is a numeric value between 0 and 1, where 1 is the highest. All cached content starts with the rank of 1, and, as time goes by, this value gradually decreases. This gives priority to newer cached content objects over older ones. In addition, the rank of a specific content depends on the number of times it was excluded and when. Also, we assign a lower rank to content with many recent exclusions, over fewer old ones. The reasoning is that few exclusions might always occur normally because even valid content objects might not always satisfy consumers. Consumers might exclude certain content if it represents a wrong (e.g., old/stale) version. Finally, we harshly penalize content which has been excluded by interests arriving on multiple interfaces, since that would indicate higher likelihood of dissatisfaction.

We use the following equation to model content ranking degradation pattern:

$$r_{n|H(C)}(t) = e^{\frac{-t}{\alpha}} \tag{4.1}$$

where t is the age of content C ($t \in [0, t_{to}]$; t_{to} is content `ExpiryTime`) in the cache, and α is a factor determining the degradation speed of the content rate, i.e., how fast can a specific content rate drops from 1 to 0. $n|H(C)$ represents the full name of the content, including its hash as the last name component. Note that $r_{n|H(C)} \in [r_{n|H(C)}(t_{to}), 1]$. The larger the value of α, the faster the content's rank degrades.

We now define α. Let r_{to} be the rank of cached $n|H(C)$ when it expires (`ExpiryTime` time elapsed), provided that $n|H(C)$ is never excluded during its lifetime. r_{to} is a system parameter that can be determined by the network administrator. For a given freshness value

and desired value of r_{to}, the corresponding value of α can be computed using Equation 4.1. We denote this as α_{to} – the value of α that makes the rank of a non-excluded content object equal to r_{to}, when it expires. For now, we set $\alpha = \alpha_{to}$ as its initial value.

In the rest of this section, we discuss criteria that affect content ranking degradation. The proposed algorithm is compatible with the current NDN design – it requires no architectural modifications or coordination between routers.

4.4.1 Number of Exclusions

This criterion is based on the number of times a certain content object C with name n is excluded. Different versions of C (including fake ones) have the same name with different payloads, resulting in different content hash values. To distinguish between these identically-named content objects, we denote each distinct version as $n|H(C)$. We define the exclusion rate as the ratio of the number of exclusions for $n|H(C)$, denoted as $E_{n|H(C)}$ to the total number of requests for C, denoted as Q_n. Exclusion rate is expressed as $R_{n|H(C)} = E_{n|H(C)}/Q_n$. We consider the ratio as more meaningful than the sheer number of exclusions. The reason is that consumers can exclude not only fake, but also incorrect content objects.[13] We assume that fake content objects tend to have much higher exclusion rates.

We want to assign a higher rank to objects excluded less. In other words, the more exclusions for $n|H(C)$, the sharper its degradation factor. If $n|H(C)$ is never excluded, the rank degradation pattern is the slowest from 1 to r_{to}.

Therefore, α is negatively affected by $R_{n|H(C)}$, i.e., $\alpha = \alpha_{to} - \left(R_{n|H(C)} \times \alpha_{to}\right)$. Thus, Equation 4.1 can be rewritten as:

$$r_{n|H(C)}(t) = e^{\frac{-t}{\alpha_{to} - \left(R_{n|H(C)} \times \alpha_{to}\right)}} \tag{4.2}$$

[13] An *incorrect* content object is an otherwise valid content object that does not satisfy the consumer.

4.4.2 Time Distribution of Exclusions

In order to give more weight to newer exclusions, we factor the time of exclusion into the ranking algorithm. We define exclusion influence $i_{n|H(C)}$ as the variable reflecting the time that the router should wait before the effect of the latest exclusion attempt for $n|H(C)$ is assigned the minimal weight. This is computed as:

$$i_{n|H(C)}(t_e) = 1 - e^{\frac{-t_e}{\beta}} \tag{4.3}$$

where t_e is time elapsed since the last exclusion and β is a factor determining the influence pattern. β reflects how soon the latest exclusion attempt is assigned a minimal weight when computing the content rank. Note that $i_{n|H(C)}(t_e) \in [0, 1]$, where $i_{n|H(C)}(t_e) = 1$ means that the latest exclusion has minimal effect on ranking.

The larger the value of β, the more time should elapse before minimally weighing the latest exclusion. This time is denoted as t_{mw}. In other words, if β is large, the latest exclusion has a strong influence on the content rank. Such strong influence results in a smaller rank value. Given t_{mw}, which can be set by the network administrator, and setting $i_{n|H(C)}(t_e) = 1$, the corresponding β can be computed using Equation 4.3. We denote it as β_{mw}.

We can now modify Equation 4.2 to include the exclusion influence factor:

$$r_{n|H(C)}(t) = e^{\frac{-t}{i_{n|H(C)}(t_e) \times \left(\alpha_{to} - \left(R_{n|H(C)} \times \alpha_{to}\right)\right)}} \tag{4.4}$$

If $i_{n|H(C)}(t_e) = 1$, the rank of $n|H(C)$ is only affected by $R_{n|H(C)}$.

4.4.3 Excluding Interfaces Ratio

This criterion considers the number of interfaces on which exclusions arrive. A higher number indicates higher dissatisfaction with a specific content. This can be used to lower the rank of that content. Given the following notations:

- f_n – total number of interfaces of a given router.
- $f_e \in [0, f_n]$ – number of interfaces on which the router received interests excluding $n|H(C)$.
- $f_s \in [1, f_n]$ – number of interfaces on which the router previously served $n|H(C)$. (Note that f_s can not be zero, since for ranking to exist, the corresponding content must have been requested and served on at least one interface.)
- $e_{n|H(C)}$ – ratio of number of interfaces on which the router served $n|H(C)$ without receiving any exclusions, to the number of interfaces on which $n|H(C)$ is served.

$$e_{n|H(C)} = \begin{cases} \frac{f_s - f_e}{f_s} & \text{if } f_s \geq f_e \\ 1 & \text{otherwise} \end{cases} \tag{4.5}$$

$e_{n|H(C)} \in [0, 1]$ where 1 means that $n|H(C)$ was not excluded at all. However, f_e might actually exceed f_s. More generally, it is possible for a router to receive an interest excluding $n|H(C)$ on an interface where this content has not been served. This could occur for at least three benign reasons: (1) routing changes, (2) consumer mobility, and (3) cache eviction. The first two are self-explanatory, whereas (3) refers to the case when content was previously requested, served, cached, and then flushed, for any reason, including normal aging.

Based on the previous definition we can rewrite equation 4.4 as follows:

$$r_{n|H(C)}(t) = e^{\frac{-t}{e_{n|H(C)} \times i_{n|H(C)}(t_e) \times \left[\alpha_{to} - \left(R_{n|H(C)} \times \alpha_{to}\right)\right]}} \tag{4.6}$$

Substituting $e_{n|H(C)}$, $i_{n|H(C)}(t_e)$ and $R_{n|H(C)}$ values:

$$r_{n|H(C)}(t) = \begin{cases} \text{if } f_s \geq f_e & e^{\frac{-t}{\left(\frac{f_s - f_e}{f_s} \times (1 - e^{\frac{-t_e}{\beta mw}}) \times \left(\alpha_{to} - \left(\frac{E_{n|H(C)}}{Q_n} \times \alpha_{to}\right)\right)\right)}} \\ \text{otherwise} & e^{\frac{-t}{\left((1 - e^{\frac{-t_e}{\beta mw}}) \times \left(\alpha_{to} - \left(\frac{E_{n|H(C)}}{Q_n} \times \alpha_{to}\right)\right)\right)}} \end{cases} \tag{4.7}$$

4.4.4 Analysis

Equation 4.7 reflects the rank of each cached content object, based on three criteria: the number of exclusions, the time distribution of exclusion attempts, and the number of interfaces on which exclusions arrived.

We now show a comparison between ranking degradation patterns of five cached content objects:

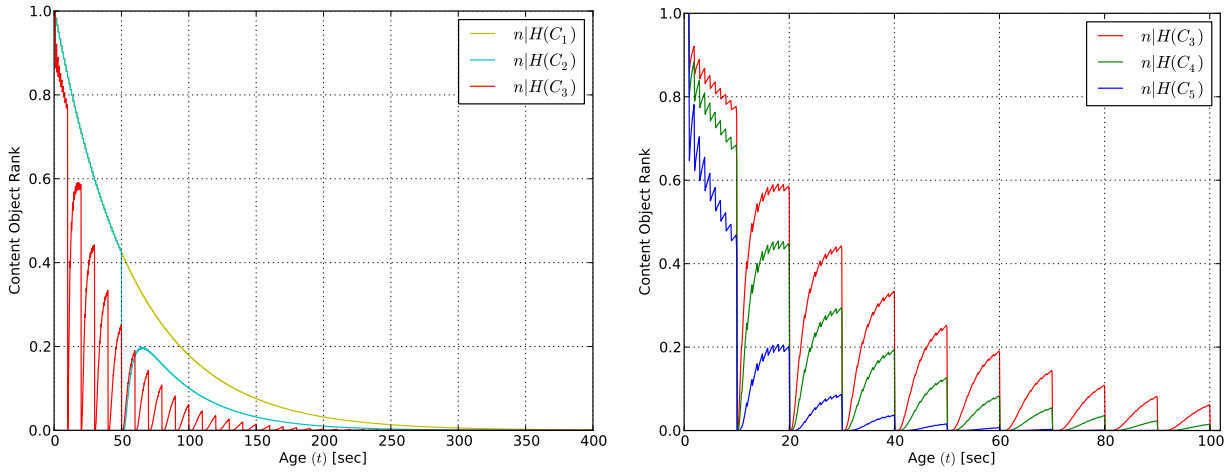
- $n|H(C_1)$: requested only once and never excluded throughout its entire lifetime in the cache.
- $n|H(C_2)$: requested only once, and at time $t = 50$ seconds, excluded once.
- $n|H(C_3)$, $n|H(C_4)$, and $n|H(C_5)$: requested once without exclusion and then excluded every 10 seconds. The difference between these three objects is their excluding interfaces ratio.

Table 4.1: Content objects parameters

Parameter	$n\|H(C_1)$	$n\|H(C_2)$	$n\|H(C_3)$	$n\|H(C_4)$	$n\|H(C_5)$
Content	C_1	C_2	C_3	C_4	C_5
Name	n	n	n	n	n
Digest	$H(C_1)$	$H(C_2)$	$H(C_3)$	$H(C_4)$	$H(C_5)$
t	$[0, 400]$, one sample every 100 $[msec]$				
freshness	400	400	400	400	400
r_{to}	0.001	0.001	0.001	0.001	0.001
Q_n	1	1 when $t \in [0, 50]$, and 2 when $t \in (50, 400]$	increased by one every 10 $[sec]$		
$E_{n\|H(C)}$	0	0 when $t \in [0, 50]$, and 1 when $t \in (50, 400]$	increased by one every 10 $[sec]$		
t_{mw}	400	400	400	400	400
t_e	∞	∞ when $t \in [0, 50]$, and increased by one every 1 $[sec]$ when $t > 50$	$[0, 10]$, increased by one every 1 $[sec]$, and reset every 10 $[sec]$		
f_n	4	4	4	4	4
f_e	0	0 when $t \in [0, 50]$ 1 when $t \in (50, 400]$	1	2	3

The parameters of Equation 4.7 used in our comparison differ for each object. They are summarized in Table 4.1.

The intuition for Equation 4.7 is that $n|H(C_1)$ should have higher ranking at all times, followed by $n|H(C_2)$, then $n|H(C_3)$, $n|H(C_4)$, and $n|H(C_5)$. Figure 4.1 confirms this by showing ranking degradation patterns of these content objects. Figure 4.1(a) shows that, when $t <= 50$ seconds, both $n|H(C_1)$ and $n|H(C_2)$ have equal ranking values, which is higher than the other three excluded content objects. The ranking of $n|H(C_2)$ decreases after it is excluded at $t = 50$ seconds. Moreover, the repetitive pattern of $n|H(C_3)$ is justified because this content object is excluded every 10 seconds. Once an exclusion occurs, ranking drops to close to 0, and starts increasing again according to Equation 4.3. On the other hand, Figure 4.1(b) compares between $n|H(C_3)$, $n|H(C_4)$, and $n|H(C_5)$ in a shorter time window of 100 seconds. It demonstrates the effect of varying f_e. For instance, $n|H(C_5)$ is excluded on 3 different interfaces, while $n|H(C_3)$ is excluded on only one. Thus, the former has a lower ranking value.



(a) $n|H(C_1)$: never excluded, $n|H(C_2)$: excluded once when $t = 50$ seconds, and $n|H(C_3)$: excluded every 10 seconds

(b) $n|H(C_3)$: excluded on 1 interface, $n|H(C_4)$: excluded on 2 interface, and $n|H(C_5)$: excluded on 3 interface. All three objects are excluded every 10 seconds

Figure 4.1: Content object ranking comparison

Based on prior definitions and the analysis of the content ranking algorithm, we conclude that newer content objects are ranked higher than old ones. This is an intentional design feature to give newer content priority in distribution and a chance for timely dissemination. Moreover, in cases of none or few malicious consumers, newer content objects are less likely to be fake, since a router always tries to satisfy an interest from its cache. As long as a router's cache contains a valid version of content, consumers are served that content and are unlikely to send another interest excluding valid content. Therefore, the only case when a fake content can be cached longer than a valid one is when malicious consumers work against the ranking algorithm by excluding valid content, or, explicitly request fake content. We elaborate on this in Section 4.5. Our approach is effective even against powerful distributed attackers as long as benign consumers are not outnumbered by malicious ones.

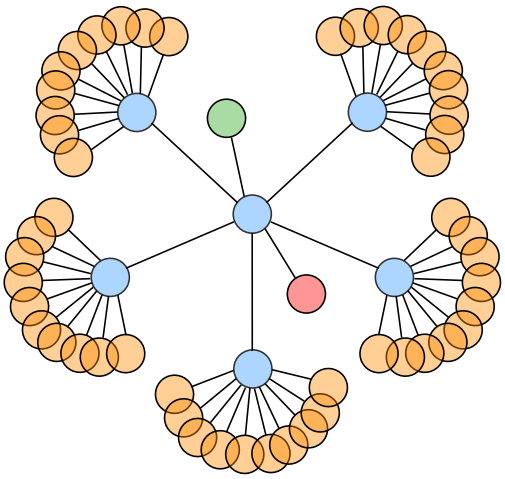


Figure 4.2: Tree-based topology - orange: consumer, blue: router, green: producer, red: Adv

4.5 Experiments and Results

ndnSIM 1.0 [23] is a simplified implementation of NDN architecture as a NS-3 [15] module for simulation purposes. To verify correctness and practicality of the proposed ranking algorithm, we extend ndnSIM to incorporate content ranking. This section describes simulation scenarios and experiments, and then discusses the results. We start by introducing some terminology:

- **Benign Consumers**: After receiving a fake content, they always exclude it in their subsequent interests for the same name.[14] They stop sending interest messages after receiving valid content.

- **Malicious Consumers**: When receiving a fake content, they consider it valid and do not exclude it. If an interest returns a valid content, they exclude it in all subsequent interests for the same name. The objective is to change statistics collected about exclusions to favor fake content.

[14]In our implementation, the maximum number of hashes of excluded content that can be included in an interest is 100. When that number is reached, a new hash added to the exclusion list replaces the oldest one.

Table 4.2: ndnSIM topologies parameters

Parameter	**Tree-based**	**DFN**		**AT&T**	
# of consumers	50	80	80	160	160
# of routers	6	30	30	132	132
# of producers	0	0	0	0	0
Cache replacement policy	LRU	LRU	LRU	LRU	LRU
Simulation time [sec]	400	400	400	400	400
Pre-populated fale content objects rate	99.9%	80%, 90%, 99%, and 99.9%	99%, and 99.9%	80%, 90%, 99%, and 99.9%	99%, and 99.9%
Pre-populated content objects freshness [sec]	400	400	400	400	400
Malicious consumer rate	0%, 2%, 4% 6%, and 10%	0%	0%, 1%, 3%, 5%, and 10%	0%	0%, 1%, 3% 5%, and 10%
Interest interval [millisecond]	$[100, 300]$	$[100, 300]$	$[100, 300]$	$[100, 300]$	$[100, 300]$

We measure how many benign consumers can retrieve a valid content and how fast they can do so when router caches are poisoned. The simulation starts with router caches pre-populated with fake versions of the target content. We vary the rate of fake pre-populated content, as discussed below. Table 4.2 shows the parameters. To assess the performance of the content ranking algorithm, we consider that all topologies used in our experiments consist of a single AS. Moreover, cache is disabled at edge network routers to allow better analysis of content ranking in core network routers only (in this case in core AS routers).

4.5.1 Tree-based Topology

The tree-based topology is illustrated in Figure 4.2. It consists of 5 consumer-facing routers (each connected to 10 consumers), connected to a single backbone router. The topology also contains one producer and one adversary[15], both connected to the backbone router. This topology is common in distributing content from a single producer.

[15]We use the scenarios where the adversary is connected to the same backbone router with the producer in order to demonstrate how a strategically located adversary can easily distribute fake content into caches. However, we skip the initial phase of the attack and start with caches pre-populated with fake content.

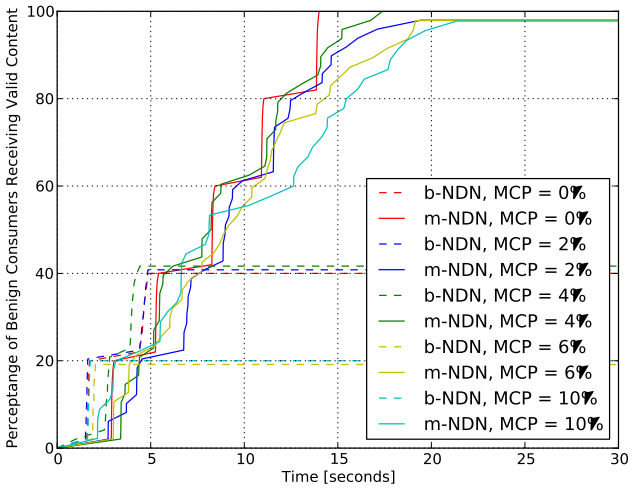


Figure 4.3: Tree-based topology with various malicious consumer rates (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, MCP: percentage of malicious nodes in consumer population)

We first pre-populate routers with 1000 versions of the same content. Only one of them is valid, i.e. the pre-populated fake content rate is 99.9%. We also consider different rates of malicious consumers, 0%, 2%, 4%, 6%, and 10%. Figure 4.3 shows the results. When ranking is not used, the network can not reach a state where all benign consumers receive valid content. The reason is that consumers can only exclude up to 100 content objects in interest messages. When all benign consumers receive valid content, they stop issuing interests. We call this state full convergence. For all malicious consumer rates simulated, applying ranking always leads the network to full convergence, even though it takes longer for higher malicious consumers rates.

4.5.2 DFN Topology

We now proceed to a more complex network topology. The DFN network, Deutsches ForschungsNetz (German Research Network) [4, 5], is a German network developed and deployed for research and education purposes. It consists of several connected routers po-

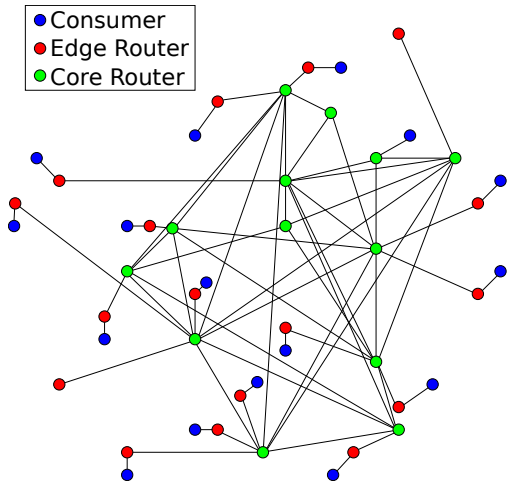


Figure 4.4: DFN topology - each edge router above is connected to 5 NDN consumers

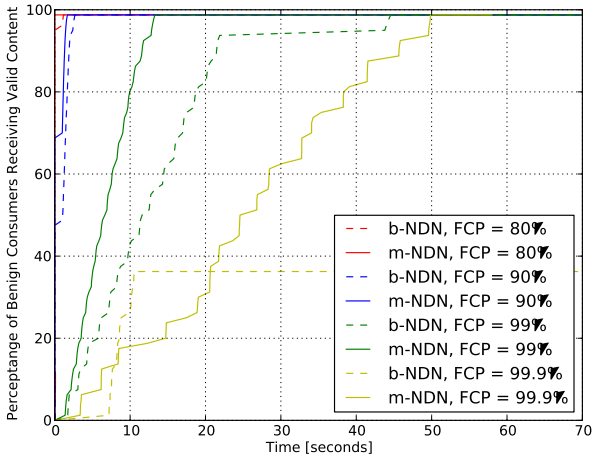
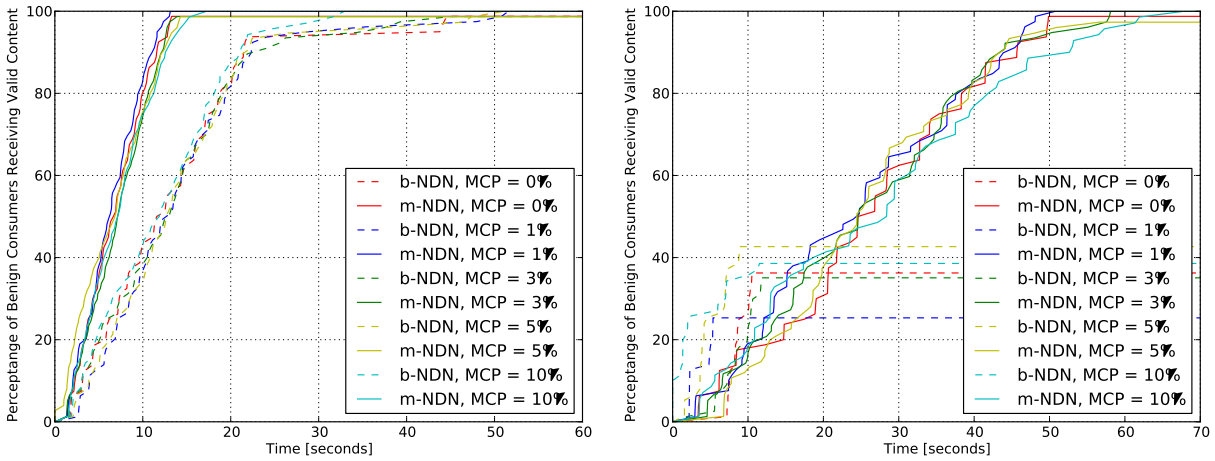


Figure 4.5: DFN topology results with different rates of pre-populated fake content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, FCP: percentage of pre-populated fake content objects)

sitioned in different areas of Germany, as shown in Figure 4.4. Our implementation of the DFN network consists of 30 routers and 80 consumers.

We ran two sets of experiments.

1. All routers are pre-populated with different rates of fake content objects, 80% (1 valid and 4 fake content objects), 90% (1 valid and 9 fake objects), 99% (1 valid and 99 fake objects), and 99.9% (1 valid and 999 fake objects). Figure 4.5 shows the results of this experiment. The network reaches full convergence in all cases (with different



(a) 99% pre-populated fake content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, MCP: percentage of malicious nodes in consumer population)

(b) 99.9% pre-populated fake content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, MCP: percentage of malicious nodes in consumer population)

Figure 4.6: DFN topology results

convergence speed) except when ranking is not applied for pre-populated fake content rate of 99.9%. This is again because consumers can exclude up to 100 content in interests. Our implementation of the ranking algorithm using ndnSIM is not fully optimized, which justifies heavy overhead imposed by the algorithm for 99.9% rate of pre-populated fake content.

2. We vary malicious consumers rate (0%, 1%, 3%, 5%, and 10%) for two rates of pre-populated fake content objects, 99% and 99.9%. Figures 4.6(a) and 4.6(b) show the results. In Figure 4.6(a), the ranking algorithm allows faster full convergence, while in Figure 4.6(b), full convergence can not be achieved without using the ranking algorithm.

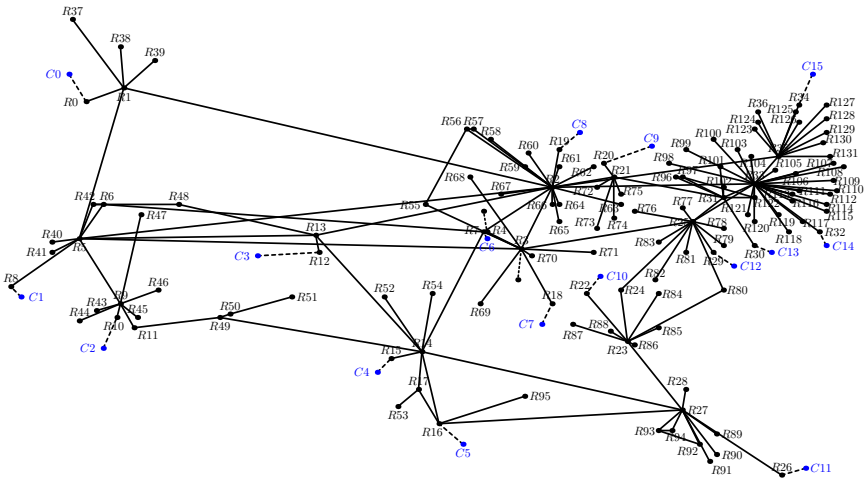


Figure 4.7: AT&T topology - each consumer above represents 10 NDN consumers

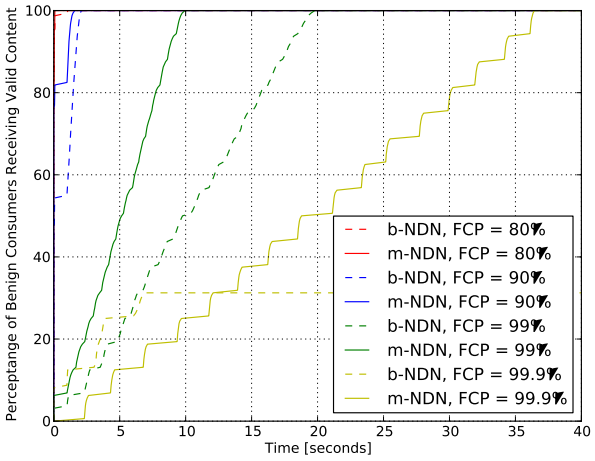
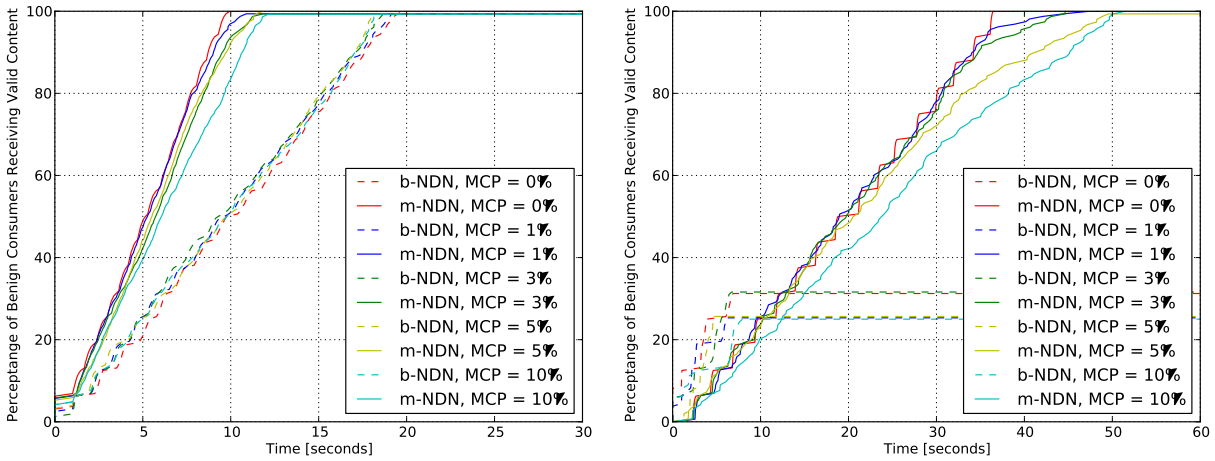


Figure 4.8: AT&T topology results with different rates of pre-populated fake content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, FCP: percentage of pre-populated fake content objects)

4.5.3 AT&T Topology

We now assess the performance of the ranking algorithm in a much bigger topology, the AT&T backbone network [54] shown in Figure 4.7. This network consists of over 130 routers and 160 consumers.

We conducted two sets of experiments. The analysis is similar to that in Section 4.5.2.



(a) 99% pre-populated content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, MCP: percentage of malicious nodes in consumer population)

(b) 99.9% pre-populated content objects (b-NDN: basic NDN with LRU cache replacement, m-NDN: modified NDN with routers implementing our ranking algorithm, MCP: percentage of malicious nodes in consumer population)

Figure 4.9: AT&T topology results

1. We pre-populate routers with various ratios of valid to fake content objects, 80%, 90%, 99%, and 99.9%, and ran experiments with and without ranking. Figure 4.8 shows the results.

2. All routers are pre-populated with either 99% or 99.9% fake content rates. Figures 4.9(a) and 4.9(b) show the results of this experiment for various malicious consumer rates (0%, 1%, 3%, 5%, and 10%).

Results from simulations on the AT&T network topology show that the content ranking algorithm improves quality of service and resilience of the network against content poisoning attacks, even if caches of all routers in a relatively large topology are poisoned.

4.5.4 Performance Analysis

To facilitate content lookup, cache can be implemented as a hash table with content name, except the last name component (the digest), serving as the key. Each key points to a priority

queue [111] where the first element contains the content with the highest rank. If an interest does not carry an exclusion filter, content lookup requires $\mathcal{O}(1)$ operations. Otherwise, a lookup takes up to $\mathcal{O}(k \log n)$ operations (if implemented using maximum heap [35]), where n is the number of content objects in the priority queue and k is the number of exclusions. This is because the rank of each excluded content objects needs to be recalculated and the priority queue needs to be rearranged. In terms of storage, less than 50 extra bytes per cache entry is required to store parameters needed to calculate the rank according to Equation 4.7.

4.6 Content Trust in Practice

CCN was designed as a candidate next-generation Internet architecture. In order to provide a smooth and successful transition path, it must contend with application-specific requirements, such as trust. In this section, we discuss how our trust architecture might be applied in practice to mitigate content poisoning attacks. We start by identifying traffic types and network topologies.

4.6.1 Traffic Types

We anticipate two main types of traffic.

Content Distribution

This corresponds to client-server communication which accounts for well over 90% of current Internet traffic [17]. Since most requested content is static, creating secure catalogs is straightforward. Consumers request catalogs and then use SCNs to request desired content. We consider two common sub-types of content distribution traffic:

- Multimedia (not live) Streaming: A large content split into several segments with different names (as described in Chapter 2). If a catalog containing SCNs of all segments is provided, consumers can use these names in subsequent interests to retrieve all segments.

- Internet Browsing: We anticipate that most HTML files would fit into a single content object [66]. A typical HTML file contains links to other static and dynamic content, such as images, audio or other HTML pages (sub-pages). While rendering HTML files, Internet browsers parse all reference links and download corresponding content. Therefore, if an HTML file uses SCNs as references, it can be viewed and treated as a catalog. Note that SCNs can only be used for static content, since the hash of dynamic (e.g., generated upon request) content can not be known *a priori*.

Internet browsing provides a good example of content that can be requested by adhering to IKB rule or using SCNs. Suppose that a web page A contains a reference link to sub-page B, and this link is expressed using an SCN. Once a consumer requests and obtains page A, the client browser can request B using the appropriate SCN in A. At the same time, other consumers might wish to directly request page B. This can be done using the IKB rule. Note that, for obvious reasons, SCNs can not be used with HTML pages (or any other content) with circular references, e.g., $A \leftrightarrow B$.

Interactive Traffic

Another major traffic type is interactive communication, where content is generated in real time, or on demand. Applications such as voice/video conferencing, remote terminal access and online gaming fall into this category. Such applications generally benefit from network caching only in cases of packet loss. Re-transmitted interests are likely to be satisfied by the first hop router. Obviously, the use of large catalogs for interactive real-time traffic is neither sensible nor feasible. Instead, this traffic should be handled using the IKB rule.

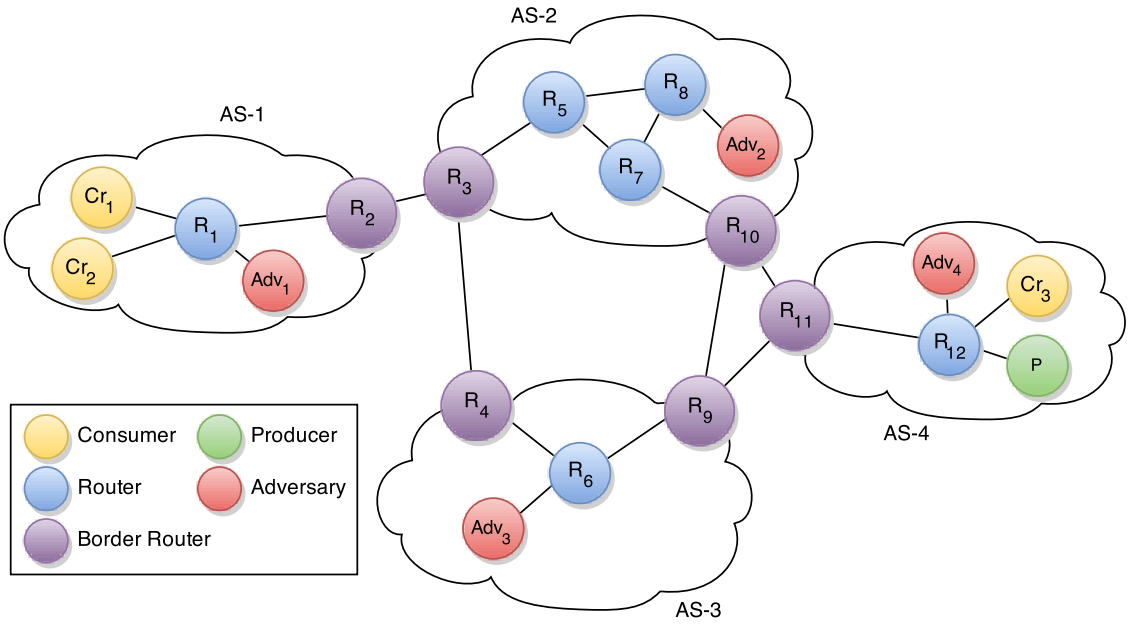


Figure 4.10: A simplified diagram of the Internet

4.6.2 Network Topologies

As mentioned in Section 4.2.3, not all CCN routers might be willing, or have the resources, to adhere to the IKB rule. However, consumer-facing (edge) AS-s should configure their border routers to use IKB. Figure 4.10 shows four AS-s: AS-1 and AS-4 are stubs while AS-2 and AS-3 are transit. Stub AS-s can be seen as consumer-facing and transit as core AS-s. We discuss how each content poisoning mitigation technique should be used in order to achieve the goals set in Section 4.1.3.

Recall that the content ranking algorithm can only be used in NDN, and not in CCNx due to the latter's lack of exclusion filters. However, CCNx does not need to implement content ranking, or any similar techniques, because it implements a more strict rule for satisfying interests from the cache. Specifically, routers *must not* serve cached content objects unless these objects are authentic, e.g., their signature is verified [142]. In other words, CCNx does not allow benign routers to reply with fake content objects from their cache.[16]

[16]We do not consider the case of malicious routers replying with fake content and leave it as follow-on work.

To minimize the effects of content poisoning attacks, at least AS-1 border router R_2, and AS-4 border router R_{11} should implement the IKB rule. This, however, does not guarantee that poisoned content objects are not delivered to consumers if such objects are cached in (or served through) R_1 and R_{12}. The adversary would be located inside AS-1 and AS-4, e.g., Adv_1 and Adv_4. Therefore, these two routers should implement the content ranking algorithm described above.

Moreover, if only R_2 and R_{11} use the IKB rule, fake content can still be cached in routers $R_3 \rightarrow R_{10}$. As mentioned in Section 4.4, such content would consume cache storage, and, bandwidth, before it gets detected and discarded. For instance, a fake content object cached (and served) by R_7, might travel all the way to R_2 before being detected. These issues can be avoided only by verifying content signatures, i.e. implementing IKB.[17] If bandwidth is more important than cache storage, all caching routers should implement content ranking.

Furthermore, attack impact varies based on adversary's locations:

- Adv_1 and Adv_4: by continuously poisoning the caches of R_1 and R_{12}, adversary can ensure that fake content is delivered to consumers if the two routers do not implement any content poisoning mitigation methods.
- Adv_2 and Adv_3: can only poison (directly or indirectly) caches of R_3 to R_{10}, consuming storage and bandwidth resources. Since caches are limited resources, fake content objects can cause the eviction of valid ones.

Even though mitigation of content poisoning attacks is not free, IKB and content ranking allow for a flexible trade-off between computational, storage, and bandwidth overheads at different locations in the network.

[17] Even if content is requested using SCNs, malicious consumers can trigger caching fake content. However, such content will never be delivered to benign consumers, assuming that they always request valid one.

Chapter 5

Accounting

As mentioned in Chapter 2, in-network caching reduces end-to-end latency and network bandwidth utilization. Also, as discussed in Chapters 3 and 4 such benefits result in significant privacy and security challenges. In this chapter, we address the specific problem of accounting for content usage, which is complicated, in large part, by caching. Since an interest can be satisfied by a router, it might not reach the producer. Consequently, producers might only receive a small fraction of all interests for a given piece of content. At the same time, the number, sources, and timing of interests represent important information that could be needed by the producer for accounting purposes. Even if the timing and the number of interests were somehow communicated to the producer, interest sources would remain unknown since CCN inherently lacks consumer information, e.g., source addresses, in interests.

Furthermore, if and when CCN is deployed in the real world, router cache space will likely be treated as a valuable (and even premium) resource. Thus, a mechanism is needed for reporting cache hits to content producers and router owners in order to inform them about content usage. To be viable, such a mechanism must incur minimal bandwidth, computation,

and storage overhead. It also must be secure in order to prevent false cache usage reporting. To this end, we design a secure lightweight network-layer accounting mechanism for CCN. It is applicable to both CCNx and NDN[1]. Our contribution is three-fold:

- Identification of, and motivation for, features needed for CCN accounting and security thereof.
- First comprehensive technique for content and cache usage accounting, with varying levels of consumer, router, and producer involvement.
- Analysis of performance and security trade-offs.

5.1 Accounting in CCN

As mentioned earlier, router caches present a major challenge for accounting in CCN. In particular, if interests are satisfied by caches, how can a content producer collect information about the popularity of, or demand for, its content?

In this section, we discuss design elements for accounting in CCN. For the time being, we do not take into account security or privacy considerations. Assuming benign (well-behaved) consumers, routers, and producers, our *initial* goal is to determine the minimal functionality needed by all CCN entities to facilitate *correct* accounting. In doing so, we consider three types of accounting information:

- *Individual* information is tied directly to a specific consumer. An example might be the number of times a particular consumer requested a particular content. It provides linkability between consumers and content they obtain. It also requires revealing consumer identities, at least to the producer.

[1]Support for NDN requires minor packet format and protocol changes.

- *Distinct* information is functionally equivalent to individual information except that consumer identities are not revealed.

- *Aggregate* information represents collective or combined values over a set of consumers. For example, it might include the number of times a particular piece of content was requested from a specific geographic location or from a particular ISP. Aggregate information enables some degree of consumer privacy.

We believe that these three types correspond to most accounting information needed in any real-world CCN application and focus on them in the remainder of this chapter. We also recognize that accounting should not be mandatory for all content. Some producers might not care about the popularity of *any* of their content while others might need accounting information only for *some* of their content. We refer to content for which producers desire such information as *accountable* content.

Another important design dimension is whether accounting information is reported in real time (online) or offline. In the latter case, a network management protocol can be envisaged whereby an AS-level accounting server periodically collects cache hit logs from its routers and reports the results to producers. This kind of accounting seems viable. However, it involves a potentially significant delay in notifying producers about demand for their content. This might be unacceptable for content for which real-time demand information is needed. Since real-time accounting presents a more difficult challenge, we concentrate on it in the rest of this chapter.

5.1.1 Counting Cache Hits vs. Content Requests

Another variable in supporting CCN accounting is exactly what is being counted: instances of cache hits or instances of requested content being served to the consumer? A cache hit

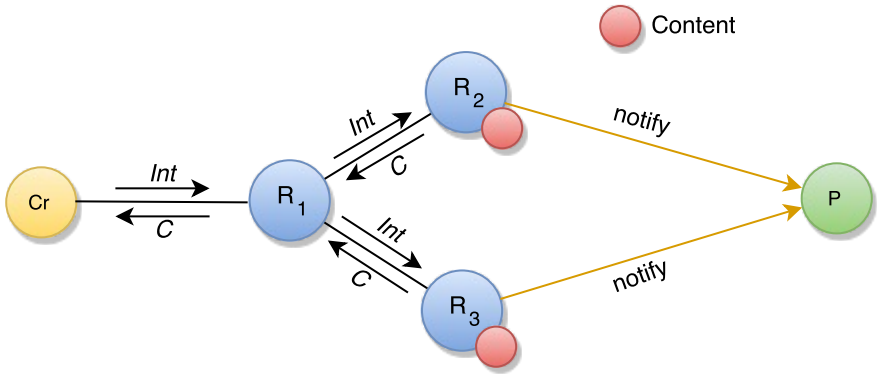


Figure 5.1: Cache hits vs. content requests

occurs when a router finds requested content in its cache. We assume that accounting for cache hits is only relevant for routers, i.e., network elements.[2] An instance of content being served occurs when a cache hit takes place *and* the content is actually delivered to a *single* consumer.

It might seem that these two types of events are the same, i.e., a content is served once for every cache hit and vice-versa. However, this is not the case in CCN. Whenever a router receives an interest, it may choose to multicast (forward) it out on multiple interfaces. This behavior is officially allowed since a router's FIB can express multiple next hops for a given name prefix. One practical reason for allowing it is to facilitate fast(er) content retrieval. However, it also complicates accounting. Consider the scenario in Figure 5.1. Suppose that a consumer's interest for content C is received by router R_1. The latter forwards it to two upstream routers R_2 and R_3, based on its FIB. Both R_2 and R_3 have C in their respective caches and each replies to R_1 with its cached version. Assuming that R_2's copy of C is the first to reach R_1, the latter forwards C downstream and flushes the appropriate PIT entry. When R_3's copy of C arrives, R_1 discards it since it does not refer to an existing PIT entry. If both R_2 and R_3 inform C's producer P about a cache hit, P would incorrectly assume that C was requested twice. Even though C is served twice by two distinct routers, a single consumer received one copy of C.

[2]Accounting for cache hits in content stores or at producers themselves is out of scope.

In general, the number of cache hits might not match that of content requests. This occurs because there is no way to distinguish among multiple interests issued for the same content.[3] In other words, if consumers Cr_1 and Cr_2 issue interests for C at different times, their interests would be identical. Moreover, even if C is not cached, i.e., interests for it reach P, and if Cr_1 and Cr_2 issue interests for C at roughly the same time, P would be unable to distinguish between this case – when two consumers ask for C – and the case in the scenario above – when one consumer asking for C and R_1 decides to multicast the interest upstream. Note that the number of cache hits is two in both cases, while the number of content objects served is two and one, respectively.

The reason for supporting both types of accounting is intuitive: a producer might need to know the exact demand for its content, whether on aggregate, distinct, or individual basis. Separately, a producer might need to know which routers experience cache hits for its content. The latter could be used to reconcile billing by the producer for cache usage.

Finally, even though accounting for cache hits and content requests is not the same thing, we naturally would like to use the same mechanism as much as possible to provide both. Therefore, in the rest of this chapter, and unless otherwise mentioned, we use the term *accounting* to refer to both accounting for cache hits and content requests.

5.1.2 Accounting via Content Access Control

One potential accounting approach is to use encryption-based access control for content. Suppose that producers encrypt all accountable content, and decryption keys – which, in CCN, are represented as content objects with well-defined names – are configured not to be cached (i.e., their `ExpiryTime` values are 0). Even if interests requesting such content are satisfied from router caches, consumers would need to separately issue interests requesting

[3]NDN interests carry a random nonce used for interest loop detection, which can be helpful in this distinction. However, CCNx interests do not carry nonces.

decryption key(s). Such interests would bypass router caches and reach the producers, thereby enabling per-request accounting.

With content encryption, the desired type of accounting might dictate how key-requesting interests should be generated. For example, in case of individual accounting, consumers can include some kind of consumer-specific data in interests when keys are requested. Such data allows producers to link these interests to specific consumers. However, if only aggregate accounting is required, interests requesting keys do not have to carry any consumer-specific data. Such interests need to carry some kind of a nonce so that the producer can distinguish between the cases of (1) receiving two interests from two different consumers, and (2) receiving duplicates that stemmed from a single interest (issued by one consumer) which was multicasted by some downstream router.

Accounting via content encryption has two primary advantages: (1) it is transparent to the network and (2) it does not require any new features or message types. However, despite its apparent simplicity, it is inefficient. All accountable content needs to be encrypted, and keys need to be requested and distributed separately. Thus, content is obtained by issuing at least two interests – one for the content and one for the key(s).

We believe that an ideal accounting mechanism should efficiently work for all accountable content. That is, it should not require a consumer to issue more than a single interest for accountable content. Also, content accounting should be distinct from content access control.

5.1.3 Accounting via Push Interests

The accounting approach proposed in this chapter is based on real-time reporting. The key element is a new message type called a *push* interest, denoted as *pInt*. Its main purpose is to inform the producer that its content has been requested, and a cache hit occurred.

Structurally, a *pInt* carries a name similar to a regular interest. However, the most important distinguishing feature of a *pInt* is that, unlike a regular interest, it does not leave behind any state in routers. A *pInt* referencing C is forwarded until it reaches the producer P, and no information about *pInt* is retained by any intervening router. A router forwards a *pInt* just as it forwards a regular interest with the exception that *pInt* messages are not multicast. This restriction is necessary to prevent producers from receiving duplicate copies of the same *pInt*.

A router R generates a *pInt* in two cases:

1. A regular interest for C is satisfied from R's cache. R generates a *pInt* referencing C and forwards it upstream towards P.

2. R receives a content C corresponding to a PIT entry. R forwards it on all downstream interfaces listed in that PIT entry. However, before flushing the entry, R generates a *pInt* that aggregates all collapsed interests. (These aggregation details are discussed in Section 5.1.4.) Note that *collapsed* refers only to those interests that were not originally forwarded upstream. This is because the interest forwarded upstream presumably already reached P, or triggered its own *pInt* via case 1 above.[4]

In summary,

A *pInt* is generated when a (cached) content is used to satisfy an incoming or a collapsed interest.

The above is summarized in Algorithm 3. In order for P to inform routers about what content requires accounting, we also introduce a new content header flag `ACCT`, which reflects one of the following:

[4]For example, suppose that R receives an interest for C on interfaces: 2, 3, 5, and 6. Regularly, only the first one is forwarded, say, on interface 9. Others are collapsed into the same PIT entry. Now, when C arrives on interface 9, it is forwarded on all 4 incoming interfaces. However, R generates a *pInt* that reflects only three interfaces: 3, 5 and 6.

Algorithm 3 *pInt*-Generation

```
1: Input: C[N, ACCT], Int[N, PL], R_id
2: pInt.Name := C.N
3: pInt.Type := C.ACCT
4: pInt.Origin := R_id
5: if C from local cache then
6:    pInt.Cdata := Int.PL
7:    pInt.Count := 1
8: else
9:    e = FindPITEntry(C.N)
10:   for each i in e/{Int} do
11:      pInt.Cdata := pInt.Cdata || i.PL
12:      pInt.Count := pInt.Count + 1
13:   end for
14: end if
15: Forward pInt according to the FIB
```

1. `NONE`: (default) no accounting information.

2. `AGGREGATE`: aggregate accounting information.

3. `DISTINCT`: distinct accounting information.

4. `INDIVIDUAL`: individual interest-level accounting.

Whenever a cache hit occurs, routers behave the same in cases 2, 3, and 4. The only difference is when a content arrives and a router has a number of previously collapsed interests for that content. In case 2, a router generates a *pInt* with the count of collapsed interests for a given content. In cases 3 and 4, a router reports the *actual* interests, which can optionally be bundled into a single *pInt*.

5.1.4 *pInt* Format and Features

We now describe *pInt* messages format which is very similar to CCNx interests described in Section 2.1:

- `Name`: copied entirely from `Name` field in the interest (or PIT entry) that triggers a *pInt*.
- `Type`: indicates whether this *pInt* is for aggregate, distinct, or individual accounting.
- `Origin`: identifies the router that generates the *pInt*, e.g., the router's prefix (if available) or public key digest.
- `Count`: set to 1 in the case of a cache hit, or the number of interfaces *minus one* on which the content object was forwarded downstream, if interest collapsing occurred.
- `Cdata`: a random nonce or consumer-specific data used by producer for different purposes based on the accounting type required (i.e., distinct or individual). If `Count`>1, `Cdata` is a sequence of `Count` consumer-specific data values culled from corresponding interests. Such data can be carried in CCNx interest `Payload` field.

Semantics of the `Cdata` field depend on the type of required accounting information. We discuss below consumer-specific data requirements for each accounting type. As stated above, aggregate accounting for cache hits does not require `Cdata` to be present.

Aggregate

The problem in this type of accounting is that producers do not have the means to distinguish between the cases where received interests (or *pInt* messages) with the same name are multicast by routers or generated by several distinct consumers. However, if consumers include random nonces and timestamps as consumer-specific data, this distinction can be achieved.

Distinct

`Cdata` needs to reflect the uniqueness of interests. This can be achieved via consumer-provided nonce and timestamp combination. The nonce format is application-specific and can range from a random number to the hash of the content name and the timestamp. Note that the same knowledge provided to the producer in the distinct accounting case can also

be attained using aggregate accounting type if `Cdata` reflects the uniqueness of interests. However, we keep the distinction between these two types for ease of classification.

Individual

`Cdata` needs to reflect identities of consumers that issued interests. This can take the form of:

1. Consumer public keys (or prefixes) or their digests. This form reveals consumer identities to all network entities – not only to producers.

2. Group public keys or their digests. A group can be an organization, autonomous system (AS), or a geographical region. In this case, the group identity is revealed rather than that of individual consumers.

3. Unique consumer identifiers (i.e., pseudonyms). Although this does not violate consumer anonymity, such identifiers need to be assigned to consumers by producers or a trusted third party before any interests are issued. This form of `Cdata` also allows interest linkability.[5]

4. Consumer identity (using any of the three previous forms) with nonces and timestamps. This form of `Cdata` allows producers to know which consumers request what content, as well as how many times such requests are made.

Each of the above incurs different overhead for consumers and producers. However, router overhead is only slightly affected. This is because routers simply populate `Cdata` of generated *pInt* messages using information contained in the `Payload` field of the corresponding interests, regardless of how consumer-specific data is generated. In other words, routers are oblivious to the accounting type used. Also, note that the choice of which form to use is an application-specific issue. We do not mandate one technique.

[5]Interest linkability is defined as the ability of an eavesdropper (observer or adversary) to learn whether two captured interests are issued by the same consumer.

5.1.5 Accounting Correctness

We define accounting correctness as follows.

Definition 5.1 (Correctness)**.** *An accounting technique is* correct *if it accurately reports cache hit and content request information to the producer, assuming that all participants faithfully follow the rules (i.e., no malicious behavior) and there are no transmission errors, no packet loss, and no node failures that affect accounting-relevant traffic.*

We also define probabilistically correct accounting as follows.

Definition 5.2 (Probabilistic correctness)**.** *An accounting technique is* probabilistically correct *if it is correct with a negligible probability of error, i.e., inaccurate or false information is reported.*

We now informally demonstrate correctness of each proposed accounting technique (individual, distinct, and aggregate) for the two cache hit and content request cases.

Cache Hit

A router R generates a *pInt* for every cache hit on accountable content objects. Since all routers (including R) on the path to producer P forward *pInt* messages according to their FIB entries, all such messages are guaranteed to be delivered to P. This provides accurate cache hit counts for accountable content objects. This argument holds for all three types of accounting. The only difference is that `Cdata` fields of *pInt* must contain appropriate consumer-specific data in some accounting types.

Content Request

Although *pInt* messages provide *correct* individual, distinct, and aggregate accounting for cache hits, they only offer *probabilistically correct* accounting for content requests. As stated above, consumers can include nonces and timestamps in interest `Payload`. This information would allow producers to distinguish between cases where received interests (or *pInt*

messages) with the same name are multicast by routers or generated by several distinct consumers.

5.2 Security Considerations

Thus far, we assumed that all entities involved in accounting are benign. However, this assumption is clearly unrealistic in practice. In this section, we identify requirements for *secure* accounting in CCN. We also demonstrate that some attacks can not be prevented or even detected without additional and non-negligible cryptographic overhead. Furthermore, secure accounting involves a trade-off between security and overhead for consumers and producers. As we show in this section, routers are unaffected.

5.2.1 Adversary Model

The anticipated adversary Adv is a *malicious router* generating *pInt* messages for bogus interests when individual accounting is required. In other words, Adv tries to inflate individual accounting information for both cache hits and content requests. For now, we assume that consumers behave honestly. We consider malicious consumers later in Section 5.3.

For completeness, we identify certain other attacks and justify their exclusion from the discussion below.

- A router that (1) does not generate *pInt* messages when necessary, or (2) generates *pInt* messages without forwarding content downstream. Both cases can be reduced to packet loss. We do not address these attacks since this kind of misbehavior is very difficult to detect.

- A *consumer* that continuously generates interests to inflate accounting information. If aggregate or distinct accounting is required, the producer would be unable to detect such malicious behavior. On the other hand, if individual accounting is required, consumer-specific data can be used to detect continuous requests. However, this scenario can be reduced to Interest Flooding attacks [77], which is outside the scope of this chapter. A similar argument applies to distinct accounting information.

- An external attacker controlling the network can eavesdrop on, drop, or replay packets, including *pInt* messages. This attack is largely irrelevant if links are encrypted, which is a realistic assumption for adjacent routers. Also, in most cases, consumers and producers communicate to edge routers over secure link-layer channels.

- An adversary tries to inflate aggregate or distinct accounting information. This can not be prevented deterministically due to the likely usage of multicast forwarding strategies.

Based on these adversarial features, we only consider security of individual accounting information. We now define a secure accounting technique as follows.

Definition 5.3 (Correctness with adversary)**.** *An accounting technique is* secure *with respect to Adv if it is correct and all Adv malicious behavior can be detected.*

Strategies and requirements to combat this adversary are discussed in the following section.

5.2.2 Mitigating Forgeries and Replay Attacks

Section 5.1.3 mentioned several options for generating consumer-specific data. However, in order to prevent inflation attacks, such data must be unforgeable and resistant to replay attacks. We define *secure* consumer-specific data as follows.

Definition 5.4 (Secure consumer-specific data)**.** *Consumer specific data is* secure *if it can be authenticated by at least the producer, and is neither forgeable nor subject to replay attacks.*

Providing replay resistance can be accomplished if consumer-specific data carries a nonce r and a timestamp t. Thus, secure consumer-specific data Sec-CrSD assumes the following format:

$$\mathsf{Sec\text{-}CrSD} = \Big[\mathsf{CrSD}||r||t, \mathsf{f}_k\left(\mathsf{CrSD}||r||t||Int.N\right)\Big] \tag{5.1}$$

where CrSD is consumer-specific data formatted as described in Section 5.1.4[6], $\mathsf{f}_k(\cdot)$ is a function that computes an authenticated integrity check using a key k. The interest name in the $\mathsf{f}_k(\cdot)$ computation binds Sec-CrSD to the interest to which it is appended. This prevents Adv from using the same Sec-CrSD for generating multiple *pInt* messages with different names. $\mathsf{f}_k(\cdot)$ can be realized as a Message Authentication Code such as HMAC [106] (if consumers share keys with producers), or a digital signature function. Each alternative has well-known advantages and drawbacks. In addition to verifying $\mathsf{f}_k(\cdot)$, producers need to maintain a list of all received nonces within the current time window, for each accountable content.

Based on this discussion, we conclude that unforgeability and replay resistance can not be achieved unless *secure* consumer-specific data is used. This is not possible with aggregate or distinct accounting since consumer-specific data is not provided. One way to fix it is to include Sec-CrSD in all interests regardless of accounting type required. However, this introduces unnecessary overhead for both consumers and producers.

[6]Note that CrSD and `Cdata` are different. The former is generated by consumers and assigned to `Payload` field of the interest, while the latter is a field in a *pInt* and may contain none or many CrSD values.

5.2.3 Consumer Anonymity

We now consider privacy issues. Ideally, Sec-CrSD needs to be opaque to all network entities, except producers. Digital signatures, by their very nature, reveal the consumer's identity and MAC-s allow multiple interests to be linked since a key label must accompany a MAC value in order for producer to identify the key to verify a MAC. Both cases are detrimental to privacy.

We start by defining consumer-specific data indistinguishability, which is necessary to maintain anonymity among an arbitrary set of consumers.

Definition 5.5 (Consumer-specific data indistinguishability)**.** *Let Cr^a and Cr^b be consumers who each generate an interest for the same C and let CrSD^a and CrSD^b be consumer-specific data values for their respective interests. Let Adv be an eavesdropper (except C's producer) not directly connected to either Cr^a or Cr^b. Let the event of Adv learning the source of CrSD^a and CrSD^b be denoted as: $Adv_{\text{rev}}\left(\mathsf{CrSD}^a, \mathsf{CrSD}^b\right) = 1$. These two interests are* **indistinguishable** *if the probability of $Adv_{\text{rev}}\left(\mathsf{CrSD}^a, \mathsf{CrSD}^b\right) = 1$ is no better than a random guess. That is,*

$$\Pr\left[Adv_{\text{rev}}\left(\mathsf{CrSD}^a, \mathsf{CrSD}^b\right) = 1\right] \leq \frac{1}{2} + \epsilon(n),$$

for any negligible function ϵ and security parameter n.

We also assume that consumers know the producer's public key pk before requesting content (see Section 5.3). Let A-CrSD denote an anonymous consumer-specific data: $\mathsf{A\text{-}CrSD} = \mathsf{Enc}_{pk}\left(\mathsf{Sec\text{-}CrSD}\right)$, where $\mathsf{Enc}_{pk}(\cdot)$ is a public key encryption function using pk, and Sec-CrSD is formed as defined in Section 5.2.2.

To prevent Adv from learning that multiple interests are generated by the same consumer, their A-CrSD values should be indistinguishable. This can only be achieved if $\mathsf{Enc}_{pk}(\cdot)$ is a

CPA-secure public key encryption scheme, i.e., secure against Chosen Plaintext attacks [96]. In some encryption schemes, this is done by mixing in a random number (nonce) with every plaintext before encryption.

Theorem 5.1. *Assuming a CPA-secure public key encryption scheme* $\mathsf{Enc}_{pk}(\cdot)$*,* A-CrSD *format shown in Equation (1) guarantees* indistinguishability *of consumer-specific data with overwhelming probability.*

Proof. The proof of consumer-specific data indistinguishability follows from the proof of CPA-secure public key encryption scheme [96]. We only prove that A-CrSD generation guarantees negligible probability of nonce collision.

We assume individual accounting, and that f is the frequency in which consumers send interests to a specific producer during a specific time window w, where each consumer generates appropriate A-CrSD values in the interests. Let the number of interests sent be $s = f \times w$. We claim that the probability of any two A-CrSD-s in the set $\{\mathsf{A\text{-}CrSD}_1, \ldots, \mathsf{A\text{-}CrSD}_s\}$ being derived from colliding nonces is negligible in N, the length of the nonce in bits. Let this collision event be denoted as $\mathsf{Col}(\mathsf{A\text{-}CrSD}_i, \mathsf{A\text{-}CrSD}_j)$ for $i \neq j$ and $1 \leq i, j \leq s$. The probability of this event occurring can be computed according to the birthday paradox:

$$\begin{aligned}
&\Pr\Big[\mathsf{Col}(\mathsf{A\text{-}CrSD}_i, \mathsf{A\text{-}CrSD}_j) = 1 \ ; \ i \neq j \ , \ 1 \leq i, j \leq r\Big] \\
&= 1 - \left(\frac{2^N}{2^N} \times \frac{2^N - 1}{2^N} \times \cdots \times \frac{2^N - s + 1}{2^N}\right) \\
&= 1 - \frac{2^N!}{(2^N)^s \, (2^N - s)!} \\
&= 1 - \frac{s! \cdot \binom{2^N}{s}}{(2^N)^s}
\end{aligned} \tag{5.2}$$

Note that Equation 5.2 assumes that $s < 2^N$; otherwise, the collision probability is equal to 1 according to the Pigeonhole Principle. □

Since public key operations are relatively expensive, symmetric cryptography is the natural alternative, albeit, with its own problems. A-CrSD can be formed as: enc_k(Sec-CrSD), where $\mathsf{enc}_k(\cdot)$ is a CPA-secure symmetric encryption function and k is a consumer-producer shared key. This would require additional operations for managing such keys. In order for producers to quickly determine the key that correctly decrypts a given A-CrSD-s field, consumers should include a cleartext key identifier (e.g., a key label), which clearly violates interest indistinguishability. One way to avoid this exposure is for multiple consumers to share the same key with the producer, e.g., based on location or time. An extreme option is to maintain unique per-consumer keys without labelling and require the server to discover the correct key by "brute force". This poses some obvious issues, e.g., new and exciting DoS opportunities.

5.3 Individual Accounting in Practice

So far, we made some assumptions in the context of individual accounting:

1. Consumers know *what* accounting information is needed in order to issue an interest for a desired content object.
2. Consumers know the producer's public key *pk* used to encrypt Sec-CrSD.
3. Consumers behave honestly, i.e., for content that requires individual accounting information in CrSD, they supply *correct* required information.

The first assumption seems to be particularly problematic, especially, if a consumer has no prior relationship with a producer. However, there are at least two ways for consumers to learn what a producer expects in an interest. First, recall that CCN network-layer trust management, Chapter 4, requires the consumer to know the public key of the producer before

requesting content. This, in turn, means that the consumer must pre-fetch the producer's public key. It is easy to extend the producer's public key certificate pk to include accounting requirements for constructing interests for that producer's namespace. In addition, catalogs (or manifest [142]) can be extended similarly to contain accounting requirements. An alternative is for a consumer to "blindly" issue a trial interest for some random content in the namespace of a given producer. This interest would likely not adhere to the producer's accounting rules. In this case, the producer simply replies with a public key certificate that includes its accounting requirements.

Without such techniques, a consumer can not be expected to provide specific information in an interest. We, therefore, conclude that individual accounting necessitates an initial phase whereby consumers learn producer's CrSD requirements and pk for generating Sec-CrSD or A-CrSD values in interests. This initial phase would address assumptions (1) and (2) above. However, a misbehaving consumer can just ignore producer's requirements and pull content from router caches without providing correct accounting information to the producer, thus bypassing the accounting mechanism.

The main problem is that routers have no means to validate Sec-CrSD fields that arrive in interests referencing cached content. Allowing routers to do so is undesirable, because: (1) at least one cryptographic operation per interest would be needed, and (2) producers would need to inform routers about key(s) needed to validate Sec-CrSD, for each individually accountable content. The former is a new expense and a DoS attack opportunity, while the latter is a key management nightmare. Also, replay prevention would add further complexity.

We, therefore, conclude that assumption (3) is unrealistic in the presence of dishonest consumers. Consequently, individual accounting should be handled at the application layer.

5.3.1 Recommendations

Based on the above discussion, we now present some recommendations for CCN accounting.

First, if individual accounting information is needed, producers must simply set all content cache time to zero (0). This will force all interests to be routed to the producer. If an interest for content that requires individual accounting is received and the required accounting information is missing, producers should reply with a NACK [55] indicating consumer-specific data requirements for obtaining that content. The consumer can then re-issue an interest with the correct information. Since producers process all interests before responding with content, they can determine if a given interest for individual accountable content is valid and thus detect consumer misbehavior.

For aggregate and distinct accountable content, consumers should always include a random nonce in CrSD. If a router caches some content for which `ACCT` flag is `AGGREGATE`, CrSD can be simply dropped when *pInt* messages are generated. Otherwise, if `ACCT` flag is `DISTINCT`, the nonce must be copied into the *pInt*. This is a simple modification to the router *pInt* generation procedure described in Algorithm 3 which yields insignificant overhead for consumers and routers.

This simple policy can be extended to all interests. Since consumers are not generally expected to know what type of accounting information is required, they can blindly generate a nonce for each interest they issue. Routers would correctly propagate these nonces in *pInt* messages to the producer according to the rule above. As previously noted, NDN already supports default nonce generation in interests (for the purpose of interest loop detection). CCNx, however, needs to be extended to satisfy this requirement.

5.4 Analysis and Experimental Assessment

In Section 5.1, we proposed two fundamental techniques for propagating accounting information to producers: encryption-based and *pInt*-based approaches. The former technique is attractive because it is entirely transparent to routers. Conversely, accounting based on *pInt* messages requires routers to generate these messages and also forward them towards producers using the same data plane logic as normal interest messages.

Consider a scenario with k consumers $Cr_1, \ldots, Cr_k$ and a single producer P. Let $[Cr_i, R_1, \ldots, R_l, P]$ be the path traversed by interests issued by Cr_i for accountable content C. Let R_c, $1 \leq c \leq l$ be the router nearest to Cr_i where C is cached. Let p_l be the number of messages traversing $R_1 - R_c$ path in one direction, and let p_r be the number of messages traversing the $R_c - P$ path in one direction. Finally, let γ be the number of interests issued by all Cr_i, $i = 1, \ldots, k$, along the $R_1 - P$ path. Recall that encryption-based accounting requires consumers to issue at least two interests: one for the content itself, and (at least) one for decryption keys. The former traverses the $R_1 - R_c$ path and the latter traverses the $R_1 - P$ path. Thus, $p_l = 4\gamma$ and $p_r = 2\gamma$. In the *pInt*-based approach, a single interest is issued for C on $Cr - R_c$ path, then a *pInt* is generated at R_c and forwarded along $R_c - P$ path. In this case, $p_l = 2\gamma$ and $p_r = \gamma$. Note that $R_c = P$ is identical to the scenario where there are no router caches and thus no *pInt* messages. This case performs worse than the *pInt*-based variant since $p_l = 2\gamma$ and $p_r = 2\gamma$; a single RTT from the Cr_i to P for C.

The differences in p_l occur because, unlike interests, *pInt* messages elicit no response from the producer. In fact, *network overhead*, in terms of the number of messages, of the encryption-based accounting approach is at least twice that of the *pInt*-based approach. Meanwhile, network overhead of the cacheless variant (which obviates the need for *pInt* messages and accounting information) is more than in the *pInt*-based approach as well. Furthermore, producers and consumers incur additional overhead due to encryption and decryption op-

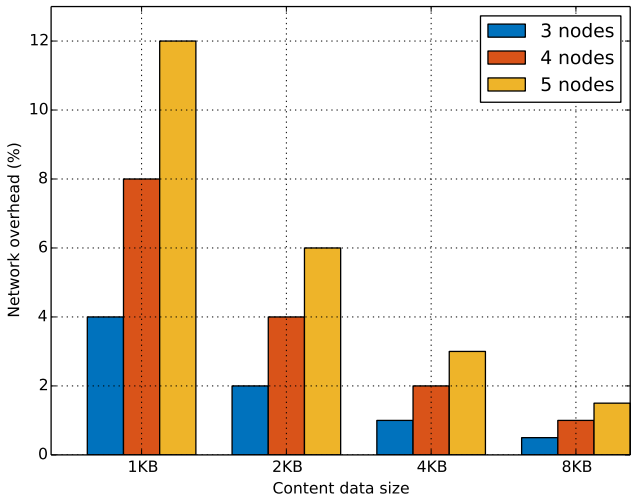


Figure 5.2: Network overhead imposed by forwarding *pInt* messages

erations. Therefore, in the following experimental assessment, we focus on the *pInt*-based accounting approach, since it is (a) more efficient and (b) proportional to the overhead incurred by network entities in the encryption-based scenario. We assume that the *pInt* generation procedure can be performed in constant time by routers. We also assume that any symmetric key management protocols are done offline and are therefore not part of real-time communication.

If interests are satisfied from R_c's cache, all upstream routers on the consumer-to-producer path incur the overhead of forwarding *pInt* messages to the producer. Figure 5.2 shows this overhead as a function of corresponding content size and the number of links between the router generating the *pInt* message and the producer. This overhead is computed as the ratio of extra bytes (due to forwarded *pInt* messages) traversing each link over the size of the corresponding content object. The x-axis represents content data size, without including the header. Also, we calculate the overhead in three line topologies with 3, 4, and 5 nodes, thus, forming 2, 3, and 4 links, respectively. The first node is the consumer Cr and the last node is P. For the purpose of this exercise, we assume the following:

- Any content requested by Cr can be satisfied by the first hop (consumer facing) router's (R) cache, i.e., cache hit rate at R is 100%. This accounts for the highest network overhead since *pInt* must traverse all links (except one) connecting Cr with R.
- Router FIBs is pre-configured to forward all interests and *pInt* messages towards P.
- Interest, *pInt*, and content object headers only contain a name of length 40 bytes.[7]

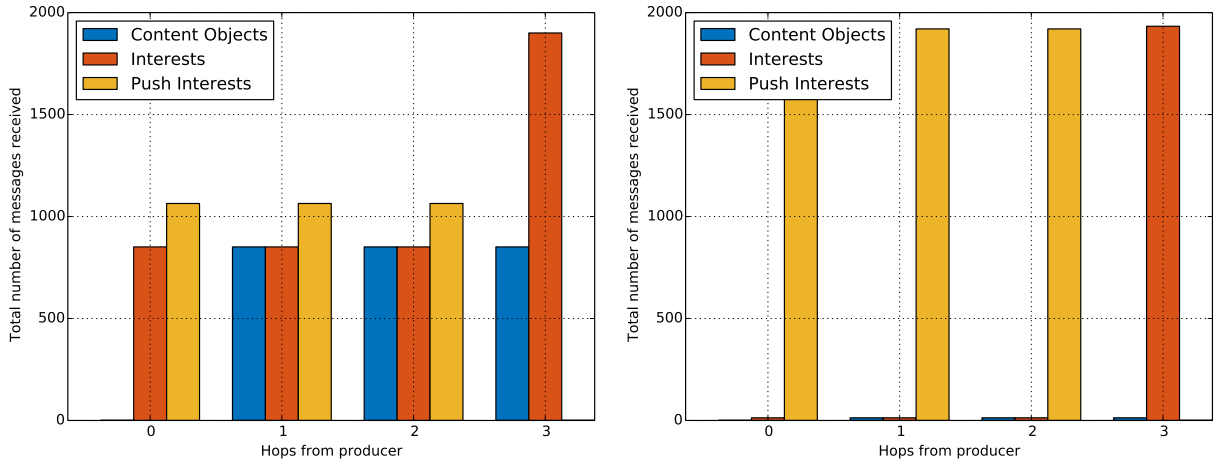
Results show that as the size of content objects grows, the bandwidth overhead of *pInt* messages decreases. This overhead would increase in complex topologies, e.g., k-ary trees rooted at P. However, network overhead would incur a similar decline as the content object size increases.

5.4.1 Message Count Overhead

To further understand the performance impact of *pInt*-based accounting in different topologies, we study the overhead incurred by each entity – consumer, router, and producer – as a function of the distance from the producer. To do this, we implement a custom discrete-time event-driven simulator that models a variety of path and binary tree CCN topologies. Consumers are configured to issue interests for a single producer at a Poisson rate with mean A. Names of each interest are uniformly sampled from a pool of M names. Each router invokes the *pInt*-Generation procedure (shown in Algorithm 3) upon every cache hit. We place no restriction on cache sizes since the set of possible content objects is small enough to fit within any reasonably-sized cache. In addition, producers respond to interests with content objects carrying fixed payload of 1MB.

We argue that the number of messages is indicative of bandwidth overhead incurred by *pInt* messages. This is because the size of *pInt* messages is proportional to the size of interests,

[7]The average URL length in IRCache HTTP traces [8] is around 40 characters.



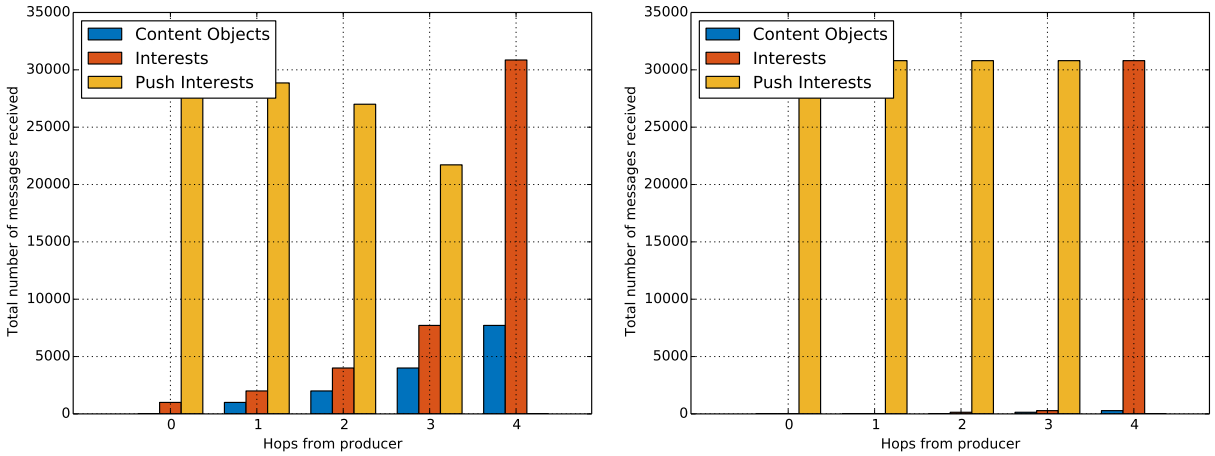
(a) Path topology with 5 nodes (1 consumer, 3 routers, and 1 producer), $A = 500$ and $M = 1000$

(b) Path topology with 5 nodes (1 consumer, 3 routers, and 1 producer), $A = 500$ and $M = 10$

Figure 5.3: *pInt*-based accounting overhead in networks with path topologies

even using *secure* consumer-specific data. Therefore, it is not the size of these messages that is important, it is their quantity.

We study this overhead in networks with line and tree topologies. For simplicity, we restrict our analysis to 5-hop lines and binary trees of height 5. By varying A and M, we show how many messages of each type are processed by each entity as a function of the distance from the producer. Figures 5.3 and 5.4 illustrate the results. The y-axis represents the total number of received messages (content objects, interests, and *pInt* messages) as a function of the number of hops from the producer. The node with hop distance 0 (the producer itself) does not receive any content. Also, the node with hop distance 3 in line topology and 4 in tree topology (the consumer facing router) does not receive any *pInt*. Also, as M decreases, the likelihood of cache hits increases. This results in a clear increase in *pInt* processing at each entity upstream from the cache hit location. For instance, when $M = 10$, approximately 99% of all messages processed by routers upstream of cache locations are *pInt* messages, in both line and tree topologies.



(a) Binary tree of height 5 (32 consumers, 30 routers, and 1 producer), $A = 500$ and $M = 1000$

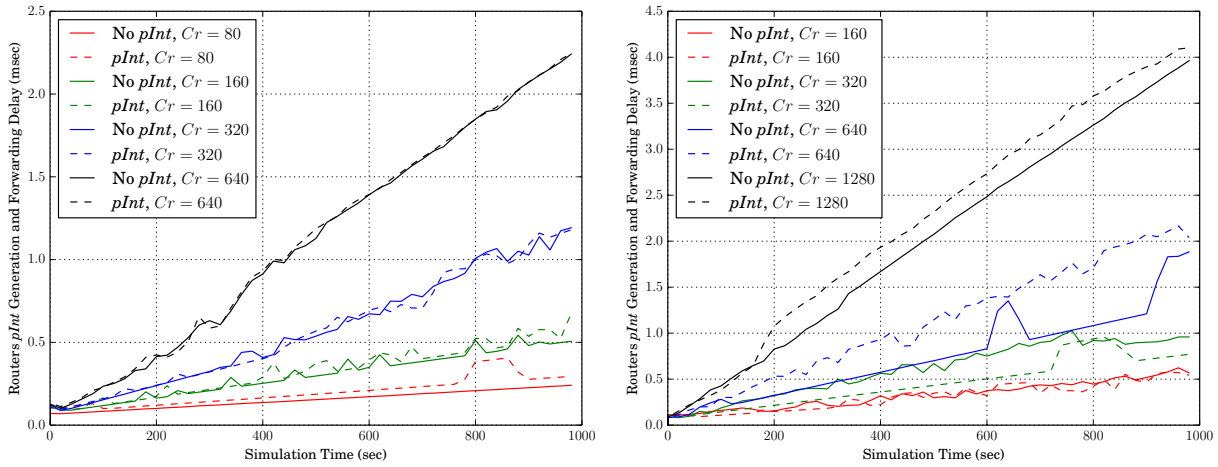
(b) Binary tree of height 5 (32 consumers, 30 routers, and 1 producer), $A = 500$ and $M = 10$

Figure 5.4: *pInt*-based accounting overhead in networks with tree topologies

The interest request rate is highly dependent on the type of application. High request rates for popular content, which is likely to be cached, will lead to a proportionally high number of *pInt* messages propagating towards the producer. If interests are issued for unpopular or uncached content, then approximately the same number of interests will be propagated upstream. In other words, from the producer's perspective, the sum of interests and *pInt* messages will equal the total number of content requests from all consumers, i.e. the producer overhead is linear in the number of content requests. The difference in these two cases is that *pInt* messages are typically smaller in size than interests.

5.4.2 Router Overhead

To measure router overhead due to generating and forwarding *pInt* messages we extended ndnSIM 2.0 [127] – an implementation of NDN architecture as an NS-3 [15] module for simulation purposes – to support *pInt* messages. Using this modified architecture we ran two sets of experiments using the DFN and AT&T topologies presented in Chapter 4.



(a) DFN topology (b) AT&T topology

Figure 5.5: *pInt* messages generation overhead at routers

In all experiments, consumers issue interests at a rate of 10/sec for the same content with the name `/prefix/A/00`. To capture the worst-case scenario, with the maximum number of *pInt* messages, we disable interest collapsing and set the `ExpiryTime` to the simulation time to ensure that this content is cached at routers for the duration of the simulation. This forces routers to generate one *pInt* for every cache hit, resulting in the maximum number of *pInt* messages.

We measure router overhead compared to the baseline case where *pInt* messages are not generated. Figure 5.5(a) shows this overhead in the DFN topology parameterized by the number of consumers connected to edge routers: 80, 160, 320, and 640. Even with 640 consumers, the overhead for an average router is negligible. Figure 5.5(b) shows the same overhead in the AT&T topology. In case of $1,280$ consumers, routers incur 15% overhead, which we consider to be tolerably low.

Chapter 6

Secure Fragmentation

One issue that straddles networking (specifically, packet forwarding) and security is fragmentation of large packets. Originally present in IPv4, intermediate fragmentation was deprecated in IPv6 for a number of reasons, many of which were identified in [98], e.g., router overhead and code complexity. Also, there have been attacks that took advantage of IPv4 reassembly [208]. Thus, eschewing fragmentation made sense for IPv6. However, the same might not hold for all network architectures. We show in Section 6.2 that, for some different types of networks, such as CCN, fragmentation is sometimes unavoidable and might even be beneficial. Furthermore, in-network caching complicates fragmentation and reassembly of content objects. Routers should ensure the authenticity and integrity of all passing fragments before caching them, separately or as a whole. This is not an issue in IPv4-based networks since fragments reassembly is deferred to end-hosts.

This chapter constructs a secure fragmentation scheme which addresses several important security and efficiency issues. Its contribution is two-fold:

First, we discuss, in detail, numerous issues related to fragmentation of both interest messages and content objects in CCN and arrive at the following conclusions:

- Interest fragmentation is unavoidable: if encountered, hop-by-hop reassembly is required.
- Content fragmentation is similarly unavoidable.
- Path Maximum Transmission Unit (MTU) discovery helps, but does not obviate the need for fragmentation.
- Intermediate reassembly is viable but buffering can be costly and latency is problematic.
- Intermediate re-fragmentation is also unavoidable for content fetched from router caches.
- Reconciling cut-through forwarding of fragments (no intermediate reassembly) with content authentication is possible in an efficient manner.

Second, we construct a secure fragmentation and reassembly method for CCN, called Fragmentation with Integrity Guarantees and Optional Authentication (FIGOA). It supports fragmentation of content packets at the network layer and cut-through switching in routers by avoiding hop-by-hop fragmentation and reassembly, thus lowering end-to-end delay.

FIGOA is fundamentally compatible with the CCN's tenet of not securing the channel but rather the content flowing through it. As its basis, FIGOA employs a delayed authentication method similar to [192], which allows routers to efficiently verify signed content based on (out-of-order) arriving fragments. In the event that a given content ultimately fails either integrity or authenticity checks, reassembly and eventual delivery of corrupt content to consumers is prevented.

However, even though cut-through switching reduces latency, it allows fragments to be temporarily stored in routers awaiting verification by FIGOA. This might result in exhausting

router resources if fragment-drop rate increases. Therefore, routers adaptively set timeouts for temporarily stored fragments. This issue is not discussed further, as it is outside the scope of this chapter.

6.1 Fragmentation Synopsis

We define *fragmentation* as a means of splitting large packets into smaller ones, at the network layer, independent of the producer and without changing any actual *content*. This is in contrast with *segmentation*, where a producer splits a large content object into smaller ones, signing and naming each separately. TCP/IP has an analogous dichotomy: TCP *segments* a byte stream into IP packets, whereas, IPv4 *fragments* IP packets into smaller packets that fit into a link MTU.

Since the late 1980s, network-layer fragmentation has been widely considered to be a headache and something to be avoided, based primarily on the IPv4 experience [98]. We briefly discuss pertinent IPv4 fragmentation concepts below.

Packet fragmentation is not a singular concept; it can be divided into two types: source-based and network-based. Source-based fragmentation is performed exclusively by the sender and is relatively simple. Assuming knowledge of the MTU for a given path to the destination, the source can fragment a packet with almost no fear that further fragmentation might be encountered along the path.[1] Knowledge of the MTU does not come for free; an MTU discovery protocol is needed, e.g., [128]. Also, the entire premise of source-based fragmentation is questionable: Why should the source fragment a large IP packet instead of simply "segmenting" it into a sequence of separate IP packets [109]? Source-based segmentation often allows for more efficient use of smaller datagrams; for example, segmented TCP datagrams can be

[1] Dynamic routing in IP may cause successive packets to take different paths, affecting the source's perceived MTU.

individually acknowledged, whereas a larger TCP segment split using IP fragmentation can only be processed as a whole by TCP.

Network-based fragmentation requires routers to support extra functionality (i.e., additional code) which entails appreciable processing overhead [98]. Having to fragment a packet takes a router off its critical path and can thus cause congestion; this can also be exploited as a DoS attack. Nevertheless, at a conceptual level, it can be claimed that intermediate fragmentation offers better bandwidth utilization than its source based counterpart, or no fragmentation at all.

Further issues are prompted by reassembly of fragments. In IPv4, reassembly takes place only at the destination. Each IP packet is allocated a buffer that stores fragments that have arrived thus far (possibly out-of-order). Once all fragments are received, the packet is physically reassembled and passed on to the upper layer. This seemingly simple procedure has been a source of many attacks and exploits [208]. Reassembly by intermediate hops/routers is not viable in IP since fragments of the same packet are not guaranteed to follow the same path.

6.2 Fragmentation in CCN

Both CCNx and NDN do not provide explicit support for cut-through fragmentation. In fact, NDN only provides hop-by-hop fragmentation with reassembly [24]. However, the current NDN implementation, which runs as part of the NDN testbed [14], is implemented as an overlay on top of TCP or UDP. In this setup, fragmentation is handled by either (1) transport layer protocols, e.g., TCP segmentation, or by (2) network layer protocols, e.g., IP fragmentation. Moreover, if NDN is running directly over the link layer, protocols such as

Table 6.1: Fragmentation terminology

Term	Description
MTU	Largest unit (packet) size for network-layer transmission over a given link between two adjacent nodes.
Content fragment (CF)	A unit of CCN network layer transmission; content fragment is the same as content object if the latter fits within the MTU of a link between two adjacent routers.
Segmentation	A process of partitioning large content into separate content objects by explicitly naming and signing each one. Can be performed only by a producer of content.
Fragmentation	A process of splitting an (already signed and named) content object into multiple content fragments. It can be performed by a producer, a router or any other CCN entity that produces, stores or caches content.
Re-fragmentation	A process of splitting a fragment of a content object into multiple fragments. Sometimes it is referred to as internetwork fragmentation [98]. Re-fragmentation can be performed by a router.
Reassembly	A process of re-composing a content object from its fragments. It can be performed by a consumer or a router (in case of intermediate reassembly).
Fragment Buffering	A process of maintaining a stash of fragments until complete packet reassembly becomes possible.
Cut-Through Switching	A process of forwarding of individual content fragments without reassembly.

NDNLP [180] can be used to handle fragmentation. Regardless of all their claimed benefits, the main drawback of these methods is that they all require reassembly at every hop.

The rest of this section discusses certain factors motivating fragmentation in CCN. Using the terminology presented in Table 6.1, fragmentation is considered in the context of interest messages and content objects, respectively.

6.2.1 Fragmentation of Interests

As discussed above, an interest message carries the name of content requested by the consumer. CCN does not mandate any confidentiality, integrity or authenticity for interest

packets. Due to no limitations on the length of content names, it is quite possible that an interest packet carrying a very long name might not fit into a network-layer MTU, thus prompting the need for source-based and/or intermediate fragmentation. Fortunately, this does not pose any real challenges, since the "design space" of interest fragmentation is very confined. Specifically, we claim the following:

If interest packets are fragmented and, possibly re-fragmented, hop-by-hop (intermediate) reassembly of fragmented interest packets is unavoidable.

The intuition behind this claim is obvious: each router that receives an interest must search its PIT, cache, and/or FIB using the name carried in said interest. If the name itself spans multiple fragments which are processed independently without reassembly, such lookups are infeasible.

Furthermore, since the consumer issuing an interest has no *a priori* knowledge of the smallest MTU on the path to the closest copy of requested content, it can not pre-fragment an interest in order to avoid further fragmentation by intermediate routers, unless there is a well-defined and globally-accepted minimum MTU for CCN interests.

For the remainder of this chapter, we assume intermediate fragmentation coupled with intermediate reassembly for interests. The remaining discussion of fragmentation is limited to CCN content objects.

6.2.2 Fragmentation of Content

Recall that CCN mandates each content to be signed by its producer. This means that, in principle, any CCN entity, whether router or consumer can check content integrity and

authenticity using the producer's public key.[2] The public key can be either referred to by name in the content header, or embedded within the content.

Consequently, in order to abide by CCN tenets, fragmentation must not preclude routers from verifying signatures, i.e., checking authenticity and integrity of content. This speaks in favor of either: (1) no intermediate fragmentation at all, or (2) hop-by-hop (intermediate) reassembly.

Producer-based Fragmentation or Segmentation

At a first glance, there seems to be no reason for a content producer to fragment large content. Instead, it can simply *segment* it into individually named and separately signed content objects. This segmentation approach is sensible for content meant to be pushed (e.g., email) or generated dynamically, e.g., in response to a database query. The segment size can be determined from an MTU discovery protocol (as described below). This would ensure no intermediate fragmentation.

However, for content that is meant to be pulled (distributed), a producer may benefit from signing and naming it once and not worrying about repeating a (possibly expensive) segmentation procedure each time it receives an interest for the same content. In this case, when an interest arrives, the producer may choose to fragment a previously produced content object. This entails no real-time cryptographic overhead. Alternatively, a producer could choose to segment content using the smallest MTU of all of its interfaces, thus incurring even less processing at interest arrival time.

An important issue is content header overhead incurred when generating small-size segments. Segmenting a large object into many MTU-sized segments requires each to have its own header, dominated by the `Signature` and related fields.

[2]Content signature verification is mandatory for consumers and optional for routers as discussed in Chapter 4.

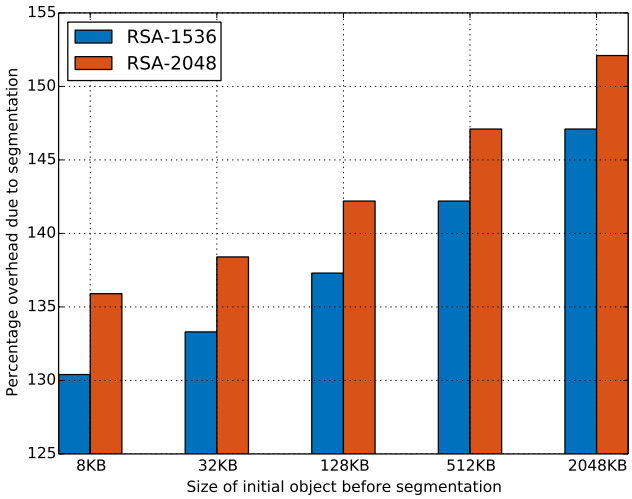


Figure 6.1: Byte count overhead for small signed segments

Without getting into details of CCN signature format, Figure 6.1 shows the overhead of segmenting larger objects down to MTU size. We use a standard 1,500-byte link MTU and SHA-256 as the hash algorithm. We considered both RSA-1536 and RSA-2048 signatures. The `Signature` field contains 12 bytes of fixed overhead (headers) and the actual signature bits (192 bytes for RSA-1536, 256 bytes for RSA-2048). However, estimating the exact size of the signature-related fields is more complex. This is because the `KeyName` field, can be of an arbitrary size. Moreover, if the content carries the actual public key or a certificate (using the `PublicKey` and `Certificate` fields respectively), its size can be *very* large. For now, we assume a small 20-byte `KeyName`. Figure 6.1 shows that there is a definite penalty for segmenting at the producer. Even in the most favorable case (8KB data objects and RSA-1536), over 30% of the bits are wasted on redundant information. As we move to larger objects, this overhead can grow to 50%.

Whither Intermediate Fragmentation

Regardless of whether a producer segments or fragments content, intermediate fragmentation can not be avoided or ruled out, since CCN does not mandate a global minimum MTU. Even if it existed, segmenting content to adhere to this MTU might be very wasteful. This is due to poor bandwidth utilization on links that have higher MTUs and increased overhead due to

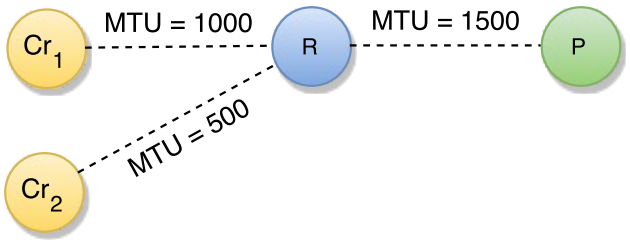


Figure 6.2: Intermediate re-fragmentation simple topology

the costs of signature generation by producers and verification by consumers and, optionally, by routers.

Another possibility is to introduce an MTU discovery method, whereby, an interest traveling towards requested content would carry a new field reflecting the smallest MTU (μMTU) discovered thus far on its path.[3] This is a viable and lightweight approach, particularly because a content must traverse, in reverse, the very same path taken by an interest for that content. Hence, when the first entity that stores, caches, or produces content receives an interest, it can use μMTU to fragment the replied content (or segment it, if this entity is the producer). Note that "entity" could encompass: (1) an application-level repository that only stores content, (2) a router that caches content, or (3) a producer that generates its own content. This way, fragmentation would occur only once per interest.

Unfortunately, fragmentation via interest-based μMTU discovery does not eliminate the need for re-fragmentation. Consider the topology in Figure 6.2, with two consumers, one router, one producer publishing content C, and the MTU values shown.

1. Consumer Cr_1 sends interest Int_1 for C to router R.

2. R receives and marks Int_1 with $\mu\text{MTU} = \text{MTU}_{(R \rightarrow Cr_1)}$ (MTU corresponding to the $R - Cr_1$ link). It then creates a PIT entry for Int_1.

3. R forwards Int_1 to the corresponding producer P.

[3]This is actually the MTU of the *opposite* link direction from the direction the interest traveled since links may have asymmetric MTUs.

4. Since $\text{MTU}_{(P \rightarrow R)} > \mu\text{MTU}_{(R \rightarrow Cr_1)}$, P does not change μMTU in Int_1.

5. P immediately satisfies Int_1, fragmenting C according to μMTU.

6. Meanwhile, between Step 3 and now, consumer Cr_2 issues interest Int_2 and forwards it to R.

7. R receives Int_2 and marks it with $\text{MTU}_{(R \rightarrow Cr_2)}$ where $\text{MTU}_{(R \rightarrow Cr_2)} < \mu\text{MTU}$. Then, R collapses Int_2 into the existing PIT entry for Int_1. At this time R is buffering fragments which have arrived from P (not all fragments of C might have arrived).

8. R partially satisfies Int_2 using fragments available in the buffer, and previously forwarded to Cr_1. These fragments have to be *re-fragmented* with $\text{MTU}_{(R \rightarrow Cr_2)}$. Any further fragments which arrive from P should also be re-fragmented by R using $\text{MTU}_{(R \rightarrow Cr_2)}$ and sent to Cr_2.

Despite the fact that μMTU discovery does not eliminate re-fragmentation, it is practically free in terms of extra processing and bandwidth overhead. More importantly, it results in less re-fragmentation, since it assures that re-fragmentation occurs **at most once** for each collapsed interest at each intermediate router. This can be particularly advantageous in the case of monotonically decreasing MTUs where re-fragmentation must occur at each hop. With μMTU, this is curtailed at the source of content, which is either the producer or some intermediate router, due to pre-fragmentation.

6.2.3 Considering Intermediate Reassembly

We now discuss intermediate reassembly, motivated by at least two factors. First, we consider the case of increasing MTUs on links that compose the reverse path taken by content fragments on the way to the consumer. If MTUs increase monotonically, it might make sense to reassemble fragments (at least partially) to obtain better bandwidth utilization.

However, this benefit is arguably outweighed by reassembly costs, i.e., processing, memory, and code complexity in routers. The second factor is security. If a fragment does not carry the content producer's signature, how can a router check its authenticity? As mentioned earlier, CCN stipulates that routers, though not required to do so, must *be able* to verify content signatures.

Hop-by-hop reassembly of content fragments would clearly solve the problem and address both factors mentioned above. With it, a router would receive fragments in arbitrary order and neither cache nor forward them until all fragments arrive. It would then reassemble them and verify the signature (see Section 6.3.5 for more details).

The main problem with hop-by-hop reassembly is the significant increase in end-to-end latency, resulting in lower throughput for adaptive algorithms such as TCP. Latency accumulates at each hop, since all fragments need to be reassembled and then re-fragmented for transmission. The alternative is cut-through fragment forwarding, where each fragment is forwarded upon arrival.

If multiple flows are passing through the router, the fairest distribution of latency overhead is to interleave fragments, as in Multi-PPP Link Fragmentation and Interleaving (MLPPP LFI) [119]. This interleaving causes significant latency between consecutive fragments of an object, which grows with the number of simultaneous flows. Latency accumulates at each hop, since all fragments need to be reassembled and then re-fragmented for transmission.

We attempt to evaluate the benefits of cut-through fragment forwarding by considering a simple topology with a linear 8-hop path with 100 Mb/s links. Each link accumulates 10ms of latency, ignoring intra-hop and queuing delays. We assume $8,400$-byte content objects split into 7 fragments of $1,300$ bytes each.

Table 6.2 shows the slowdown caused by intermediate reassembly as each node waits for all fragments of an object, for varying numbers of parallel flows (which controls the amount of

Table 6.2: Latency due to per-hop content reassembly

	Number of flows					
	5	10	20	30	50	100
Inter-fragment gap (ms)	0.52	1.04	2.08	3.12	5.20	10.4
First-to-last fragment gap (ms)	3.22	6.34	12.58	18.82	31.30	62.50
E2E latency: reassembly (ms)	105.79	130.75	180.67	230.59	330.43	580.03
E2E latency: cut-through (ms)	83.22	86.34	92.58	98.82	111.30	142.50
Reassembly slowdown %-age	127.12	151.43	195.14	233.34	296.87	407.03

interleaving). The *inter-fragment gap* is the time elapsed between consecutive fragments of an object, caused by fragments of other objects being interleaved. The *first-to-last fragment gap* is the time elapsed between the arrival of the beginning of the first fragment and the end of the last fragment. *E2E latency: reassembly* is the total latency for each content object, with intermediate reassembly. *E2E latency: cut-through* is the total latency in case of content fragmented at the first hop and all fragments cut-through forwarded with no re-fragmentation or reassembly in route. Finally, *Reassembly Slowdown* shows the extra cost of reassembling and re-fragmenting at every hop, as compared with cut-through forwarding.

Figure 6.3 shows the evolution of increased latency for various object sizes and fragment counts. Using the aforementioned 8-hop topology, we vary the number of flows on each link. Two links (close to the ends) have 10 flows across them, two links have 20 flows, two links have 50 flows, and the two core links have 100 flows. The graph shows that even a small number of fragments can significantly increase latency over commonly seen path lengths and flow counts. It takes only 6 fragments per object to **double** end-to-end latency of hop-by-hop reassembly when compared to cut-through forwarding of fragments. This clearly shows that any fragmentation scheme that requires hop-by-hop reassembly of every content object (as is the case today with CCNx [142] and NDNLP [180]) incurs severe penalties. We believe that fragments must be forwarded in a cut-through fashion; consequently, our scheme implements this feature.

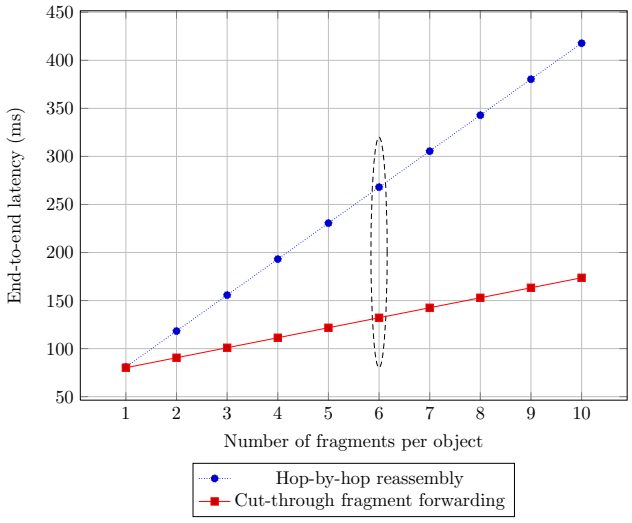


Figure 6.3: Latency with varying fragment counts per object

At this point, it is worth asking: should routers perform reassembly and verify signatures? We believe that it does not make much sense for backbone routers to do so since potential attacks are not likely to originate in the backbone, but rather at the edges of the Internet. As described in Chapter 4, signature verification at stub AS (ingress) routers is more appropriate, e.g., because of policy dictating that no fraudulent content must reach consumers.[4] Also, stub AS egress routers might reassemble fragments and verify signatures if there is a policy disallowing any fraudulent content to exit the AS, e.g., for reasons of liability.

The above discussion yields a trivial observation that reassembly implies the ability to verify signatures. However, it is unclear whether signature verification implies the need for reassembly. This triggers the following challenge which we attempt to address in the remainder of this chapter:

If content objects are fragmented (and, possibly re-fragmented), and intermediate reassembly is not viable, can routers still check content authenticity?

In other words, if verifying integrity/authenticity is the main reason for intermediate reassembly, is there a way to obtain the former while avoiding the latter?

[4]Even though CCN stipulates consumer-based signature verification.

6.2.4 Fragment Delivery Order

One important issue relevant to intermediate reassembly is whether fragments are always delivered in order of transmission between any two adjacent CCN routers.

Clearly, reassembly is easier if ordered delivery can be guaranteed. In a hypothetical network setting where CCN is universally deployed directly on top of the MAC layer, ordered delivery of content fragments might be a reasonable assumption.[5] However, certain connectivity and communication choices make ordered delivery hard to guarantee. For instance, if adjacent routers support multiple/parallel physical links with variable data transfer rates, it is possible that an earlier-transmitted fragment is received after a later-transmitted one. Also, an error on one of the links might cause the same situation even if link speeds are comparable. Even without multiple links, if pipelined data-link layer transmission is used, especially over a wireless channel, one fragment could be corrupted and discarded and the next one could be received intact, resulting in the latter being received first.

In case where CCN is deployed as an overlay on top of IP, and not directly on top of the MAC layer, out-of-order fragments delivery is also possible. This is because two adjacent CCN routers might have multiple IP routers between them. Since IP does not guarantee ordered delivery of packets, it is possible for two adjacent CCN routers to receive fragments out of order.

6.2.5 Incremental or Deferred Fragment Caching?

Recall that router-based content caching is not, strictly speaking, a hard requirement. However, it is expected that each CCN router will maintain a cache of a certain size.

[5] All content fragments traverse, in reverse, the very same path taken by an interest.

A router that employs intermediate reassembly can defer the decision to cache content until it receives all fragments and, optionally, verifies overall content integrity and/or authenticity. Whereas a router that employs *cut-through switching* of individual fragments has a choice to either: (1) cache fragments incrementally as they arrive, or (2) defer caching (i.e., buffer fragments) until all fragments arrive and their overall content integrity and/or authenticity is verified. Assuming that most content is authentic, the former optimizes the common case of quickly caching the last fragment once optional security checks are performed. On the other hand, incremental caching may complicate matters, since it might, depending on the specific cache architecture, involve non-contiguous caching of related fragments.

If deferred caching is used, another fragmentation- and caching-related issue arises: how to store fragments. One possibility is to store them in the same form they arrive. This might work if no re-fragmentation is performed locally. Otherwise, it might make sense to store fragments in the form they are forwarded. This becomes more complicated in case of collapsed interests, i.e., when content needs to be forwarded out on multiple interfaces with different MTUs. Another approach would be to proactively re-fragment cached fragments for all possible link MTUs on the router. Pre-fragmentation would reduced delay at the cost of additional cache storage space. We believe this technical issue deserves further consideration, which is beyond this scope of this chapter.

6.3 Secure Fragmentation

This section describes a scheme called FIGOA: Fragmentation with Integrity Guarantees and Optional Authentication. It supports arbitrary intermediate fragmentation [98] of content while preserving security and not requiring intermediate reassembly before forwarding all fragments.[6] FIGOA does not rely on in-order arrival of fragments, nor does it mandate

[6] A variant of FIGOA can be used in conjunction with intermediate reassembly, with the key advantage of faster cryptographic processing.

any type of fragment caching strategy. It is primarily geared for routers that support cut-through switching and maintain dedicated storage for buffering content fragments, distinct from the cache. While cut-through fragment switching is generally beneficial, it complicates signature verification, as discussed in Section 6.2.3. We address this problem by using delayed authentication (DA). In addition, FIGOA allows free mixing of routers that do not perform intermediate reassembly with those that do.

6.3.1 Delayed Authentication

Delayed authentication was first introduced in [192]. Its goal was to *"reconcile fragmentation and dynamic routing with network-level authentication in IP gateways."* The essence of delayed authentication is that a given packet's authenticity can be obtained from the authenticity of its fragments. Packet authentication is computed incrementally as individual fragments are received (possibly out of order), processed and forwarded by a router. This requires a queue for each partially received packet that maintains the current state of partial verification. For every fragment, incremental verification is performed, queue state is updated, and the fragment is forwarded. Upon receipt of the final fragment (called a "hostage"), the router completes verification. If it succeeds, this fragment is forwarded. Otherwise, it is discarded along with the entire queue. The end-result is that the destination receives the packet in its entirety only if it is verified by the router.[7]

The main differences between delayed authentication in its original IPv4 context [192] and the proposed use in CCN are as follows:

- **Symmetric routing**: unlike IP, where fragments of the same IP packet might travel via different paths, fragments of the same CCN content are guaranteed to traverse the

[7]Recall that, in IPv4, the destination must flush all fragments of a packet that it can not reassemble, either due to a timeout or other errors.

same set of CCN routers. This results in much higher probability of ordered fragment delivery and faster timeouts in cases of lost or corrupted fragments. Note that it is the responsibility of consumers to re-request the entire content in case of lost or corrupted fragments.

- **Not just ingress routers**: delayed authentication was initially designed for ingress routers (i.e., border routers of the destination AS). In CCN, any intermediate router can unilaterally choose to perform delayed authentication.

- **Possible intermediate reassembly**: in IP, only the destination reassembles fragments, whereas, any intervening router can decide to reassemble whether or not it decides to do cut-through forwarding.

- **Signatures instead of MACs**: delayed authentication was initially proposed for authenticating IP packet traffic flowing between two hosts (in two stub AS-s) that share a symmetric key. Message Authentication Codes (MACs) were used for incremental fragment authentication. In CCN, signatures are used to ascertain content authenticity.

6.3.2 Hash Functions

The last item above – the use of signatures – is what most distinguishes delayed authentication in CCN from its IP counterpart. CCN routers do not use symmetric cryptography for packet authentication. Even if they did, assuming a key shared among (possibly all) routers that forward a given content is unrealistic. The only realistic means of authenticating content in routers is by verifying signatures. This prompts the question: how does one reconcile delayed authentication (of fragments) with signatures?

We approach this issue by observing that a signature is computed over a fixed-size hash (digest) of content, i.e., using the so-called "hash-and-sign" paradigm. A hash provides

integrity while a signature of a hash provides authenticity or origin authentication. The underlying cryptographic hash function $H(\cdot)$ must satisfy a set of standard properties [133]. Unlike a MAC or a HMAC, a hash function requires no secret key and can be computed by anyone.

Most modern hash functions operate on input of practically any size.[8] They typically use an iterative model, also known as the Merkle-Damgård construction, whereby input is broken into a number of fixed-size blocks and processed one block at a time by an internal compression function $h(\cdot)$. The latter forms the core of the hash function. After processing each block, $h(\cdot)$ produces an intermediate value – internal state that we call IS – that is usually of the same size as the final hash output. In case of the first block, the intermediate state is fixed and referred as the Initialization Vector (IV). The last block is typically padded with zeros followed by the total input size in bits. For example, the well-known SHA-256 hash algorithm [19] operates on 512-bit blocks, maintains 256-bit internal state and yields a 256-bit hash.

In constructing FIGOA, we take advantage of internal state produced by the underlying compression function $h(\cdot)$. The main idea is to include, in each fragment, the internal state of the hash function **up to**, but not including, that fragment. This allows incremental hashing of each fragment without having received either preceding or subsequent fragment(s).

We assume that the absolute minimum MTU of any link or interface that takes advantage of FIGOA is equal to at least one block of data: one block of internal state and whatever size is needed to accommodate a content fragment header (i.e., content name, flags, etc.). More precisely, we assume that any fragment must carry at least a header, internal state and some blocks of data. All data must be aligned on block boundaries.

[8] We consider 2^{64} or 2^{128} bits as "practically any".

Table 6.3: Fragmentation notation

Term	Description
β	block size of $h(\cdot)$
C^n	Raw (unsigned) content of total size n bits.
$\mathsf{Sig}(C^n)$	Producer's signature on C^n.
$\overline{C}^N$	Signed version of C^n of size $N = n + \lvert\mathsf{Sig}(C^n)\rvert$ bits.
$b_{v,s}$	Contiguous component of $\overline{C}^N$ where $0 \leq v < N$, i.e., $b_{v,s}$ represents s bits, starting with offset v and ending with offset $v + s - 1$, inclusive. s and v are multiples of β.
$CF^N_{v,s}$	Fragment of $\overline{C}^N$ that carries $b_{v,s}$.
IS_v	Internal state of $h(\cdot)$ after processing v bits of input. v is a multiple of β
oMTU	MTU of router's outgoing interface.
aoMTU	oMTU adjusted for each fragment header size i.e., $aoMTU = oMTU - \lvert$fragment header$\rvert$
$\mathbb{F}$	Set of content fragments.
$\mathbb{B}$	Temporary buffer storing all fragments received so far.

6.3.3 FIGOA Description

From here on, we use additional notation presented in Table 6.3. The proposed scheme includes three main tasks, described separately below.

Content Fragmentation

This task, shown in Algorithm 4, is triggered whenever a CCN node (router or producer) needs to forward a content object larger than oMTU. Each resulting fragment $CF^N_{v,s}$ includes: (1) s bits of original content – $b_{v,s}$, (2) starting offset v, and (3) IS_v – intermediate state, i.e., output of $h(\cdot)$ on inputs of IV and $b_{0,v-1}$ ($IS_v = h(IV, b_{0,v-1})$).[9] To simplify presentation, Algorithm 4 makes two assumptions: (1) aoMTU is a multiple of β, i.e., $aoMTU = s * \beta$ and (2) N (signed content size) is a multiple of aoMTU, i.e., $N = k * aoMTU$, which makes all fragments of equal size.

Re-fragmentation

This task, illustrated in Algorithm 5 is very similar to the initial fragmentation task, except

[9] In the very first fragment, $v = 0$ and $IS_v = IV$.

Algorithm 4 Fragment-Content

Input: $\overline{C}^N = b_{0,N-1}$, aoMTU, IV, $h(\cdot)$
Output: $\mathbb{F}$
$\mathbb{F} := \emptyset,\ v = 0,\ IS_v = IV$
$s = \frac{aoMTU}{\beta},\ k = \frac{N}{s}$
for $i = 0,\ i < k,\ i++$ **do**
    $CF^N_{v,s} := \langle v, b_{v,s}, IS_v \rangle$
    $\mathbb{F} := \mathbb{F} \cup CF^N_{v,s}$
    $IS_v := h(IS_v, b_{v,s})$
    $v = v + s$
end for
Output $\mathbb{F}$

Algorithm 5 Refragment-Fragment

Input: $CF^N_{v,s} = \langle v, b_{v,s}, IS_v \rangle$, oMTU, $h(\cdot)$
Output: $\mathbb{F}$
$\mathbb{F} :=$ Fragment-Content$(b_{v,s}, oMTU, IS_v, h(\cdot))$
Output $\mathbb{F}$

that it is performed only by CCN routers, and on content fragments, instead of content objects.

Content Verification

As mentioned earlier, FIGOA provides integrity/authenticity for fragments received in any order. Recall that a router or a consumer can unilaterally decide whether to either: (1) incrementally verify integrity of each fragment as it is received or (2) defer overall verification until all fragments are received. Regardless of the choice, a router should forward each fragment in a cut-through fashion, i.e., without waiting for others to arrive. Moreover, a node receiving fragments should store them in a buffer until the last fragment arrives and (final or overall) verification is performed. (See Section 6.3.5.)

When a router performing incremental fragment verification receives $CF^N_{v,s}$, one of the following cases occurs:

1. $CF^N_{v,s}$ is the very first received fragment. A new buffer $\mathbb{B}$ is created to store $CF^N_{v,s}$. $IS^*_w = h(IS_v, b_{v,s})$ is computed and stored.

2. Neither previous $CF^N_{u,s}$ (for $v = u + s$) nor next $CF^N_{w,s}$ (for $w = v + s$) fragment is in the buffer, i.e., received. $CF^N_{v,s}$ is placed in $\mathbb{B}$. $IS^*_w = h(IS_v, b_{v,s})$ is computed and stored.

3. $CF^N_{u,s}$ is in the buffer (along with IS^*_v) while $CF^N_{w,s}$ is not. IS^*_v must match IS_v in $CF^N_{v,s}$. $IS^*_w = h(IS_v, b_{v,s})$ is computed and stored.

4. $CF^N_{w,s}$ is in the buffer while $CF^N_{u,s}$ is not. IS_v should be stored and $IS^*_w = h(IS_v, b_{v,s})$ is computed and must match IS_w from $CF^N_{w,s}$.

5. Both $CF^N_{u,s}$ and $CF^N_{w,s}$ are in the buffer. IS^*_v must match IS_v in $CF^N_{v,s}$, and $IS^*_w = h(IS_v, b_{v,s})$ is computed and must match IS_w from $CF^N_{w,s}$

Once the last fragment is received, authenticity of the entire content can be finally verified. If verification fails, the last fragment is dropped, the PIT entry is flushed, and nothing is cached. The same applies for any failed check in the 5 cases above. This process is illustrated in more detail in Algorithm 6. We assume that routers perform incremental verification of fragments and verify the reassembled content signature. If signature verification is not possible, routers must verify that the reassembled content hash matches the original content hash included in every fragment (see Section 6.4 for details.)

6.3.4 Examples

We now describe FIGOA via two operational examples. **In the first example,** consider a situation where $\overline{C^N}$ has two fragments: $CF^N_{0,s}$ and $CF^N_{s,s}$, i.e. $N = 2 \times s$. A router R receives $CF^N_{0,s}$. First, R invokes $h(\cdot)$ iteratively and computes $IS^*_s = h(IV, b_{0,s})$. Then, it forwards $CF^n_{0,s}$ out on the interface(s) reflected in the corresponding PIT entry. R creates a buffer for $\overline{C}^N$ where it records the fact that it received the first s bits of content, along with the computed IS^*_s. Now, R receives $CF^N_{s,s}$. It compares the stored IS^*_s value with IS_s carried in

Algorithm 6 Verify-Fragment

Input: $CF^N_{v,s}$, associated PIT entry e, $h(\cdot)$
Output: no output
if Is-First($CF^N_{v,s}$) **then**
  $\mathbb{B}$:= Get-New-Buffer();
end if
Insert $CF^N_{v,s}$ in $\mathbb{B}$
Store $IS^*_w = h(IS_v, b_{v,s})$
if $CF^N_{u,s} \in \mathbb{B}$ **and** $IS^*_v \neq IS_v$ in $CF^N_{v,s}$ **then**
  goto *CleanUp*
end if
if $CF^N_{w,s} \in \mathbb{B}$ **and** $IS^*_w \neq IS_w$ of $CF^N_{w,s}$ **then**
  goto *CleanUp*
end if
if Is-Not-Last($CF^N_{v,s}$) **then**
  Forward $CF^N_{v,s}$ according to e
end if
if Content-Complete() **then**
  $\overline{C}^N$:= Assemble($\mathbb{B}$)
  if Verify-Signature($\overline{C}^N$) **then**
    Forward $CF^N_{v,s}$ according to e
    Cache $\overline{C}^N$
    return
  end if
end if
CleanUp: Flush $\mathbb{B}$ and e

$CF^N_{s,s}$. If they do not match, R discards the buffer and flushes the corresponding PIT entry. Otherwise, it invokes $h(\cdot)$ iteratively and computes $IS^*_N = h(IS_s, b_{s,s})$. In the end, before the content is cached and the last fragment is forwarded, R extracts $\mathsf{Sig}(C^n)$ from the received content, and computes a putative hash H' of the entire reassembled C^n. Finally, R verifies whether $\mathsf{Sig}(C^n)$ is the producer's signature on H'. If so, $CF^N_{s,s}$ is forwarded; otherwise, it is discarded along with the buffer and the PIT entry.

A similar process takes place if $CF^N_{0,s}$ and $CF^N_{s,s}$ arrive out of order. R first receives $CF^N_{s,s}$. Using IS_s carried in this fragment, R invokes $h(\cdot)$ iteratively on each block of data and terminates with IS^*_N. Next, R forwards $CF^N_{s,s}$. Then, R creates a buffer for $\overline{C}^N$ where it records the fact that it received the last $N - s$ bits (which is, in fact, the last s bits) of

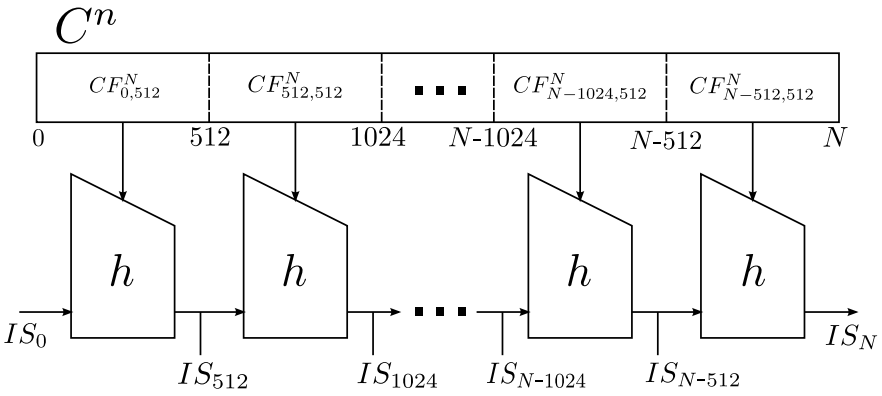


Figure 6.4: Implementing Merkle-Damgård construction to generate content fragments

content, along with IS_s and IS^*_N. Now, R receives $CF^N_{0,s}$. It invokes $h(\cdot)$ iteratively and computes $IS^*_s = h(IV, b_{0,s})$ which should match IS_s received earlier as part of $CF^N_{s,s}$. If they do not match, R discards the buffer and the PIT entry. Otherwise, R computes a putative hash H' of entire reassembled C^n, extracts $\mathsf{Sig}(C^n)$ and verifies whether it is the producer's signature on H'. If so, $CF^N_{0,s}$ is forwarded, otherwise, it is discarded along with the buffer and the PIT entry.

The second operational example involves R receiving a fragment $CF^N_{x,s}$ of content $\overline{C}^N$. The total size of this fragment is $(s + |\text{fragment header}|)$ bits. Suppose that, after processing this fragment as in the first example, R needs to forward it out on an interface with oMTU smaller than the total size of $CF^N_{x,s}$, e.g., R needs to re-fragment it into two sub-fragments. R creates $CF^N_{x,s'}$ with IS_x and $CF^N_{y,s'}$ with IS_y, where (1) $s' < s$, (2) $y = x + s'$, (3) IS_x is simply copied from $CF^N_{x,s}$, and (4) $IS_y = h(IS_x, b_{x,s'})$. This example aims to show that R can easily re-fragment already-fragmented content while preserving overall content integrity.

Figure 6.4 demonstrates how to use any hash function based on the Merkle-Damgård construction to generate content fragments. The hash function used in Figure 6.4 is SHA-256, and the length of input is discarded at the end of construction to simplify the demonstration.

6.3.5 Content Authentication

Although trust and key management are out of the scope of this chapter, we can not ignore the fact that authenticating a content object requires presence of a signature, as well as availability of a public key which must be trusted, as described in Chapter 4. CCN stipulates that public keys are encapsulated in named and signed content objects, i.e., a form of a certificate. Also, CCN allows the public key to be either: (1) referred to by name within a content object header, using the `KeyName` field, or (2) enclosed with the content object itself, using the `PublicKey` field. In the former case, unless the referred public key is already cached, the router presumably must fetch it by name, i.e., issue an interest for it. As mentioned in Chapter 4, this is a burdensome task that routers should not perform, for obvious reasons.

Fortunately, techniques proposed in Chapter 4 can be employed even if content is fragmented. For instance, if the IKB rule is universally adopted, the `KeyId` and `PublicKey` are used to verify the content signature. However, implementing FIGOA with SCNs is more challenging. This is because the hash used to form the last component of an SCN is computed over the entire content object (including its signature). In contrast, FIGOA's hash is computed over the content object without its signature field. We defer supporting SCNs in FIGOA to follow-on work.

6.3.6 Security Analysis

Security of FIGOA is based on that of delayed authentication (DA). We say that $H(\cdot)$ is constructed using the Merkle-Damgård construction with $h(\cdot)$ as the building block. If $h(\cdot)$ is collision-resistant, then so is $H(\cdot)$.

A function F has strong collision resistance if it is "computationally infeasible" to find inputs $x \neq y$ such that $F(x) = F(y)$. See [133] for information regarding Merkle-Damgård construction and hash-and-sign paradigm.

A signature computed via hash-and-sign over an unfragmented content object is considered secure. Whereas, with DA, a content object is fragmented and we arrive at the final hash by incrementally hashing its fragments.

To subvert DA we consider an adversary given a valid $\overline{C}^N$ with signature $\mathsf{Sig}(C^n)$. The goal is to send to some router R a sequence of fragments, $CF'^{N'}_{x_0=0,s}, CF'^{N'}_{x_1,s}, \ldots, CF'^{N'}_{x_k,s}$ ($x_{i+1} = x_i + s, 0 \leq i \leq k-1$) corresponding to $\overline{C'}^{N'} \neq \overline{C}^N$ with $H(C'^{x_{k+1}}) = H(C^n)$. Recall that $CF^N_{v,s}$ embodies intermediate state IS_v of the hash function computed up to, but not including, v bits of content.

First, consider fragments arriving in order. R receives $CF'^{N'}_{0,s}$, initializes $h(\cdot)$ with IV, and computes and retains $IS'^*_{x_1} = h(IV, b_{0,s})$. Now, when R receives subsequent fragments $CF'^{N'}_{x_i,s}, i = 1, \ldots, k$, it compares the current (computed) $IS'^*_{x_1}$ with IS'_{x_1} contained in $CF'^{N'}_{x_1,s}$. If they match, R invokes $h(\cdot)$ iteratively over each block of $CF'^{N'}_{x_i,s}$ using IS'_{x_i} as the starting intermediate state, and compares $IS'^*_{x_{i+1}}$ with $IS'_{x_{i+1}}$ in $CF'^{N'}_{x_{i+1},s}$. This process is exactly the same as computing $H(\cdot)$ over the entire $\overline{C'}^{N'}$. If $\overline{C'}^{N'} \neq \overline{C}^N$, the adversary must have found a collision for $H(\cdot)$, which violates our collision-resistance assumption.

Now, assume that fragments arrive out-of-order. R receives $CF'^{N'}_{x_i,s}$. It can readily compute $IS'^*_{x_{i+1}}$ by invoking $h(\cdot)$ over the blocks starting with IS'_{x_i}. R retains IS'_{x_i} as part of its state until $IS'^*_{x_i}$ is computed (using all previous fragments) and matched. If $IS'^*_{x_i}$ has been already computed, then R must have invoked $h(\cdot)$ over the data in $CF'^{N'}_{x_{i-1},s}$ using $IS'_{x_{i-1}}$. If R computes the final hash and its state contains *only* $IS'^*_{N'}$, then R has compared $IS'^*_{x_i}$ with IS'_{x_i} for $i = 1, \ldots, k$ such that each match was successful. In other words, all fragments have arrived and $i = 1, \ldots, k, IS'^*_{x_i} = IS'_{x_i}$. We observe that the set of equations that must

be *satisfied* here is exactly the same as that in the in-order-arrival case. Therefore, the same argument applies.

6.4 Implementation

Our implementation is consistent and compatible with the CCNx 0.8.2 code base, with no changes to the architecture except to support fragmentation. Due to lack of support for signature verification and key management at the network layer in CCNx 0.8.2, we do not support signature verification of content objects processed by routers. However, it can be easily extended.

Our implementation only requires modification of the CCNx forwarder code. Its design limits fragmentation, reassembly, and cut-through switching for outgoing interfaces. Moreover, a (consumer) forwarder must reassemble fragments prior to forwarding the content to the application.

To implement fragmentation, we introduce a new type of CCN packet, *content fragment*. It is used for both initial fragmentation of content and re-fragmentation of fragments. The format is:

- `Name`: identical to content name.
- `ContentObjectSize`: size of the original content object before fragmentation.
- `InternalState`: internal state of SHA-256 computation up to `PayloadOffset` of the content.
- `PayloadOffset`: where fragmented data begins with respect to the unsigned content object.

- `PayloadSize`: size of fragment payload – a multiple of 512-bits, except for the last fragment.
- `ContentObjectHash`: digest of the original content object.
- `Payload`: actual content fragment data.

As mentioned in Section 2.1, `ContentObjectHash` is an optional field in CCNx content object headers. (Recall that it is the last component of the content name in NDN.) However, we mandate this field in all FIGOA fragments. Every node (producer or router) that fragments content must compute and include `ContentObjectHash` field in all fragments. If this hash is already present in the original content object, it is simply copied into all fragments.

Once all fragments are received and content is reassembled, the router caches it if its integrity is verified.

The above format lends itself to natural re-fragmentation. If a fragment requires further fragmentation, only `InternalState`, `PayloadOffset`, `PayloadSize`, and `Payload` need to be adjusted to reflect new fragments. This prevents nested fragments and simplifies reassembly, which increases router performance and reduces end-to-end latency.

To evaluate the implementation, we compared its performance to an unmodified version of CCNx 0.8.2. This version runs as an overlay atop TCP or UDP (similar to the current NDN testbed). With TCP, content larger than the negotiated MTU (at connection setup) is segmented by TCP. This reduces the chance of IP fragmentation, unless the MTU is smaller at an intermediate router. In case of UDP, IP is responsible for content fragmentation and reassembly. In this case, every CCNx node receives the whole content from a UDP socket after reassembly is performed by IP. We use UDP as a transport layer protocol in the experiments.

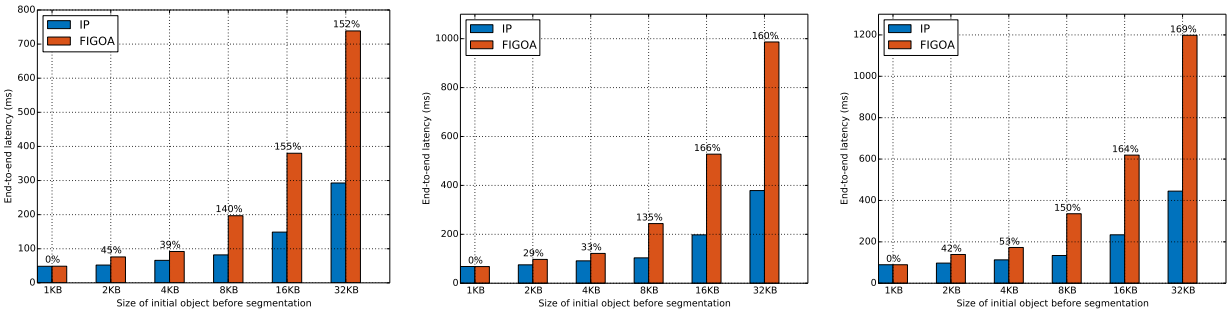
6.5 Evaluation

We employ a server with 8-core Intel i7-3770 CPU at 3.40GHz and 16GB of RAM, and running Ubuntu 12.10 with KVM hypervisor [9]. We construct a testbed by provisioning virtual machines to act as CCNx nodes interconnected on the same LAN and NAT-ed by the host server. Each node is connected to a virtual Ethernet interface at 100Mbps, with MTU set to $1,500$ bytes.

Experiments are run on a 3-, 4-, and 5-node linear topology. The first hop acts as consumer sending interests for specific content published by the last hop – the producer. For each topology, the consumer requests content of size: 1, 2, 4, 8, 16, and 32 KB. We chose linear topology since content objects and fragments thereof always traverse the same path, in reverse, of preceding interests.

To capture the worst case scenario, we ensure that all nodes have an empty cache at the beginning of each experiment. Results in Figure 6.5 demonstrate average consumer end-to-end latency measured from repeated experiments. For all settings, IP performs consistently better than the FIGOA cut-through approach. The bottleneck of FIGOA is that routers need to perform additional processing to compute the hash of every fragment. Since all computation is currently performed in software, these results make sense. However, we believe that if CCN is deployed as a network layer, hash computation would be performed in hardware, with much better performance.

We run another 3-node experiment that involves re-fragmenting fragments. We measure end-to-end latency at the consumer for different values of intermediate router's MTU: $1,500$, $1,100$, and 700 bytes. The consumer requests content of size 4KB. In case of MTU $= 1,500$, content is fragmented at the producer into 4 fragments. Each fragment, except the last one, contains $1,152$ bytes of payload plus fragment header length. Note that $1,152$ is a multiple of SHA-256 block size, which is 64 bytes. However, when MTU drops to $1,100$, payload length



(a) 3 nodes (b) 4 nodes (c) 5 nodes

Figure 6.5: End-to-end latency for retrieval of various sizes of content. IP represents the unmodified version of CCNx, while FIGOA represents modified version of CCNx. Values above bars represent the increased overhead of FIGOA fragmentation as compared to IP fragmentation

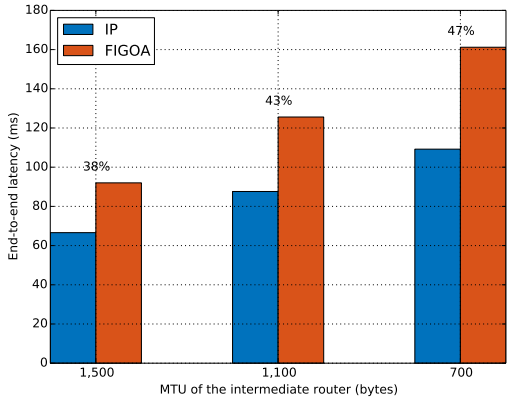


Figure 6.6: End-to-end latency of various MTU values at intermediate routers for content of size 4KB. IP represents the unmodified version of CCNx, while FIGOA represents modified version of CCNx. Values above bars represent the increased overhead of FIGOA fragmentation as compared to IP fragmentation

of each outgoing fragment drops to 768 bytes, leading to re-fragmentation of each fragment into 2 smaller ones. Similarly, each fragment is re-fragmented into 3 smaller fragments when MTU drops to 700.

Results in Figure 6.6 indicate that, as MTU decreases, end-to-end latency increases for fixed content size. This makes sense since smaller MTU leads to more processing due to re-fragmentation. Although re-fragmentation using FIGOA requires additional hash computations at each hop after where re-fragmentation occurs, FIGOA end-to-end latency does not increase dramatically as compared to IP. This is because the CCNx implementation we

are using runs the CCN protocol as an application on top of UDP (and IP), i.e., all CCN operations are handled at the application layer in each router. In this case, if a content object (encapsulated in a UDP datagram) is fragmented by IP, it must be reassembled at every hop before getting delivered to CCNx application layer (which is responsible for forwarding the content to the next CCNx hop). Since the same does not hold for FIGOA, IP reassembly adds more end-to-end latency that compensates for additional hash computation overhead imposed by FIGOA.[10]

[10]Refer to Section 6.2.3 for more details about delay imposed by IP reassembly at each hop.

Chapter 7

Negative Acknowledgments

In communication protocols, there are two ways to confirm whether a packet has been received: acknowledgments (ACKs) or negative acknowledgments (NACKs). In ACK-based protocols, a receiver informs the sender about all successfully received packets. In NACK-based protocols, a receiver informs the sender whenever it believes that a received packet is unrecognized, non-sensical, or corrupted [190]. However, NACKs might be issued for other reasons. Assume a user receives the first and third packets in a transmission. Considering the second one as lost and immediately (or even after a timeout) issuing a NACK for it is not necessarily the right decision. This is because that packet might arrive later due to network delays. We do not discuss this any further since such case is handled differently by various protocols, and there is no guaranteed way for the receiver to know that the second packet is missing or simply delayed.

Since NDN's initial release, producers simply drop interests they can not satisfy. Also, routers behavior is unclear in cases when interests can not be forwarded along, e.g., due to unknown next hop. Even though [203] suggested network-layer NACKs for notifying downstream routers about forwarding failures, the proposed scheme is not adopted in NDN.

However, recent discussions [11] indicate that network-layer NACKs might be adopted by NDN.

CCNx implemented end-to-end network-layer NACKs until version 0.8.2. In the latest release, this functionality was removed from the network-layer. Instead, a new type of CCNx message, `InterestReturn` [142], was introduced. This message is a one-hop notification to the downstream node indicating interest forwarding errors due to no route to destination, congestion, malformed interest, etc. Downstream nodes, however, have the option of ignoring `InterestReturn` messages.

The use of NACKs as an alternative to simply dropping unsatisfiable interests has some advantages. First, NACKs can quickly notify consumers about non-existing content instead of waiting for issued interests to time out. Second, NACKs allow consumers to differentiate between non-existing content and packet (interest) loss. In the latter case, consumers have to wait until an issued interest expires before attempting retransmission. Third, the use of NACKs can help mitigate the effects of Interest Flooding (IF) attacks [77, 54, 22] (see Section 7.1 for details). Finally, NACKs can be useful for notifying downstream routers that received interests can not be forwarded further. Routers can thus pursue alternative paths.

Below, we consider network-layer NACKs from a security perspective. In particular, we discuss two types of NACK messages relevant to CCN: Content-NACKs (cNACKs) and Forwarding-NACKs (fNACKs) (which are similar to `InterestReturn` messages). For each type, we present its benefits for network entities (consumers, producers, and routers) and specify its security requirements. Then, we show that – even with these requirements met – introducing cNACKs has negative security implications for producers, while fNACKs are generally beneficial. Intended contributions of this chapter are:

- Discussion of benefits and scenarios that justify the use of network-layer NACKs.
- Discussion of security requirements for implementing NACKs.

- Showing that naïve security for NACKs can facilitate DoS attacks against producers.
- Demonstrating the effects of NACK-based DoS attacks.

7.1 Content-NACKs

A cNACK is generated by a producer: it indicates that a content – with the name reflected in a received interest – does not exist, i.e., has not been produced or published. A cNACK is realized as a special kind of a content object, of type `CNACK` (set in the `PayloadType` field). One intuitive analogy (though at a higher layer) is the well-known "HTTP 404 not found" message [70].

cNACK generation can involve different layers depending on the application and the type of traffic:

- Application-layer: the application is the sole decision maker on generating cNACK messages. Such cNACKs are transparent to the network-layer.
- Application- and network-layer: cNACKs are generated, i.e. formed and signed (since they are content objects), at the network layer, though the decision is made at the application layer. For every received interest, the network layer needs to consult with the application whether a cNACK should be generated. This approach is clearly inefficient.
- Network-layer: cNACK generation takes place at the network layer without consulting corresponding applications. However, the latter should provide the network-layer with a list of published content names, which can be done *a priori*. Moreover, network-layer cNACKs are only possible for static content. For dynamic content, applications must provide the network layer with the list of prefixes under which content is published.

The network layer should generate fNACKs when receiving interests that can not be forwarded – see Section 7.2 for details.

For the rest of this section, we focus on network-layer cNACKs.

7.1.1 Benefits

cNACKs offer several benefits. On the consumer side, they help applications to: (1) distinguish between *packet loss* and *content not found*, and (2) reduce waiting time for consumers, i.e., inform consumers faster than interest timeouts. For routers and producers, cNACKs can reduce the effects of Interest Flooding (IF) attacks. Recall that a router creates a PIT entry for each distinct interest that it forwards.[1] A PIT entry is not purged until content arrives (from upstream) and gets forwarded downstream. However, if an interest requests some non-existing content and the producer simply drops such interest, corresponding PIT entries at all intervening routers (and at the consumer) remain until they expire. A producer-generated cNACK traverses, in reverse, the path of the corresponding interest and allows routers to purge PIT entries and free resources. Even though this strategy does not fully mitigate the impact of IF attacks, it may reduce their effects.

A router that receives a new interest (i.e., there is no PIT or cache hit) might determine, based on its local FIB, that multiple outgoing interfaces can be used for forwarding. If so, a router either: (1) forwards the interest on multiple interfaces at the same time, or (2) forwards the interest on one interface; in case of a timeout, it tries the next possible interface, and so on.[2] In the latter case (2), a router might incur considerable delay by sequentially trying (and timing out on) every viable interface. However, since cNACKs are generated by the producer to indicate non-existing content, if a router receives a genuine cNACK, trying

[1] Clearly, this excludes collapsed interests.
[2] Other forwarding strategies are possible. However, we do not consider them here.

other possible interfaces would be useless. This early detection of non-existent content is another advantage of cNACKs. Finally, since a cNACK is a type of content and is thus cached, subsequent interests for the same name are satisfied accordingly.

7.1.2 Security Issues

Despite aforementioned benefits, cNACK security implications should be carefully examined. Support for insecure cNACKs opens the door for simple content-focused DoS attacks. Assume that Adv controls a network link and can inject cNACKs for interests traversing that link. In this case, Adv can prevent consumers from obtaining legitimate content. Even if there are multiple paths to the producer, Adv only needs to inject cNACKs on just one path to succeed.[3] This attack can be trivially exploited to enforce censorship over content considered subversive or simply undesirable.

More generally, an unsecured cNACK can be abused to essentially *poison* router caches as described in Chapter 4. Recall that content poisoning attacks can occur in either reactive or proactive mode [77]. The former corresponds to the adversarial scenario above. The latter involves Adv that, anticipating demand for certain content, issues one or more bogus interests (perhaps from strategically placed zombie consumers), ahead of genuine interests being issued. Adv, then, replies with fake content. Thus pre-poisoning the caches of victim routers. Section 4.1 describes this attack in detail.

One variation of proactive content poisoning attack is even simpler. Again, predicting the name of content that has not yet been produced, Adv issues an interest for such content and receives a legitimate cNACK from the genuine producer. Routers on the path cache this cNACK. Even if the actual content is published soon thereafter, subsequent interests for that

[3]Even if adjacent routers use pairwise secure channels, Adv can be a malicious router.

content will be satisfied with a cached cNACK, thus resulting in denial of service for that content.

The above clearly motivates securing cNACKs, which intuitively translates into two requirements: (1) authenticating cNACK origin and integrity, and (2) checking cNACK freshness, i.e., detect replays. We discuss these issues in the next section.

7.1.3 Securing cNACKs

Authentication of cNACK origin and integrity seems easy, since one of the basic tenets of CCN is that all content must be signed by its producer. Universal adoption of the IKB rule (introduced in Chapter 4), would prevent cNACKs from being modified or generated by entities other than legitimate producers.

A complementary means of preventing content poisoning is via SCNs. Using SCNs, a consumer specifies, in the interest packet, the hash of expected content. Routers only need to verify that the hash of a received content matches the value specified in the interest. A key advantage of this approach is that a content matched in this manner does not need to be signed.

However, cNACKs cause a problem for routers when consumers use SCNs. Suppose that a benign consumer requests a content using SCN in an interest. Even though a consumer might have pre-obtained the hash of currently requested content from a legitimate source (e.g., a catalog that it previously obtained using IKB), the content in question could be no longer available from its producer, for various reasons. In that case, the producer would respond to the interest with a cNACK. However, the hash of the latter would certainly not match the content hash reflected in the SCN in the interest. Therefore, such a cNACK would be dropped by routers as an invalid content. Fortunately, this problem can be solved by

a minor modification to network-layer trust rules proposed in Chapter 4: interests bearing SCNs should also (as a backup) adhere to IKB, i.e., reflect the producer's public key, in order to handle (via signed cNACKs) expired or simply no-longer-available content.

Although the motivation for producer-signed cNACKs is not surprising, why this process should take place at run-time might not be obvious. First and foremost, producers can not create and pre-sign cNACKs for all possible non-existing content. This is because content names can have arbitrary suffixes, resulting in an infinite number of possible names. In other words, a producer responsible for a name prefix `/ccn/x/y/z`, should be ready to respond to (in particular, by generating a signed cNACK) an interest requesting any (non-existing) content name starting with that prefix.

To prevent replay attacks, signed cNACKs must include a challenge by the consumer, and/or a timestamp set by producers. However, both approaches have drawbacks:

- If each interest contains a unique consumer-selected challenge, then caching a signed cNACK that also includes the challenge response is useless for other consumers who issue interests for the same content at, or near, the same time. Caching such a cNACK is beneficial only in the case of packet error or loss and retransmission. Moreover, PIT interest collapsing becomes a problem, since each interest to-be-collapsed would have a different challenge. Thus, we conclude that consumer challenges are problematic in cNACKs context.

- If each cNACK contains a producer-set timestamp, a time window ω needs to be defined to allow for transmission and caching delays. The selection of ω poses a problem. If it is too large, cNACK objects can be replayed for a longer time. On the other hand, if ω is too small, the probability of successful replay attacks decreases, while the probability of cNACKs wrongly considered invalid increases. One viable alternative is to use producer-specified expirations for signed and time-stamped cNACKs. This would address cNACK

replay attacks. Nonetheless, we note that timestamps require a global synchronization protocols, e.g., a secure version of NTP [136].

As mentioned above, cNACKs are treated similar to content objects, thus, they might be cached by routers. Although challenge-based cNACKs only benefit from the cache in case of retransmission, timestamp-based cNACKs can be cached and used to satisfy future interests. However, there are some considerations regarding the latter's `ExpiryTime` value. Producers must not set this value to be larger than ω. If a cached cNACK is served after ω elapsed, it should be considered invalid and dropped by receiving consumers.

Based on the above discussion, we summarize the requirements for securing cNACKs:

1. **Signature:** a cNACK must be signed by its producer, similar to any other content.
2. **Timestamps:** a cNACK should be generated, not per interest, but per time interval.
3. **Expiration:** a cNACK for plausible content (e.g., not yet published) should include expiration time.

7.1.4 Secure cNACKs: a Blessing or a Curse?

Unfortunately, secure cNACKs that satisfy our three requirements (which are themselves motivated in part by DoS prevention) facilitate producer-focused DoS attacks. Such attacks occur when Adv sends a large number of closely-spaced interests requesting non-existing (and possibly non-sensical) content. A producer that receives a barrage of these interests generates a cNACK for each one, which requires generating a signature. The resultant computational load on the producer could be overwhelming. Furthermore, large numbers of useless cNACKs would pollute router caches.

Note that generating one cNACK for all interests arriving within a certain time interval is not effective against this DoS attack. This is because a smart Adv, instead of issuing interests for the same (non-existent) name, would issue many interests, each for a distinct name composed of a common prefix (registered to the victim producer) and a random suffix, e.g., `/ccn/bbc/news/world/$&F(?%`. One simple countermeasure is to allow producers to issue cNACKs for prefixes. For example, a cNACK for `/ccn/bbc/news/world/`, once cached in routers, would throttle all interests with that prefix, including non-sensical ones. However, the very same cNACK would result in DoS for legitimate interests, e.g., referring to `/ccn/bbs/news/world/usa`.

The discussion above leads us to a logical conclusion that secure cNACKs should be implemented carefully. Specifically, a producer must first decide whether an incoming interest is plausible or non-sensical. An interest is *plausible* if the producer believes that the referenced content name might have existed in the past or might exist in the future. In contrast, an interest is *non-sensical* if it refers to implausible (or unlikely to ever exist) content name. There is no guaranteed way of distinguishing between these two types of interests. This task is perhaps best left up to individual applications. As far as producer's strategy, we believe that it should have the option of replying with a secure cNACK in response to a plausible interest. However, a producer should not reply at all to a non-sensical interest. This prompts another requirement for securing cNACKs:

4. **Plausibility:** a cNACK should be generated only for a plausible interest.

7.1.5 Experimenting with Secure cNACKs

To assess the efficacy of producer-focused DoS attacks, we performed several experiments, using ndnSIM 2.0 [127], to demonstrate additional overhead imposed by generating a network-layer cNACK per interest. Although, as discussed above, secure cNACKs should be generated

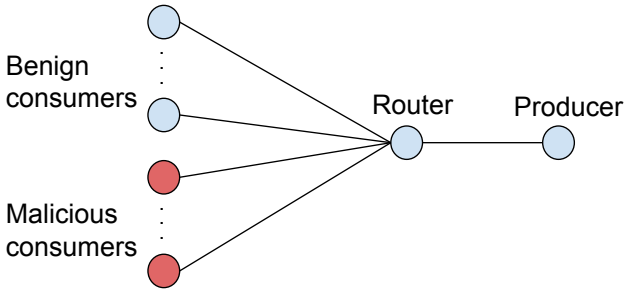


Figure 7.1: cNACK simulation topology

only as a response to plausible interests, a smart Adv can still generate many names that a producer application can consider to be plausible. This can be caused by poorly implemented applications, or by the difficulty of distinguishing plausible from non-sensical names.

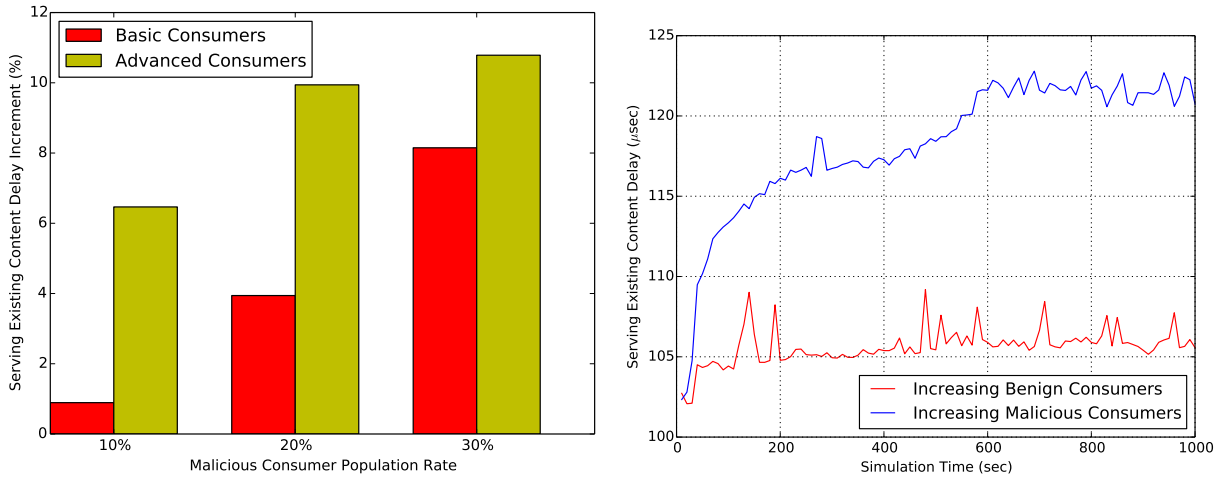
In the experiments, we use the simple network topology in Figure 7.1. We let benign and malicious consumers issue a large number of interests to a single producer at different rates: benign send 10 interests per second; while malicious send 100 non-sensical interests per second. We implement two consumer modes:

1. Basic: consumers request sequential content under a specific name space, e.g., `/ccn/a/1`, `/ccn/a/2`, etc.

2. Advanced: content requested by consumers adheres to a Zipf distribution. This reflects applications where some content is more popular than another.

The main difference between these two modes is that basic consumers do not trigger interest collapsing in routers. Therefore, this mode is suitable for malicious behavior since more non-sensical interests reach the producer, forcing the later to generate and sign more cNACKs.

Figure 7.2(a) shows the delay increment in serving existing content, for both basic and advanced benign consumers. In the base case all consumers are benign. The results show additional overhead imposed on the producer to serve existing content, as compared to the base case, for different malicious consumers population (MCP) rates (10%, 20%, and 30%).



(a) Serving existing content delay increment compared to the base case for varying MCP rates

(b) Serving existing content delay for gradually increasing number of consumers

Figure 7.2: cNACKs experiment results

As expected, increasing the number of malicious consumers increases producer overhead when serving existing content. The overhead is even higher for advanced consumers. The reason is that collapsing of interests requesting existing content reduces the number of these interests on the link between the router and the producer. Therefore, the router forwards more non-sensical interests to the producer.

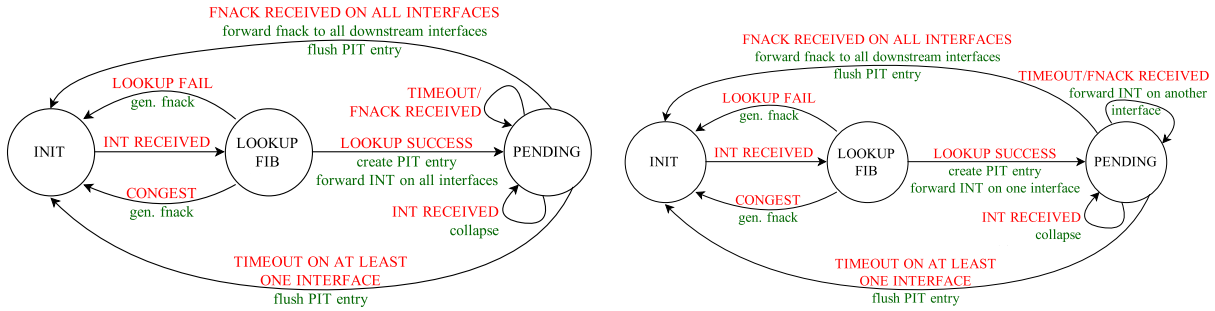
We also explore the delay in serving existing content, as the number of consumers increases. We start the simulation with 200 benign consumers and consider two scenarios: (1) adding one benign consumer per second; (2) adding a malicious consumer per second. In both cases, we stop adding new nodes after 500 seconds and measure the delay in serving content until the 1000th second. The result is illustrated in Figure 7.2(b). Increasing the number of benign consumers does not significantly affect the producer performance, while increasing the number of malicious consumers does (e.g., after 500 seconds, the delay is 10% higher than in the case with only benign consumers).

7.2 Forwarding-NACKs

An fNACK is generated by a router at the network layer. Its purpose is to inform downstream routers that an interest can not be forwarded due to congestion or unknown next hop [203]. Since edge routers are usually configured with a default route to an upstream peer, fNACKs generated due to unknown next hop are most likely to occur at the network core. A good analogy to an fNACK is an ICMP destination unreachable message [164].

We distinguish between the cases of a router *generating* and *forwarding* fNACKs. There are two reasons for a router to generate an fNACK: (1) FIB lookup failure, i.e., an entry indicating the next-hop of the received interest does not exist, or (2) all FIB-specified outgoing interfaces are congested. A router that generates an fNACK, sends it out on each interest incoming interface in the appropriate PIT entry. It then flushes that PIT entry.

Since a FIB might specify multiple interfaces on which interests can be forwarded, a router can forward these interests in parallel or in sequence. A router *must* forward fNACKs on all downstream interfaces (on which interests were received) if it receives an fNACK on *every* upstream interface specified in the FIB, regardless whether parallel or sequential forwarding is used. Conversely, if an fNACK is not received on at least one upstream interface (i.e., at least one timeout occurs) a router *must not* forward fNACKs downstream. This is because a timeout does not imply producer unreachability. A producer might have actually received the interest and decided to drop or ignore it. Figure 7.3 shows two state diagrams (for parallel and sequential forwarding) for generating and forwarding fNACKs.



(a) fNACK parallel interest forwarding strategy (b) fNACK sequential interest forwarding strategy

Figure 7.3: fNACK generation and forwarding state diagrams (red/upper case: events, green/lower case: actions)

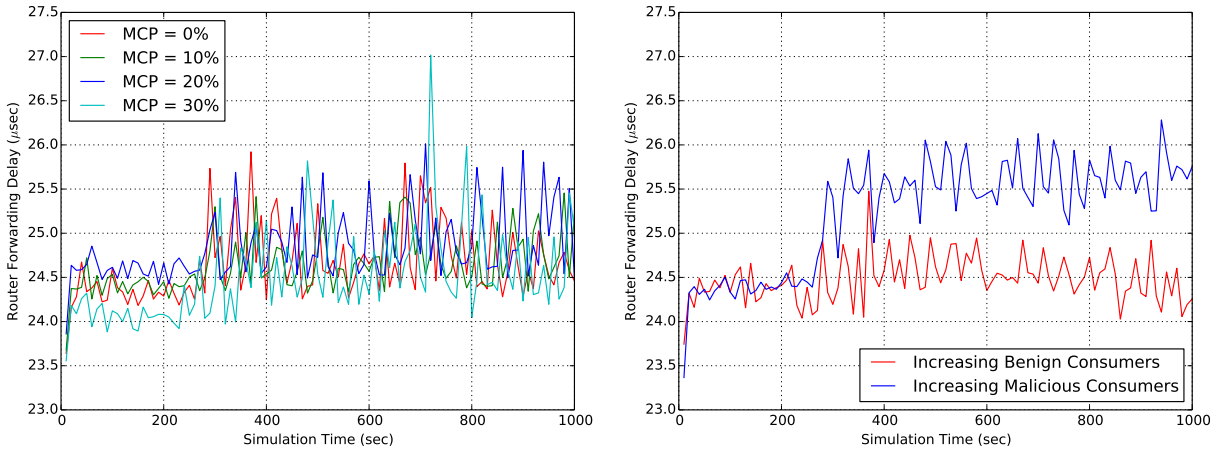
7.2.1 Securing fNACKs

Similar to cNACKs, insecure fNACKs trigger content-focused DoS attacks. Adv controlling a link can inject fake fNACKs in response to interests on that link. This would prevent consumers from obtaining requested content.

Securing fNACKs seems similar to cNACKs, i.e., via origin authentication and replay prevention. However, we can not use the same methods from Section 7.1. If we require each fNACK to be signed, Adv can easily generate many spurious interests that can not be forwarded by a particular router. That victim router would then be forced to sign one fNACK for each spurious interest. Since signing is often appreciably more expensive than verification (e.g., in RSA), computational overhead for the victim router would easily translate into a full-blown DoS attack.[4]

Furthermore, signing fNACK would trigger the need for a routing PKI since verifying fNACK signatures can not be done mechanically: public key certificates must be fetched, verified and revocation-checked. This represents another challenge for supporting signed fNACKs. Note that the IKB rule can not facilitate signature verification in fNACKs. Recall that, in

[4]Note that some digital signature techniques flip this balance, e.g., in DSA, verification is more expensive than signing. However, the DoS attack would then be even worse since multiple downstream routers would verify the fNACK signatures.



(a) Router forwarding performance for different MCP rates

(b) Router forwarding delay for gradually increasing number of consumers

Figure 7.4: fNACKs experiment results

IKB, a consumer must specify the public key of the *producer* expected to sign the content. Since fNACKs are signed by upstream routers, downstream peers can not accept them as a legitimate response. This is because their public keys do not match the ones specified in the corresponding interests. Since consumers can not know in advance whether an fNACK will be generated and by what router, they can not include the correct key digest in interests. One way to mitigate this is for consumers to specify the public keys of all routers on the path to the producer. This approach is impractical because it requires a path discovery-like protocol, as well as increases the size of interests, complexity of interest collapsing, and router overhead.

If we assume that long-term trust relationships can be established between neighboring routers, fNACK authentication can be easily achieved. In this case, fNACKs can be sent downstream over a sequence of pair-wise secure channels between neighboring routers. Therefore, one trivial way of securing fNACKs hop-by-hop is by using a keyed hash, e.g., HMAC [106]. Replay prevention can be achieved via timestamps, considering that adjacent routers are likely to maintain closely synchronized clocks.

7.2.2 Experimenting with Secure fNACKs

We ran several experiments using ndnSIM 2.0 to demonstrate negligible impact of secure fNACKs. We use the same topology as in Figure 7.1. Benign consumers request 10 content per second, while malicious consumers request 100, which can not be forwarded by the router. Our evaluation metric is processing time for the router to forward an interest towards the producer. All consumers (benign and malicious) implement the basic mode.

We implement two scenarios. In the first, we compare router forwarding performance for different rates of MCP (0%, 10%, 20% and 30%). The total number of consumers in this scenario is 200. Figure 7.4(a) shows that even with 30% MCP rate, router forwarding performance is not affected. In the second scenario, the number of consumers increases gradually. Initially, there are 200 benign consumers. We then either: (1) increase the number of benign consumers (one every second) until reaching 700, or (2) introduce 500 malicious consumers (one every second). Figure 7.4(b) illustrates the results: for up to 300 malicious consumers, router performance is similar to the case where the network contains only benign consumers. Even if the number of malicious consumers exceeds 300, router performance decreases only by an average of 4%.

7.3 Mitigating Producer-Focused DoS Attacks

As discussed earlier, securing cNACKs comes at a price of possible DoS attacks on content producers. We now discuss some ways to mitigate such attacks.

In doing so, we separate content-serving and cNACK-generation activities. Producers can set up special-purpose gateways that distinguish between interests requesting existing and non-existing content. The former are forwarded to the actual content repository that serves requested content, while the latter are forwarded to a special server that generates and

signs cNACKs. However, this only works for static content because producers need to keep gateways updated with all published content. This is not applicable for dynamic content generated upon request.

By redirecting the attack towards the cNACK generation server, producers can still continuously serve content. However, the network needs to deal with attack traffic, which might consume a lot of bandwidth. Moreover, routers have to create PIT entries for all interests since they can not differentiate between interests requesting existing and non-existing content. If routers could differentiate, DoS attacks would be preventable closer to their sources.

One mitigation approach is to let a producer relay all its published content names to routers. A producer can use these names to construct a Bloom filter [45] and disseminate it to routers processing interests for this producer. Dissemination depends on producer's policies. For instance, a producer can fall back on Bloom filters when its load of generating and signing cNACKs reaches a certain threshold.

Bloom filters are created by producers periodically, or whenever new content objects are published. They can be represented as content objects of a special type, e.g., `BLM-FLTR`. However, Bloom filters, if cached, should not be used to satisfy future interests. Moreover, caching duration depends on the `ExpiryTime` value included in their headers. Producers need to carefully set this value to be compatible with the frequency at which new content is being published, e.g.,

$$\texttt{ExpiryTime} = \frac{1}{\text{avg}(f)|_{\tau}} \tag{7.1}$$

where the denominator represents the average value of content publishing frequency over a specific period of time τ.

The size of a Bloom filter depends on the number of elements (content names) it contains. Large content objects can be divided into smaller segments, each with a unique name. The size of each Bloom filter should be upper bounded by the maximum size of a content segment. This avoids the case where a Bloom filter is split across multiple segments, thus requiring multiple interests to request the whole filter. Since producers disseminate Bloom filters as a reply to a single interest, they should fit in a single content segment.

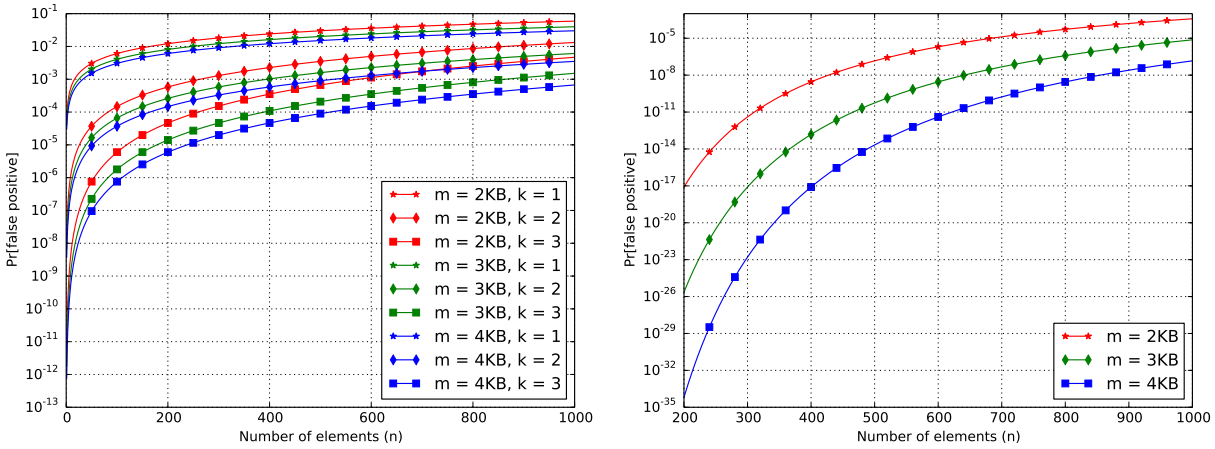
On the other hand, Bloom filter's false positive probability (illustrated in Figure 7.5) depends on its size m (in bits), the number of elements n in the set $\mathbb{S}$, which are included in the filter[5], and the number of hash functions k. This probability increases as n and k increase and decreases as m increases. Assuming that hash functions $(h_1, \ldots, h_k)$ map each element of $\mathbb{S}$ into a random value uniformly distributed over $[1, \ldots, m]$, the false positive probability can be expressed as in Equation 7.2 [48].

$$\begin{aligned} \Pr\left[\text{false positive}\right] &= \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \\ &\approx \left(1 - \mathrm{e}^{-\frac{kn}{m}}\right)^{k} \end{aligned} \tag{7.2}$$

Figure 7.5(a) illustrates it for varying m, n, and k. However, for a given m and n, k can be optimized. In this case, the false positive probability can be computed using Equation 7.3 [48].

$$\Pr\left[\text{false positive}\right] = (0.6185)^{\frac{m}{n}} \tag{7.3}$$

[5]In this case, $\mathbb{S}$ is the set of published content names.



(a) Variable filter size, number of elements in the set $\mathbb{S}$, and number of hash functions used

(b) Variable filter size and number of elements in the set $\mathbb{S}$, and optimized number of hash functions used

Figure 7.5: Bloom filter false positive probability

In practice, producers can optimize k in order to achieve lower false positive probability. However, an upper bound for k can be set to limit the hashing overhead for routers. Figure 7.5(b) demonstrates the Bloom filter's false positive probability when varying m and n, and optimizing k.

Based on the plots in Figure 7.5, including all published content names in a single Bloom filter (the size of which is upper bounded by the maximum size of a single content fragment), might not lead to desired false positive probability. However, producers can create a separate Bloom filter for each namespace (or sub-namespace) they publish. Therefore, achieving desired false positive probability requires an upper bound on the number of content objects published under each namespace. It might also require redesigning the namespace hierarchy of producers. We do not discuss this optimization problem any further since it is out of the scope of this chapter. Moreover, the number of interests requesting non-existing content (that forwarded to producers due to the probabilistic nature of Bloom filters) can be minimized with proper configuration of filters parameters.

Although Bloom filters are content objects that follow the same path, in reverse, of their corresponding interests, they should not be delivered to consumers. This is because malicious

consumers might benefit from these filters. Adv can pre-compute a list of content names that pass verification and use it to launch a distributed DoS attack against the target producer.[6] Such attacks can be circumvented if edge ISPs do not forward Bloom filters towards their customers, or filters are re-created periodically with different parameters.

[6]We do not consider the case where malicious routers deliver Bloom filters to malicious consumers using side channels.

Chapter 8

Related Work

In this chapter, we provide a detailed overview of prior work related to this dissertation. We start by reviewing the security and privacy features of other FIA projects: MobilityFirst, XIA, and NEBULA. We then discuss related work on: cache privacy, network-layer trust, accounting, secure fragmentation, and negative acknowledgment.

8.1 Security and Privacy of FIA Projects

In this section, we discuss different security and privacy features of the FIA projects.

8.1.1 Trust

NEBULA's ICING-based network layer defines trust in a way that is orthogonal to IPSec. Trust is defined between a host and all nodes forwarding its packets. As described in Chapter 1, a host agrees on a "contract" with the network providers carrying the data (i.e., the path negotiated using NVENT) to specify the operation executed at each hop. Such contracts

are cryptographically enforced. ICING assumes mutual trust between forwarding nodes and their consent servers, i.e., servers responsible for creating PoC tokens. Therefore, this notion of trust does not require any PKI [145]. However, ICING does not provide an end-to-end definition of trust, which can be added by adopting an IPsec-like approach.

MobilityFirst places trust in principals. Depending on their type, trust may be established with (1) hosts, similar to IPsec, (2) content, similar to CCN, or (3) centralized or distributed services. Trust semantics in XIA also vary depending on principal types. However, the intrinsic security feature of these principals (described in Chapter 1) increases trustworthiness of end-to-end communication and content retrieval. For instance, ensuring that a content hash matches its identifier allows receivers (and caching routers) to trust that content.

As shown in Chapter 1, an XIA address consists of a DAG containing a (partial) path to the destination. To provide trusted path selection for host-to-host communication, SCION is integrated with XIA [146]. SCION [207] is an architecture that provides control and isolation for secure and highly available end-to-end communication. The network is divided into multiple trust domains consisting of several Autonomous Systems (AS-s) that trust each other. Each domain has a trusted root AS responsible for relaying packets to and from other domains. Roots initiate path establishment to all hosts in their domains based on local policies and available bandwidth. This process results in constructing a path between each host and its domain root. Whenever two XIA hosts, in different domains, want to communicate, the two half paths (from each host to its domain root) are combined to establish a complete end-to-end path. Such path is trusted since it is created by the trusted roots of each domain.

8.1.2 Authentication and Integrity

NEBULA's ICING-based network layer does not directly provide data origin authentication. Instead, it is delegated to applications. ICING allows a sender to authenticate entities

issuing cryptographic tokens, i.e., PoCs. However, the design does not specify how PoCs are retrieved or authenticated [145].

Similarly, MobilityFirst and XIA do not provide any data origin and entity authentication at the network layer. However, their usage of self-certifying identities as principal identifiers facilitates entity authentication. Recall that for host, network, and service principals, identifiers are generated by computing the hash of the public key associated with these principals. Therefore, entity authentication can be achieved by ensuring that such principal identifiers match their keys. Peer authentication for content principals can be achieved similar to CCN since such principals are self-authenticating. Neither MobilityFirst nor XIA provides a secure mechanism for securely retrieving content identifiers.

As for data integrity, each packet in NEBULA carries a sequence of cryptographic verifiers V_j, one for each hop on the path. The packet hash is used as part of V_j's calculation. Therefore, ICING guarantees that neither the packet nor the path can be modified. Also, ICING is recommended only at domain gateways [145]. Thus, integrity can only be guaranteed by border routers. Within domains, such guarantees are deferred to either the network-layer protocol or the application.

In both MobilityFirst and XIA, integrity is only available for content principal types. This is again due to the fact that such a principal identifier is generated based on the content hash itself. Whenever a content is received, its hash is compared with its identifier to ensure content integrity. For other principal types, MobilityFirst and XIA defer integrity guarantees to the application.

8.1.3 Authorization and Access Control

In NEBULA, paths must be established before communication begins, i.e., clients must obtain required PoC tokens. Therefore, access control can be implemented by the consent server granting or denying PoC requests. Traffic sent without valid PoC tokens can be easily detected and dropped.

Since MobilityFirst and XIA can support different principal types, they facilitate the combination of both CCN- and IP-based access control schemes. For content principals, access control is done at the content granularity, similar to CCN, i.e., content is encrypted using keys disseminated to only authorized users. For all other principal types, ACLs can restrict access to hosts and other network services.

8.1.4 Privacy and Anonymity

Hosts anonymity is not provided by ICING-based NEBULA. By inspecting packet headers, eavesdroppers can easily determine a packet's source, as well as the path it traversed. However, host anonymity can be achieved by replacing ICING with TorIP [117], thus resulting in a level of anonymity similar to that provided by TOR in Today's Internet.

MobilityFirst and XIA suffer from the same privacy and anonymity problems as IP. Packets contain both source and destination GUIDs (or principal identifiers), thus revealing the hosts involved. To make the matter worse, XIA packets path can be revealed by inspecting their destination DAG addresses. This is because such addresses might include (part of) the path to the destination, as described in Chapter 1. Due to the communication model similarity of these two architectures to IP, approaches developed to preserve users anonymity in IP networks can be adopted. For instance, TOR can be used to protect MobilityFirst and XIA

host principals' anonymity. Preserving content principals' anonymity can be achieved using a protocol similar to ANDāNA in NDN [65].

8.2 DoS and DDoS Attacks on FIA Projects

We now present some DoS and DDoS attacks that apply to the current as well as the future Internet architectures discussed above. We also discuss new types of attacks made possible by FIA architectures.

8.2.1 Bandwidth Depletion Attacks

The current Internet architecture is susceptible to bandwidth depletion attacks [184]. Their goal is to exhaust bandwidth of a specific link. These attacks can be mounted in two ways: (1) distributed – with packets sent at low rate by each attacking node, or (2) centralized – a single powerful adversary flooding the target link at high rate. Due to today's high bandwidth and redundancy, centralized bandwidth depletion attacks are harder to mount.

Several mitigation and prevention techniques have been proposed and implemented in the current Internet. Some examples are: (1) tracing back traffic to the source of the attack [40, 182, 175], (2) distinguishing between legitimate and malicious traffic [30, 201], (3) using puzzles to increase the cost for adversaries trying to consume bandwidth [61, 95], and (4) using rate-limiting mechanisms for traffic that causes congestion [187, 125]. However, none can effectively defeat this attack.

Bandwidth depletion in the data plane is harder to mount in NEBULA, because senders (adversaries) must obtain consent of all nodes on the path before sending packets. Thus, unauthorized packets will be dropped by adversary-facing routers. Unfortunately, this only

shifts the attack from the network layer to the consent servers and causes negative effects on the network. The reason is because a single consent server might be responsible for a large number of routers in its domain. Thus, lowering its ability of issuing PoCs can disable all routers in that domain.

Since MobilityFirst and XIA use a communication model similar to IP, they both are susceptible to bandwidth depletion attacks. Similar countermeasures applied in IP networks can be adopted. However, they can only reduce the effect of these attacks.

Nugraha et al. [150] suggested integrating STRIDE with XIA to protect against DoS attacks. STRIDE [88] is an architecture resilient to bandwidth depletion (D)DoS attacks. It modifies SCION path establishment to perform a tree-based bandwidth allocation. Whenever a trusted domain root initiates the path establishment process, bandwidth is allocated as the path is branching out as a tree from that root. This guarantees the required bandwidth for benign flows. STRIDE also supports long-term static paths to provide high available connectivity.

8.2.2 Routers Resource Exhaustion

Exhausting storage resources of routers is another target for adversaries. As mentioned in Chapter 1, NAT-enabled router might assemble fragments in some scenarios. Reassembly buffers in these routers can be exploited as follows. Each fragment includes a 16-bit field to indicate the size of the original packet. Adversaries can send a single fragment with a large original packet size and never send the rest of the fragments. This forces assembling routers to allocate a buffer and wait for the rest of the packet to arrive. To make the matter worse, adversaries can set the original packet size to its maximum value, 64KB, thus allocating maximum buffers.

Computation resources of routers is another victim of exhaustion. Since ICING's design requires the extensive use of cryptographic operations, adversaries can send a large number of packets to routers forcing them to perform all verification operations described in Chapter 1. Such attacks have very low cost on adversaries since the latter can flood victim routers with packets carrying invalid (e.g., randomly generated) PoC and PoP values.

8.2.3 Cache-Related Attacks

As discussed earlier caching opens the door for new types of DoS attacks, such as content poisoning (Chapter 4). This attack occurs when adversaries inject fake content into router caches. A fake content is not generated by a benign producer and, consequently, does not satisfy user requests. If such content is cached in routers, it is used to reply to future benign user requests.

Due to the use of network caching in MobilityFirst and XIA, both are susceptible to content poisoning attacks, because forcing routers to verify data authenticity before caching might incur undesirable overhead. Fortunately, using SCNs can limit the effect of these attacks. Since MobilityFirst and XIA use the hash of content principals to compute their identifiers, content poisoning attacks are obviated. As mentioned above, MobilityFirst and XIA do not support secure retrieval of content principal identifiers.

Cache Pollution is another type of (D)DoS attacks against router caches. Both Mobility-First, and XIA are susceptible to this attack (see Section 2.6 for more details about cache pollution).

8.3 CCN Cache Privacy

There is a large body of work on using side channels to extract information about other users' (or applications') behavior. Techniques proposed in [71, 82] allow malicious websites to learn whether a user visited a specific web page. The attacker sends a Java applet to the victim and detects cache hits with respect to the user's browsing cache.

Similarly, Felten et al. [69] show how a malicious website can determine whether a web page has been downloaded by its victim. The attack uses a Java applet or Javascript code and timing information to determine the content of the browser's cache.

Baron [36] proposes a countermeasure for attacks in [71, 82], based on completely hiding one's browsing history: rendering behavior of the browser (e.g., link colors, output of CSS functions) with respect to previously visited web pages is identical to that with new pages. However, this technique does not work for interactive attacks. In particular, Weinberg et al. [197] conducted experiments to show that interactive and timing attacks can still be used to disclose user's previous visited sites.

Bortz et al. [46] show two types of timing attacks that allow the adversary to learn the content of the browser's cache. The first, called *direct timing attack*, reveals whether one or more public websites have been visited by the victim. The second, *cross-site timing attack*, is more dangerous as it can reveal information about private sections of websites. For instance, it can determine whether a user is logged in to a specific service.

Another side-channel exploit is the timing attack on SIP VoIP networks. A tool described in [205] can be used to reveal the "calling history" of a SIP domain by observing which "recipient digital certificates" are stored in the local cache.

Several countermeasures to cache attacks have been developed. [94] proposes a server-side approach that prevents users from leaking the content of their cache. The idea is to randomize

and personalize the links in web pages. Thus, a malicious site can not guess them when it tries to discover whether they have been visited.

Schinzel [176] discusses three techniques for mitigating timing-based side channel attacks in web applications. The first delays *all* responses such that the total delay of each response is identical. While this does not leak any information, it introduces considerable delay and affects user experience. The second approach entails adding a random delay to *each* response (the responses to identical requests are independently randomized). However, by requesting the *same* content sufficiently many times, the adversary can remove this random noise. The third approach is similar to the second. However, instead of randomizing the delay per response, a single random delay is selected per destination in order to prevent the aforementioned attack.

Lauinger [110] considers several NDN-related security issues, identifies the problem of cache privacy and overviews several countermeasures, including some approaches similar to those in Chapter 3. Crosby et al. [58] investigate how network latency deteriorates due to time-based side channel attacks, and design filters to reduce the effects of jitter.

Finally, Mohaisen et al. [139, 140] study cache privacy in ICN (NDN) and propose countermeasures similar to ours, discussed in Chapter 3. However, this work only considers privacy-preserving delays at edge routers. Proposed methods require keeping per-user state in routers, in order to enable fast re-transmission of replies without artificial delays. However, per-user state makes such techniques more complex, while privacy information in our approach is distilled in a single bit and timestamp per content (and request). Also, [139, 140] does not consider advanced distributed adversaries described in Chapter 3.

8.4 CCN Network-Layer Trust

Content poisoning was identified in [77], where some tentative countermeasures were proposed. To the best of our knowledge, Chapter 4 presents the first comprehensive study of content poisoning attacks in CCN.

Some prior research efforts discussed naming in content-oriented networks and its relationship to security. Notably, [79] proposes establishing bindings between three ICN entities: (1) identity coupled with the producer of each content object, (2) name, and (3) public key used to verify the content signature. Only two of the three possible bindings (identity–name, name–key and identity–key) are required, while the third can be transitively derived.

Self-certifying naming was discussed in [79, 80, 68, 104]. Names are of the form P:L where P is the digest (hash) of the producer's public key, and L is a label set by the producer. It is the latter's responsibility to make sure that names of this form are unique. This guarantees the name–key binding and sacrifices human readability of names for strong security properties. Although, CCN uses human-readable names, name-key binding is achievable by adding the `KeyID` field to interest messages and content objects. Also, hierarchical names in CCN facilitate interest forwarding based on longest-prefix matching of their names. In contrast, using the P:L scheme would result in very large routing tables.

Trust and trust management are well studied in the literature, especially, in distributed environments, such as MANETs and wireless sensor networks (WSNs). [52] surveys the state of the art in trust management systems for MANETs. It emphasizes the need to combine the notions of "social trust" with "quality-of-service (QoS) trust". A similar survey is [153]. [122] presents an extensive review of trust management systems in WSNs. [204] discusses security challenges in designing WSNs. It distinguishes between the definitions of trust and security and shows that cryptography is not always the solution for trust management.

Since a single trust metric might not suffice to express trustworthiness of nodes, a multi-dimensional trust management framework is suggested in [113]. Three metrics are used: (1) node collaboration while performing tasks, e.g., packet forwarding, (2) node behavior, e.g., flagging nodes that flood the network, and (3) correctness of node-disseminated information, e.g., routing updates.

[56] proposes a framework for calculating a network entity's reputation score based on previous interactions feedback. Each service can apply its own reputation scoring functions. It also supports caching of trust evaluations to reduce network overhead and provides APIs for reporting feedback and calculating reputation scores.

All aforementioned techniques involve keeping track of other nodes' behavior in order to decide whether they are trusted. However, this general strategy is a poor match for CCN because routers need an efficient mechanism to trust content, and not other entities. Since content can be served from any node, it is impractical for routers to trust many other routers.

8.4.1 Cache Poisoning

Cache poisoning has been extensively studied in different contexts. Whenever an architecture involves a cache, this attack type is possible. Address Resolution Protocol (ARP) (that resolves the mapping between MAC/IP addresses [76]) is susceptible to cache poisoning attacks by design. Each node that runs this protocol has an ARP cache. The original design of ARP did not consider malicious behavior. Adversaries can reply with bogus MAC/IP mappings that will be cached by other nodes. ARP cache poisoning is a preparatory step for many network attacks such as DoS, Man-In-The-Middle (MITM), and host impersonation.

There are several countermeasures to ARP poisoning. `arpwatch` [10] is a tool that observes ARP protocol messages and keeps track of "potentially" valid MAC/IP mappings. Whenever

an anomaly is detected, `arpwatch` notifies the network administrator. Snort [18], an Intrusion Detection System (IDS), follows a similar procedure in detecting ARP poisoning. However, administrator incompetence and high false positives might raise performance inefficiencies.

Nam et al. [144] proposed a voting-based resolution for mitigating ARP poisoning attack. When a new node boots up in a LAN environment, all other nodes send it their MAC/IP mapping of the network gateway. This allows the new node to detect any malicious mapping advertisements for that gateway. Trabelsi et al. [191] proposed the use of a stateful cache and applied a fuzzy logic approach to detect MAC/IP mapping anomalies. In this model, nodes share trust information about all other nodes. Unlike the current design, when an ARP request is sent, the sender waits to receive all replies and then selects (based on trust information) the most trustworthy. However, this increases end-to-end latency.

Another type of cache poisoning attacks is DNS spoofing. It is also the first step in mounting MITM attacks. Klein [102] identifies several vulnerabilities in DNS daemons that can be exploited by adversaries to modify valid DNS entries. This results in redirecting legitimate users to false destinations.

DNSSEC [32] solves the DNS cache poisoning attacks by ensuring the authenticity of DNS responses using signatures. A response is trusted only if the signature is successfully verified. Moreover, current DNS [138] provides a built-in mechanism for lower-level DNS server public keys distribution. End-users only need to trust the root servers which provide the key of the next DNS server in the certification hierarchy.

Some attempts were made to improve DNSSEC's performance. Sun et al. [188] proposed a new DNS client that queries multiple DNS resolvers instead of only one. Bassil et al. presented S-DNS [37], a secure and backward compatible protocol that provides lower computation and communication overhead, by replacing the traditional PKI with identity-based encryption. Perdisci et al. [156] proposed WSEC DNS that exploits wildcard domain names

and the fact that all DNS responses must copy the information in the corresponding requests. Using widecards increases entropy of DNS requests and greatly complicates guessing the response. WSEC DNS is easily deployable due to its fully backward compatibility with DNS.

8.5 Accounting in CCN

Network-layer accounting in CCN and other interest-based ICN architectures remains an open topic [200]. However, certain economic aspects, such as *how* to set and enforce prices, has been discussed in [31, 159], which imply an application-layer strategy wherein payment (not usage) information is willingly sent on behalf of the consumer. This conflicts with the approach advocated by Agyapong et al. [25], wherein only ISP-related entities are involved in payment coordination. [25] considers payment as an application layer concern. Accounting techniques presented in Chapter 5 facilitate a blend between these two schemes in which usage and payment information is sent *autonomously* by the network layer. ISP entities and producers are informed of usage information for billing purposes, and can follow up with payment collection.

Patané et al. [155] study a similar problem in the context of IP networking, focusing on Content Distribution Networks (CDNs) and transit networks that channel traffic between ISPs. Payment policies proposed in [155] are identical to each other, though. All parties pay for resources used to deliver their content. [155] does not describe how this payment and usage information can be propagated. Similar to [159], Kocak et al. [103] discuss how content providers can coordinate price information and contracts between ISPs. They also opt for an open, unfederated approach, which fits with the proposed model of autonomous accounting information propagation described in Chapter 5.

[193] proposes the Secure E-Commerce Transactions for Multicast Services (SETMS) framework. It is an overlay functionality on top of the core network and supports significantly more functionality than the proposed accounting scheme, e.g., dynamic subscription to content, secure e-payments, and authentication. Our approach is more low-level in that it can serve as a *building block* for such a framework. However, it does not replace it.

In the context of multicast communication, [172] presents a distributed management architecture for IP multicast services. Agents (e.g., routers) individually collate information about multicast groups traffic that is later used for billing purposes. This information is not propagated to producers in real-time.

Another important element of Chapter 5 is the generation of secure consumer-specific data in *pInt* messages. There is on-going work on the topic of packet-level authentication in the current Internet [108]. However, digital signatures and symmetric-key MACs require some unrealistic assumptions, such as shared keys amongst all pairs of routers, and trusted third parties responsible for key generation and management. Public-key algorithms based on elliptic curves offer better signature efficiency (e.g., DNSCurve [6, 41]). However, the sheer volume of interests in CCN would likely be substantially higher than DNS queries in IP networks, leading to only modest performance gains.

8.6 Secure Fragmentation in CCN

8.6.1 Secure Fragmentation

The first attempt to address security implications of IP fragmentation is [192] which investigated how to efficiently authenticate IP packet fragments in egress/ingress routers of stub AS-s. A source host is assumed to share a key with appropriate router(s). Two techniques are

proposed: The first one is delayed authentication (DA) [192] where an authenticating router verifies a packet MAC incrementally from its fragments. To prevent reassembly of a corrupted packet at the destination, an authenticating router holds one small fragment "hostage" until authenticity of the entire packet authenticity is confirmed. The second scheme is an MTU probe mechanism that a source host can use to pre-segment a large packet into smaller authentic packets sized to the smallest MTU on the (current) path. Some extensions to [192] were later proposed in [163]: extended delayed authentication (EDA) requires fragments to always traverse the same path. [163] also provides a detailed comparison of several secure fragmentation techniques.

A secure fragmentation scheme for Delay-Tolerant Networks (DTN) [51] is presented in [154]. This scheme is referred to as "toilet-paper" approach. The basic idea is that, prior to bundling, data is check-pointed into fragments with a cryptographic hash at specified intervals. Hashes are included in a bundle and authenticated with a signature. Gateways forward a fragment only if its hash and signature are valid. A variation supports variable increments of authentic fragments, allowing routers greater flexibility to choose fragment size, thus potentially saving link resources.

An enhancement to the "toilet-paper" approach is presented in [34]. After fragmentation, the sender constructs a Merkle Hash Tree (MHT) [135] where each fragment is placed in a leaf, and signs the root. Fragment verification requires knowledge of this signature and $\log(n)$ hashes (where n is the number of fragments). To authenticate all fragments, a verifier computes $n \log(n)$ hashes and a single signature verification. However, these approaches are not applicable in CCN since they lack support for in-network fragmentation. Moreover, in FIGOA, verifying all fragments requires only n hashes and one signature verification.

Most current networks employ IPsec [72] for network-level authentication in IP networks. It is compatible with both IPv4 and IPv6. IPsec operates in transport and tunnel mode. Transport mode is used by two hosts to establish a security association to authenticate and

encrypt IP packet payloads. Tunnel mode facilitates Virtual Private Networks (VPN) which are composed of IPsec-enabled gateways that share bilateral security associations. Gateways encrypt IP datagrams and encapsulate them into new ones.

Regardless of the mode, fragmentation between IPsec-enabled hosts (gateways) occurs at the IP layer. Since IPsec authenticated/encrypted packets reflect the destination address of another IPsec-capable host (or gateway), they must undergo packet-level scrutiny which requires reassembly. Hop-by-hop reassembly at IPsec adjacent hosts ensures security.

8.6.2 Fragmentation in ICN

NDN currently supports TCP/UDP tunnels to interconnect its forwarders. Fragmentation is relegated to IP, limiting maximum packet size to that of IP. Hop-by-hop reassembly allows routers to authenticate content (although this is not presently supported in the forwarder implementation) at the increased cost of reassembly.

NDN Link Protocol (NDNLP) [180] alleviates this issue while operating over both link-layer and virtual transports, such as Ethernet and TCP/UDP. Fragmentation occurs for both interest and content packets. NDNLP supports intermediate reassembly which makes it compatible with NDN security requirement. However, it uses a packet format incompatible with NDN. NDNLP also supports reliable transmission of fragments.

The CCN-lite project [50] aims to provide a "level-0" forwarder for CCN. It is compatible with the CCNx protocol and provides a rudimentary implementation of the forwarder with simple data structures for PIT, FIB, and CS. Native fragmentation and reassembly are supported over Ethernet and TCP/UDP. Fragments are identified by a sequence number without any addressing scheme on per-fragment basis, which implies that cut-through fragmentation

is not supported. This fragmentation scheme also allows for reliable re-transmission of individual fragments

ICN Transport Protocol (ICTP) [173] is a scheme, which implements TCP native to ICN. Similar to TCP, ICTP segments data to avoid further fragmentation. This provides cut-through delivery of fragments. Akin to TCP, this scheme does not prevent fragmentation from occurring at a lower layer. Unlike FIGOA, ICTP does not address content authentication at intermediate routers.

The NetInf project [67] is another emerging ICN architecture which supports location-independent Named Data Objects (NDO) (similar to content objects in CCN). NDOs are signed and cacheable units. NetInf does not perform segmentation and relies on a "convergence layer" (CL) to synthesize necessary services for heterogeneous transports used to connect NetInf gateways. CL is tasked with fragmentation and reassembly of NDOs. With no native cut-through fragmentation scheme, NetInf appears to rely on hop-by-hop reassembly for verification of NDO authenticity.

8.7 Negative Acknowledgments in CCN

In the current Internet, negative acknowledgments are used as error notifications in control protocols. For instance, in TCP, Automatic-Repeat-Request (ARQ) implements error control using Go-Back-N and Selective Repeat [114]. In Go-Back-N, receivers detecting a lost packet send a notification indicating the missing packet. The sender, then, re-transmits everything, starting from the lost packet. In Selective Repeat, receivers use notifications to report packet loss. In this case, the sender only resend that specific packet.

In broadcast communication, NACKs are preferred over ACKs due to lower congestion and fewer packet collisions [160]. This is because using selective NACKs allows lowering the

number of packets sent by receivers, hence reducing the probability of packet collision closer to the sender. This, however, is prone to NACK implosion. In case of packet loss, the sender receives many NACKs from all receivers. Stran et al. [174] propose a time-based mechanism to reduce NACK implosion. Every receiver detecting a packet loss initiates a random timer. The receiver having the shortest random interval unicasts a NACK to the sender, which immediately multicasts the NACK to the other receivers. All receivers having the same missing packet suppress their own NACKs. Yamamoto et al. [202] demonstrate that the delay incurred by a NACK-suppression mechanism does not affect the performance of NACK multicast control flow.

In wireless networks, selective NACKs can be used in RTS/CTS handshake mechanism in order to reduce congestion and packet collision rates [90, 171]. In [115], NACKs at the data-link layer are combined with NACKs at the transport layer in order to improve video streaming performance over 3G cellular networks. In case of frame loss, a mobile device sends a selective data-link NACK to the base station. If the lost frame is not recovered after several successive NACKs, a transport-layer NACK is issued to the sender requesting re-transmission of the entire packet.

At the transport layer, NACKs are used for reliable communications [21, 20, 89, 152]. [21, 20] describes the NACK-Oriented Reliable Multicast (NORM) Transport Protocol. NORM operates between one or more senders and a group of receivers over an IP multicast network. Receivers use selective NACKs to notify senders about lost packets. In a similar approach [89], NACKs are used as a packet loss detection mechanism in satellite communication. A NACK is generated by sending a signal. Senders detect NACKs by monitoring the total electrical power in the frequency band used for the uplink. This enables several receivers to share a low-speed uplink circuit simultaneously, which prevents NACK collisions. Obraczka [152] surveys multicast transport protocols, including NACK-based protocols, ACK-based protocols and other hybrid approaches.

Chapter 9

Conclusions and Follow-On Work

This dissertation addressed several issues that stem from in-network caching in Content-Centric Networking (CCN).

Chapter 3 explored cache privacy in CCN, identified several important privacy threats, and introduced some plausible and effective countermeasures. We suggested that consumers and producers should indicate which content is privacy-sensitive. We also proposed several techniques that provide certain trade-offs between privacy and latency. These techniques were assessed with respect to local and distributed adversaries with varying capabilities. We also introduced a formal model to quantify the degree of privacy offered by various caching algorithms. We believe that proposed techniques are general and may be of interest beyond caching. Follow-on topics include: (1) analyzing different possible artificial delay periods added by edge routers and studying their effect on content distribution performance, and (2) introducing techniques for consumers and producers to link distinct private content in order to prevent correlation attacks.

NDN is one of several prominent candidates for the next-generation Internet architecture. It is susceptible to some new threats, such as content poisoning, whereby adversaries inject fake

content into router caches. To mitigate these attacks, we postulated the intuitive trust management goals needed to support content validation, in both CCNx and NDN routers, and presented a simple rule (IKB) that makes it practical. This rule is compatible with the tenets of both architectures. We also suggested several optimization techniques for the proposed rule. Given computational overhead of signature verification, we proposed a lightweight probabilistic content ranking algorithm. It ranks content based on consumer acceptance. In case of multiple cached content objects matching an interest, the highest ranked one is returned. Experimental results supported our assertion that this ranking algorithm detects and mitigates content poisoning attacks. The two complementary approaches described in Chapter 4 are the first practical countermeasure to mitigate content poising attacks in CCNx and NDN. We note that both IKB rule and catalogs are implemented as part of the current CCNx standardization draft [142].

Chapter 5 studied the effect of in-network caching on accounting in CCN. Since content can be served by any router, producers do not have the means to learn about the number of cache hits and content requests. We presented a simple and lightweight network-layer accounting technique and showed how to securely extend it to the application-layer. We assessed performance of this technique and demonstrated that accounting in CCN is both possible and practical.

In Chapter 6, we argued that secure fragmentation is an important issue in CCN. It is complicated by the rule that each content object must be signed by its producer. Thus far, fragmentation of content objects has been considered incompatible with CCN since it precludes authentication of individual fragments by routers. We showed that secure and efficient content fragmentation is both possible and advantageous in CCN and similar architectures that involve signed content. We demonstrated a concrete technique (FIGOA) that facilitates efficient and secure content fragmentation in CCN, discussed its security features,

and assessed its performance. We also described a prototype implementation and presented some preliminary results.

Finally, we provided a comprehensive analysis of network-layer NACKs in both NDN and CCNx. NACKs are an important feature of both architectures and their adoption has been debated. As shown in Chapter 7, NACKs can be beneficial in mitigating the impact of Interest Flooding attacks. Despite their benefits, we also showed that NACKs have certain challenging security implications. We identified two types of NACKs (cNACKs and fNACKs) and explored their security requirements. We then described how secure cNACKs can trigger producer-focused flooding attacks and discussed some potential countermeasure.

Many security and privacy issues in both CCNx and NDN must still be addressed before either architecture can be considered as a replacement for IP. Even if implemented as an overlay on top of IP, all CCN security issues should be resolved *a priori*. Some issues studied in this dissertation deserve more attention. Our work in Chapter 4 can be extended to take into account an on-path adversary (e.g., a router) injecting fake content as a response for all or some received interests. In Chapter 7, we proposed a countermeasure to mitigate producer-focused DoS attacks stemming from securing cNACKs. This countermeasure requires further investigation in order to assess its validity and suitability. Moreover, Interest Flooding attacks (although not discussed in this dissertation) remain a major threat in CCN. There are no deterministic countermeasures to Interest Flooding attacks.

Bibliography

[1] CCN now supports android. http://blogs.parc.com/blog/2010/11/ccn-now-supports-android/. Accessed: February 12, 2016.

[2] Circuit switching (CS) vs packet switching (PS) networks — difference between circuit switching and packet switching. http://www.rfwireless-world.com/Terminology/circuit-switching-vs-packet-switching.html. Accessed: March 4, 2016.

[3] Content centric networking (CCNx) project. http://www.ccnx.org. Accessed: February 12, 2016.

[4] DFN-Verein. http://www.dfn.de/. Accessed: February 18, 2016.

[5] DFN-Verein: DFN-NOC. http://www.dfn.de/dienstleistungen/dfninternet/noc/. Accessed: February 18, 2016.

[6] DNSCurve: Usable security for DNS. https://dnscurve.org/. Accessed: March 22, 2016.

[7] IPv6 extension headers review and considerations. https://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html. Accessed: May 26, 2016.

[8] IRCache project. http://www.ircache.net/. Accessed: July 27, 2012.

[9] Kernel virtual machine (KVM). http://www.linux-kvm.org. Accessed: March 2, 2016.

[10] LBNL's Network Research Group arpwatch, the ethernet monitor program; for keeping track of Ethernet/IP address pairings. ftp://ftp.ee.lbl.gov/arpwatch.tar.gz. Accessed: March 18, 2016.

[11] NACKs in ndnSIM2.0? http://www.lists.cs.ucla.edu/pipermail/ndnsim/2015-February/thread.html. Accessed: March 9, 2016.

[12] The national laboratory for advanced network research project. http://www.caida.org/projects/nlanr/. Accessed: February 12, 2016.

[13] NDN packet format specification. http://named-data.net/doc/ndn-tlv/. Accessed: January 30, 2016.

[14] NDN testbed. http://named-data.net/ndn-testbed/. Accessed: February 12, 2016.

[15] Network simulator 3 (NS-3). http://www.nsnam.org/. Accessed: February 18, 2016.

[16] NSF future internet architecture project. http://www.nets-fia.net/. Accessed: January 25, 2016.

[17] Sandvine global internet phenomena report. https://www.sandvine.com/trends/global-internet-phenomena/. Accessed: February 18, 2016.

[18] Snort project, the snort: The open source network intrusion detection system. http://www.snort.org. Accessed: March 18, 2016.

[19] Secure hash standard. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, 2002. Accessed: March 1, 2016.

[20] B. Adamson and C. Bormann. RFC 5401: Multicast negative-acknowledgment (NACK) building blocks, 2008.

[21] B. Adamson, C. Bormann, M. Handley, and J. Macker. RFC 5740: NACK-oriented reliable multicast (norm) transport protocol, 2009.

[22] 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.

[23] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. Technical report, University of California, Los Angeles, 2012.

[24] 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.

[25] 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.

[26] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE*, 50(7), 2012.

[27] 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.

[28] 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.

[29] 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.

[30] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review*, 34(1), 2004.

[31] 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.

[32] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4033: DNS security introduction and requirements, 2005.

[33] 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.

[34] N. Asokan, K. Kostiainen, P. Ginzboorg, J. Ott, and C. Luo. Towards securing disruption-tolerant networking. Technical report, NRC-TR-2007-007, Nokia Research Center, 2007.

[35] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte. Min-max heaps and generalized priority queues. *Communications of the ACM*, 29(10), 1986.

[36] L. D. Baron. Preventing attacks on a users history through CSS: Visited selectors. `http://dbaron.org/mozilla/visited-privacy`, 2010. Accessed: March 16, 2016.

[37] R. Bassil, R. Hobeica, W. Itani, C. Ghali, A. Kayssi, and A. Chehab. Security analysis and solution for thwarting cache poisoning attacks in the domain name system. In *19th International Conference on Telecommunications (ICT)*, 2012.

[38] M. Baugher, B. Davie, A. Narayanan, and D. Oran. Self-verifying names for read-only named data. In *INFOCOM Workshops*, 2012.

[39] S. M. Bellovin. Problem areas for the ip security protocols. In *USENIX Security*, 1996.

[40] S. M. Bellovin, M. Leech, and T. Taylor. ICMP traceback messages. *IETF Draft, AT&T Labs Research*, 2003.

[41] D. J. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In *Progress in Cryptology (LATINCRYPT)*, 2012.

[42] H. Bidgoli. *The internet encyclopedia*, volume 3. John Wiley & Sons, 2004.

[43] R. Birke, M. Mellia, M. Petracca, and D. Rossi. Experiences of VoIP traffic monitoring in a commercial ISP. *International Journal of Network Management*, 20(5), 2010.

[44] A. Biryukov, I. Pustogarov, and R. Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *IEEE Symposium on Security and Privacy (SP)*, 2013.

[45] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[46] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, 2007.

[47] R. Braden. RF 1122: Requirements for internet hosts-communication layers, 1989.

[48] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4), 2004.

[49] J. Calvert. The electromagnetic telegraph. `http://mysite.du.edu/~jcalvert/tel/telhom.htm`, 2008.

[50] CCN-Lite. `http://ccn-lite.net/`.

[51] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. RFC 4838: Delay-tolerant networking architecture, 2007.

[52] J.-H. Cho, A. Swami, and I.-R. Chen. A survey on trust management for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials*, 13(4), 2011.

[53] 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.

[54] 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.

[55] 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.

[56] W. Conner, A. Iyengar, T. Mikalsen, I. Rouvellou, and K. Nahrstedt. A trust management framework for service-oriented environments. In *Proceedings of the 18th international conference on World wide web*, 2009.

[57] 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.

[58] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Transactions on Information and System Security (TISSEC)*, 12(3), 2009.

[59] 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.

[60] 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.

[61] D. Dean and A. Stubblefield. Using client puzzles to protect tls. In *USENIX Security Symposium*, 2001.

[62] S. E. Deering. RFC 2460: Internet protocol, version 6 (IPv6) specification, 1998.

[63] J. P. Degabriele and K. G. Paterson. Attacking the ipsec standards in encryption-only configurations. In *IEEE Symposium on Security and Privacy (SP)*, 2007.

[64] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic. Pollution attacks and defenses for internet caching systems. *Computer Networks*, 52(5), 2008.

[65] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. ANDaNA: Anonymous named data networking application. In *Symposium of Network and Distributed System Security (NDSS)*, 2011.

[66] T. Everts. The average web page has almost doubled in size since 2010. `http://www.webperformancetoday.com/2013/06/05/web-page-growth-2010-2013/`. Accessed: February 18, 2016.

[67] S. Farrell, E. Davies, and D. Kutscher. The NetInf protocol. *IRTF Draft, Trinity College Dublin*, 2013.

[68] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable ICN. *ACM SIGCOMM Computer Communication Review*, 43(4), 2013.

[69] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security*, 2000.

[70] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol–HTTP/1.1, 1999.

[71] R. Focardi, R. Gorrieri, R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini, and E. Tronci. Formal models of timing attacks on web privacy. *ENTCS*, 62, 2002.

[72] S. Frankel and S. Krishnan. RFC 6071: Ip security (ipsec) and internet key exchange (ike) document roadmap, 2011.

[73] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, 2000.

[74] 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.

[75] R. Garner. Forerunner in wireless telegraphy. *Journal of the Institution of Electrical Engineers*, 4(48), 1958.

[76] J. Garrett, J. Hagan, and J. Wong. RFC 1433: Directed ARP, 1993.

[77] 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.

[78] C. Ghali, G. Tsudik, E. Uzun, and C. A. Wood. Living in a PIT-less world: A case against stateful forwarding in content-centric networking. *arXiv preprint arXiv:1512.07755*, 2015.

[79] 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.

[80] 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.

[81] M. E. Gorman. Alexander graham bell. *Encyclopedia of Creativity, Two-Volume Set*, 1999.

[82] R. Gorrieri, R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini, and E. Tronci. Automated analysis of timed security: A aase study on web privacy. *International Journal of Information Security*, 2(3), 2004.

[83] M. Götz, A. Machanavajjhala, G. Wang, X. Xiao, and J. Gehrke. Publishing search logsa comparative study of privacy guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 24(3), 2012.

[84] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, F. Schamel, and M. Franke. Service-centric networking. *Handbook of Research on Redesigning the Future of Internet Architectures*, 2015.

[85] T. Hain. RFC 2993: Architectural implications of NAT, 2000.

[86] 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.

[87] S. Hares, Y. Rekhter, and T. Li. RFC 4271: A border gateway protocol 4 (BGP-4), 2006.

[88] H.-C. Hsiao, T. H.-J. Kim, S. Yoo, X. Zhang, S. B. Lee, V. Gligor, and A. Perrig. STRIDE: sanctuary trail–refuge from internet DDoS entrapment. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013.

[89] E. Ichihara, K. Kikuchi, T. Tsuchida, K. Kawazoe, and H. Kazama. Reliable IP-multicast protocol. In *21st International Communications Satellite Systems Conference and Exhibit*, 2003.

[90] M. Impett, M. S. Corson, and V. Park. A receiver-oriented approach to reliable broadcast in ad hoc networks. In *IEEE Wireless Communications and Networking Confernce*, volume 1, 2000.

[91] C. V. N. Index. Forecast and methodology, 2014-2019 white paper. Technical report, Cisco, 2015.

[92] V. Jacobson, J. Burke, D. Estrin, L. Zhang, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, et al. Named data networking (NDN) project 2012-2013 annual report. `http://named-data.net/wp-content/uploads/2013/10/ndn-annualreport2012-2013.pdf`, 2014. Accessed: February 18, 2016.

[93] 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.

[94] M. Jakobsson and S. Stamm. Web camouflage: Protecting your clients from browser-sniffing attacks. *IEEE Security & Privacy*, 5(6), 2007.

[95] A. Juels and J. G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Symposium Network and Distributed System Security (NDSS)*, 1999.

[96] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols*. CRC press, 2007.

[97] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. RFC 7296: Internet key exchange protocol version 2 (IKEv2), 2014.

[98] C. A. Kent and J. C. Mogul. *Fragmentation considered harmful*. Digital Equipment Corporation Western Research Laboratory [WRL], 1987.

[99] S. Kent. RFC 4302: IP authentication header, 2005.

[100] S. Kent. RFC 4303: IP encapsulating security payload (ESP), 2005.

[101] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe. RFC 3947: Negotiation of NAT-Traversal in the IKE, 2005.

[102] A. Klein. BIND 8 DNS cache poisoning. `http://packetstorm.foofus.com/papers/attack/BIND_8_DNS_Cache_Poisoning.pdf`, 2007. Accessed: March 18, 2016.

[103] F. Kocak, G. Kesidis, T.-M. Pham, and S. Fdida. The effect of caching on a model of content and access provider revenues in information-centric networks. In *International Conference on Social Computing (SocialCom)*, 2013.

[104] 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.

[105] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Advances in Cryptology (CRYPTO)*, 2001.

[106] H. Krawczyk, R. Canetti, and M. Bellare. RFC 2104: HMAC: Keyed-hashing for message authentication, 1997.

[107] J. F. Kurose. *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E*. Pearson Education India, 2005.

[108] D. Lagutin. Redesigning internet-the packet level authentication architecture. *Licentiates Thesis-Helsinki University of Technology*, 2008.

[109] K. Lahey. RFC 2923: TCP problems with path MTU discovery, 2000.

[110] T. Lauinger. *Security & scalability of content-centric networking*. PhD thesis, TU Darmstadt, 2010.

[111] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[112] K. Leung and Y. Lee. RFC 7337: Content distribution network interconnection (CDNI) requirements, 2012.

[113] W. Li, A. Joshi, and T. Finin. Coping with node misbehaviors in ad hoc networks: A multi-dimensional trust management approach. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, 2010.

[114] S. Lin, D. Costello, and M. Miller. Automatic-repeat-request error-control schemes. *Communications Magazine, IEEE*, 22(12), 1984.

[115] H. Liu, W. Zhang, S. Yu, and J. Cai. A client-driven scalable cross-layer retransmission scheme for 3G video streaming. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2005.

[116] 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.

[117] 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.

[118] 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.

[119] B. Lloyd, D. Carr, G. McGregor, and K. Sklower. RFC 1990: The PPP multilink protocol (mp), 1994.

[120] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006.

[121] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Communications of the ACM*, 52(11), 2009.

[122] J. Lopez, R. Roman, I. Agudo, and C. Fernandez-Gago. Trust management systems for wireless sensor networks: Best practices. *Computer Communications*, 33(9), 2010.

[123] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *24th International Conference on Data Engineering (ICDE)*, 2008.

[124] 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.

[125] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3), 2002.

[126] A. Mason. *CCSP Self-Study: Cisco Secure Virtual Private Networks (CSVPN)*. Pearson Higher Education, 2004.

[127] 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.

[128] M. Mathis and J. Heffner. RFC 4821: Packetization layer path MTU discovery, 2007.

[129] D. Maughan and M. Schneider. RFC 2408: Internet security association and key management protocol (ISAKMP), 1998.

[130] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. *ACM SIGOPS Operating Systems Review*, 33(5), 1999.

[131] J. McCann, J. Mogul, and S. E. Deering. RFC 1981: Path MTU discovery for IP version 6, 1996.

[132] R. McKinney, W. A. Montgomery, H. Ouibrahim, P. Sijben, and J. J. Stanaway. Service-centric networks. *Bell Labs Technical Journal*, 3(3), 1998.

[133] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[134] R. C. Merkle. Method of providing digital signatures, 1982. US Patent 4,309,569.

[135] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology (CRYPTO)*, 1987.

[136] D. Mills, J. Martin, J. Burbank, and W. Kasch. RFC 5905: Network time protocol version 4: Protocol and algorithms specification, 2010.

[137] P. Mockapetris. RFC 1035: Domain names - implementation and specification, 1987.

[138] P. Mockapetris. RFC 1035: Domain names: implementation and specification, 1987.

[139] 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.

[140] 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.

[141] M. Mosko and I. Solis. CCNx messages in tlv format. *IRTF Draft, Palo Alto Research Center, Inc*, 2016.

[142] M. Mosko, I. Solis, and C. Wood. CCNx semantics. *IRTF Draft, Palo Alto Research Center, Inc*, 2016.

[143] S. Mukherjee, A. Baid, I. Seskar, and D. Raychaudhuri. Network-assisted multihoming in the mobilityfirst future internet architecture. Technical report, Rutgers University, 2013.

[144] S. Y. Nam, D. Kim, and J. Kim. Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks. *IEEE Communications Letters*, 14(2), 2010.

[145] 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.

[146] D. Naylor, M. K. Mukerjee, P. Agyapong, R. Grandl, R. Kang, M. Machado, S. Brown, C. Doucette, H.-C. Hsiao, D. Han, et al. XIA: architecting a more trustworthy and evolvable internet. *ACM SIGCOMM Computer Communication Review*, 44(3), 2014.

[147] 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.

[148] B. Niven-Jenkins, F. L. Faucheur, and N. Bitar. RFC 6707: Content distribution network interconnection (CDNI) problem statement, 2012.

[149] 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.

[150] B. Nugraha, R. Khondoker, R. Marx, and K. Bayarou. A mutual key agreement protocol to mitigate replaying attack in expressive internet architecture (XIA). In *Proceedings of the ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?*, 2014.

[151] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3), 2010.

[152] K. Obraczka. Multicast transport protocols: a survey and taxonomy. *IEEE Communications Magazine*, 36(1), 1998.

[153] M. Omar, Y. Challal, and A. Bouabdallah. Certification-based trust models in mobile ad hoc networks: A survey and taxonomy. *Journal of Network and Computer Applications*, 35(1), 2012.

[154] C. Partridge. Authentication for fragments. In *Proceedings of the ACM SIGCOMM HotNets-IV workshop*, 2005.

[155] R. Patané and J. Remond. Economics of information-centric networks. *Internet Economics VIII*, 2014.

[156] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee. WSEC DNS: Protecting recursive DNS resolvers from poisoning attacks. In *IEEE/IFIP International Conference on Dependable Systems & Networks*, 2009.

[157] 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.

[158] L. Peterson, B. Davie, and R. van Brandenburg. RFC 7336: Framework for content distribution network interconnection (CDNI), 2014.

[159] T.-M. Pham, S. Fdida, and P. Antoniadis. Pricing in information-centric network interconnection. In *IFIP Networking Conference*, 2013.

[160] S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *ACM SIGMETRICS Performance Evaluation Review*, 22(1), 1994.

[161] D. Plummer. RFC 826: Ethernet address resolution protocol: Or converting network protocol addresses to 48. bit ethernet address for transmission on ethernet hardware, 1982.

[162] K. Pogran, R. Tomlinson, J. White, and A. Bhushan. RFC 561: Standardizing network mail headers, 1973.

[163] R. L. Popp. Implications of internet fragmentation and transit network authentication. In *Local area network interconnection*. Springer, 1993.

[164] J. Postel. RFC 792: Internet control message protocol, 1981.

[165] J. Postel. RFC 793: Transmission control protocol, 1981.

[166] J. Postel et al. RFC 791: Internet protocol, 1981.

[167] T. Przygienda. RFC 3359: Reserved type, length and value (tlv) codepoints in intermediate system to intermediate system, 2002.

[168] J.-F. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, 2001.

[169] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1), 1998.

[170] 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.

[171] N. M. Sabah and A. Hocanin. The use of negative acknowledgement control packets (NACKs) to improve throughput and delay in IEEE 802.11 networks. In *2nd International Conference on Computer Technology and Development (ICCTD)*, 2010.

[172] H. Sallay and O. Festor. A highly distributed dynamic ip multicast accounting and management framework. In *Integrated Network Management VIII*. Springer, 2003.

[173] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, and N. Blefari-Melazzi. Transport-layer issues in information centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, 2012.

[174] J. Satran, G. Gershinsky, and B. Rochwerger. Nack suppression for multicast protocols in mostly one-way networks, 2004. US Patent 6,807,578.

[175] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking (TON)*, 9(3), 2001.

[176] S. Schinzel. An efficient mitigation method for timing side channels on the web. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011.

[177] K. Seo and S. Kent. RFC 4301: Security architecture for the internet protocol, 2005.

[178] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri. Mobilityfirst future internet architecture project. In *Proceedings of the 7th Asian Internet Engineering Conference*, 2011.

[179] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav. A global name service for a highly mobile internetwork. In *Proceedings of the ACM conference on SIGCOMM*, 2014.

[180] J. Shi and B. Zhang. NDNLP: A link protocol for NDN. Technical report, NDN-0006, 2012.

[181] D. Smetters and V. Jacobson. Securing network content. Technical report, Citeseer, 2009.

[182] 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.

[183] N. Solis, G. Scott, and G. Edens. Ccn 1.0. In *Proceedings of the 1st international conference on Information-centric networking*, 2014.

[184] S. M. Specht and R. B. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *ISCA International Conference on Parallel and Distributed Computing (and Communications) Systems*, 2004.

[185] P. Srisuresh and K. B. Egevang. RFC 3022: Traditional IP network address translator (traditional NAT), 2001.

[186] P. Srisuresh and M. Holdrege. RFC 2663: IP network address translator (NAT) terminology and considerations, 1999.

[187] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Transactions on Networking (TON)*, 11(1), 2003.

[188] H.-M. Sun, W.-H. Chang, S.-Y. Chang, and Y.-H. Lin. DepenDNS: Dependable mechanism against DNS cache poisoning. In *Cryptology and Network Security*, 2009.

[189] P. Syverson, R. Dingledine, and N. Mathewson. Tor: the second-generation onion router. In *Usenix Security*, 2004.

[190] A. Tanenbaum and W. David. *Computer Networks, 5th Edition*. Prentice Hall, 2010.

[191] Z. Trabelsi and W. El-Hajj. Preventing ARP attacks using a fuzzy-based stateful ARP cache. In *IEEE International Conference on Communications*, 2007.

[192] G. Tsudik. Datagram authentication in internet gateways: implications of fragmentation and dynamic routing. *IEEE Journal on Selected Areas in Communications*, 7(4), 1989.

[193] A. K. Venkataiahgari, J. W. Atwood, and M. Debbabi. Secure e-commerce transactions for multicast services. In *The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services*, 2006.

[194] 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.

[195] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin, and D. Raychaudhuri. Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet. In *IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[196] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF based routing protocol for named data networking. Technical report, University of Memphis and University of Arizona, 2012.

[197] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *IEEE Symposium on Security and Privacy (SP)*, 2011.

[198] J. White. RFC 524: Proposed mail protocol, 1973.

[199] Z. Wilcox-OHearn. Names: Decentralized, secure, human-meaningful: Choose two, 2003.

[200] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16(2), 2014.

[201] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate ddos flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[202] K. Yamamoto, Y. Sawa, M. Yamamoto, and H. Ikeda. Performance evaluation of ACK-based and NAK-based flow control schemes for reliable multicast. In *Proceedings of TENCON 2000*, volume 1, 2000.

[203] 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.

[204] T. Zahariadis, H. C. Leligou, P. Trakadas, and S. Voliotis. Trust management in wireless sensor networks. *European Transactions on Telecommunications*, 21(4), 2010.

[205] G. Zhang, S. Fischer-Hübner, L. A. Martucci, and S. Ehlert. Revealing the calling history of SIP VoIP systems by timing attacks. In *International Conference on Availability, Reliability and Security*, 2009.

[206] 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.

[207] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *IEEE Symposium on Security and Privacy (SP)*, 2011.

[208] G. Ziemba, P. Traina, and D. Reed. RFC 1858: Security considerations for IP fragment filtering, 1995.

Glossary

acknowledgment A message sent by packet receivers to indicate the successful reception of a packet. 173

address translation tables A method of translating an entire network IP addresses into a single IP address. 5

adversary model A model describing an adversary or a set of adversaries assumed in a study. 53, 229

arpanet The first implementation of packet-switched networking in late 1960s. 3

authentication header An IPsec protocol that guarantees integrity and data origin authentication. 5

cache A temporary storage of data (content) to be use to respond to future requests. 9, 34, 45, 82, 118, 144, 228

cache pollution An attack in CCN where adversaries attempt to manipulate reference locality of caches, causing incorrect decisions by cache eviction strategies. 44, 198

circuit-switched A communication model where a dedicated channel is established between two devices before transmission begins. 3

consumer An end-user that requests content by sending interest messages. 31, 229, 230

content A CCN piece of data addressed using a unique name. 30, 228–230

content distribution A method for delivering content to end-users with high availability and performance. 58, 60

content poisoning An attack in CCN where adversaries inject fake content in the network to be cached by routers and used to satisfy future interests issued by benign consumers. 44, 82, 85

countermeasure An algorithm, a protocol, or a scheme used to prevent (protect against) one or more attacks under a well pre-defined adversary model. 1, 42, 58, 181, 197

datagram An IP packet containing IP addresses of source and destination nodes. 4, 230

encapsulating security payload An IPsec protocol that guarantees integrity and data origin authentication and confidentiality. 5

fake content A content object that contains an invalid signature or a valid signature generated with the wrong key. 44, 82, 84, 229

forwarding interest table The CCN routing table containing name prefixes and outgoing interfaces leading to corresponding producers. 34

fragmentation A process that divides large packets into fragments with sizes smaller than a specific link's Maximum Transmission Unit. 5, 143, 145

hash function A function that compresses an arbitrary length input to a fixed size output. 26, 79, 93, 160, 189

interactive communication A type of communication that requires all involved parties to actively send packets. 58

interest A message sent by CCN consumers to request content. 31, 228–230

interest collapsing A process where CCN routers only forward the first out of many closely timed interests and collapsed the others in the PIT. 35, 42, 126, 142, 179

interest flooding An attack in CCN where adversaries sends a large number of non-sensical interests in order to fill up router PITs. 43, 130, 174, 212

maximum transmission unit The maximum datagram size that can be transferred over a specific link in the network. 5, 229

negative acknowledgment A message sent by packet receivers to indicate the failure of receiving a packet, e.g., packet is considered lost or corrupted. 173

packet-switched A communication model where data is divided into packets that are transmitted over a medium shared among many devices and not only the ones involved in the communication. 3

pending interest table A CCN router's component containing pending interests metadata (including name) and a list of interfaces on which they arrived. 34

principal A network entity that can be a device, content, interface, service, human end-user, or a collection of identifiers. 7, 14, 193

producer An entity that produce (publish) and disseminate content. 31, 229, 230

replay attack A network attacks where adversaries replay legitimate and valid packets in order to gain access to restricted resources or leak private information. 130, 179

router An entity that forward interests and content to/from consumers and producers. 31, 229, 230

segmentation A process where CCN producers split a large content object into smaller individually signed and named pieces called segments. 145, 147, 149

system model A model describing a system (including all its properties and characteristics) under which a study is performed. 51

timing attack A type of side channel attack in which adversaries attempt to compromise a system or reveal some secret information by measuring the time required to perform a specific normal network (or cryptographic) task. 47, 67

transmission control protocol A transport-layer protocol that enables host-to-host communication and provides reliability, connection establishment, and flow and congestion control. 4