UNIVERSITY OF CALIFORNIA
Los Angeles

Named, Secured Data:
A Fundamental Building Block for Secure Networking

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Zhiyi Zhang

2021

ABSTRACT OF THE DISSERTATION

Named, Secured Data:
A Fundamental Building Block for Secure Networking

by

Zhiyi Zhang
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2021
Professor Lixia Zhang, Chair

Network security is one major goal of the Internet. Over time the channel-based security model, represented by Transport Layer Security, has been applied over TCP/IP to secure network communication between hosts. However, with the growth of content delivery applications and networks, recent years have seen the mismatch between what application needs and what is provided by the channel-based security model. A newly proposed architecture, Named Data Networking, provides an alternative to today's security model by securing named data instead of channels. Not relying on the network context, a piece of named, secured data can be forwarded, cached, reused without breaking the security primitives. At the same time, the same concept has been seen in the new trends of application layer projects like Web Packages and Verifiable Credential.

Under this background, there is an urge to understand the difference between the two security models and how to use the new building block – named secured data – to build new security solutions. To answer this question, we start by revisiting the key concepts of network security from the application's perspective. From there, we define and compare

the two security models and after the analysis, we believe the data-centric security model matches the need of applications more. Furthermore, we present a number of NDN based security systems, ranging from DDoS mitigation to providing secure communication in smart home and vehicular networking, to demonstrate how data centric security and named secured data can be applied to address some challenges that are intractable to channel-based security model. Through the design discussion, we confirm the unique advantages of the new security model and also share insightful experiences of building network systems with named secured data.

The dissertation of Zhiyi Zhang is approved.

Alexander Afanasyev

Peter Reiher

Songwu Lu

Leonard Kleinrock

Lixia Zhang, Committee Chair

University of California, Los Angeles

2021

I dedicate my dissertation to Jiusong, my amazing wife, who has offered unwavering support in the past five years of my doctorate journey. She encourages me when I am upset, believes in me when I hesitates, and laughs with me when I make progresses. Her love and sacrificial care of me and Alan, our child, make it possible for me to finish this journey.

I also want to thank my mom and dad, who instilled in me the courage to think big and pursue my dream when I was a child.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I sincerely thank my esteemed advisor Prof. Lixia Zhang for her insightful guidance, patience, and continuous support throughout my Ph.D. study. Her relentless pursuit of better networking systems and her diligent, rigorous, and earnest attitude to scientific research have encouraged me in all the time of my academic research and daily life. I want to thank my colleagues Philipp Moll, Xinyu Ma, Tianxiang Li, Tianyuan Yu, Eric Newberry, Sichen Song, Tyler Liu, Yufeng Zhang at UCLA. I would also appreciate all the support and insightful discussion I received from my collaborators Prof. Beichuan Zhang (University of Arizona), Prof. Lan Wang (University of Memphis), Prof. Alexander Afanasyev (Florida Int'l University), Dr Craig Lee (Aerospace), Prof. Etienne Riviere (UCLouvain), Dr Alberto sonnino (Facebook), Prof. Michal Krol (City University of London), and the members of the NDN team. At last, I would like to gratefully thank my wife Jiusong Tang for supporting and encouraging me in the past eight years.

VITA

2012–2016	Undergraduate majored in Software Engineering, Undergraduate researcher at Database and Information System Lab, Awarded the National Scholarship of China in 2015, Nankai University, Tianjin, China.
2016	Bachelor of Engineering, Outstanding Graduate Student Award, Nankai University, Tianjin, China.
2016–2021	Graduate student researcher at Internet Research Lab, Teaching assistant in CS118, CS35L, CS217, Computer Science Department, UCLA, California, US.
2017	Research and development intern at Alibaba Group, Seattle, Washington, US.
2020	Development intern at Facebook, Menlo Park, California, US.

PUBLICATIONS

Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, Etienne Riviere, *EL PASSO: Efficient and Lightweight Privacy-preserving Single Sign On.* In proceedings of Privacy Enhancing Technologies Symposium (PETS 2021)

Zhiyi Zhang, Tianxiang Li, John Dellaverson, Lixia Zhang, *RapidVFetch: Rapid Downloading of Named Data via Distributed V2V Communication.* In proceedings of IEEE Vehicular Networking Conference (IEEE VNC 2019)

Yanbiao Li, Zhiyi Zhang, Xin Wang, Edward Lu, Dafang Zhang, Lixia Zhang, *A Secure Sign-On Protocol for Smart Homes over Named Data Networking.* IEEE Communications Magazine (IF=11.052), 2019

Craig Lee, Zhiyi Zhang, Yukai Tu, Alex Afanasyev, Lixia Zhang, *Supporting Virtual Organizations Using Attribute-Based Encryption in Named Data Networking.* In proceedings of IEEE International Conference on Collaboration and Internet Computing (IEEE CIC 2018)

Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik Ramani, Alex Afanasyev, and Lixia Zhang, *NAC: Automating Access Control via Named Data.* In proceedings of IEEE Military Communications Conference (IEEE MILCOM, 2018)

Zhiyi Zhang, H Zhang, E Newberry, S Mastorakis, Y Li, A Afanasyev, L Zhang, *An Overview of Security Support in Named Data Networking.* IEEE Communications Magazine (IF=11.052), 2018

H Zhang, Y Li, Zhiyi Zhang, A Afanasyev, Lixia Zhang, *NDN Host Model.* ACM SIGCOMM Computer Communication Review, 2018

Yu, Yingdi, Alexander Afanasyev, Jan Seedorf, Zhiyi Zhang, and Lixia Zhang, *NDN DeLorean: An authentication system for data archives in named data networking.* In proceedings of ACM Conference on Information-Centric Networking (ACM ICN, 2017)

Tianyuan Yu, Zhiyi Zhang, Xinyu Ma, Philipp Moll, Lixia Zhang, *A Pub/Sub API for NDN-Lite with Built-in Security.* Named Data Networking Technical Report 0071, 2021

Li, Yanbiao, Alexander Afanasyev, Junxiao Shi, Haitao Zhang, Zhiyi Zhang, Tianxiang

Li, Edward Lu, Beichuan Zhang, Lan Wang, and Lixia Zhang, *NDN Automatic Prefix Propagation.* Named Data Networking Technical Report 0045, 2018

Zhiyi Zhang, Yingdi Yu, Alex Afanasyev, and Lixia Zhang, *NDN Certificate Management Protocol (NDNCERT).* Named Data Networking Technical Report 0054, 2017

CHAPTER 1

Introduction

1.1 The Development of Network Security

The requirement of network security came along as far back as the ARPANET [Wik21a] where the Morris worm [Orm03] hit more than six thousands computers in 1988. After that, people starts to realize the growing threats in computer networks and the network security becomes a common requirement.

The essential abstraction of the network, as indicated by TCP/IP, centers around host identifiers and consequentially, the model of communication is based on point-to-point channels between two hosts. At the time when TCP/IP was designed back in 1970s, network security is not considered as a big concern yet and thus TCP/IP protocols do not have sufficient countermeasures to network attacks.

Due to the risk of using unprotected TCP/IP, a number of supplementary mechanisms such as IPSec [FK11], SSL [FKK11] and its successor TLS [Res18] have been patched to the network stack. Since TCP/IP is based on the channel established by the host-to-host connection, the security added by these solutions is also bound to channels. That is, the security of data relies on setting up a direct channel to a known host and the security primitives are only applied to data being transferred in a channel.

In 2006, Van Jacobson gave the talk about Content-centric Networking [JST09], in which a new abstraction of networking is proposed – the network centers around the data name, and the communication model becomes to fetch named data interested by the applications.

The idea was later followed by different projects under the umbrella of Information-centric networking (ICN) in Europe and Named Data Networking (NDN) [ZAB14] sponsored by NSF in the U.S.

To work with the new network abstraction, a new network security model called data-centric security was also proposed together with the new network model. The main idea is the use of *named secured data*. Being named, the data can be located and fetched by the name rather than the IP address of the host where the data originates, allowing data being fetched from any host who caches it. Being secured, the security protection of data is self-containing and independent from where the data is fetched or stored in the network. A typical example of a piece of named secured data is composed of a channel-independent data name, the data payload, and a digital signature covering them generated by the data producer. As such, anyone who knows the signer (*i.e.* signer's public key) can authenticate the data regardless no matter if the content is in transmission or cached in some storage systems.

Under this background, we want to understand the difference between the two security models and identify the one that can better fit the application's network security requirements. To answer this question, in this dissertation, we start by revisiting the key concepts of network security. Instead of being general, we specifically to investigate what is exactly needed by applications: secured channels or secured data? From there, we dive deeper and understand (i) what is the fundamental difference between the two security models, (ii) analyze which model fit which type of application scenarios better, and (iii) what are the potential trade offs. Because the problems require a rather lengthy discussion, we first give a brief preview of our findings.

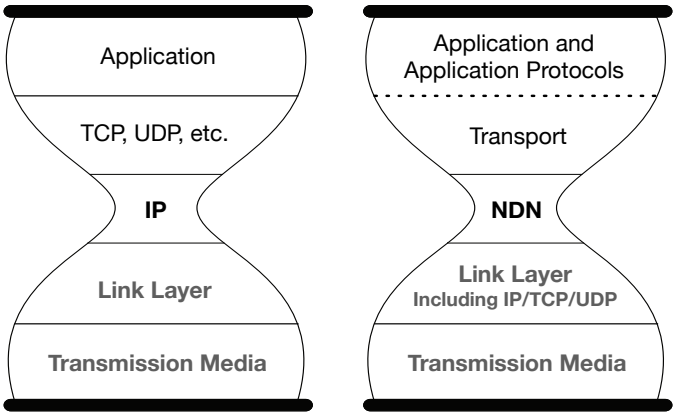


Figure 1.1: TCP/IP and NDN Network Layering Models

1.2 Named Secured Data is What Applications Needed

The ultimate goal of network security solutions is to serve applications. To be more specific, they help applications to obtain and generate data that can meet applications' security requirements, including data integrity, authenticity, confidentiality and so on. The importance of the application can be confirmed by the network layering model (Figure 1.1) where the application layer is on top of the network layer.

Importantly, what applications care most is the data, *i.e.*, getting the right data. However, the channel-centric security addresses the problem of "who", *i.e.*, talking to the right host, and thus there is a mismatch. This mismatch was subtle in the early days of the Internet because when requests are not too many to handle and the latency of content delivery is not much concerned, channel-based security is sufficient to satisfy application's needs as talking to the right host usually leads to getting the right data. For example, if the client can ensure that the a piece of news is downloaded from the ABC News' server (*i.e.*, the right IP address) through TLS, the client can believe the data is truly from ABC News with the security properties provided by TLS.

However, with the growth of content delivery applications (*e.g.*, Youtube, Facebook, Amazon, ABC News), the Content Delivery Network (CDN) services (*e.g.*, Akamai, Cloud-

flare, Amazon CloudFront, Google Cloud CDN) have been widely used to speed up the content delivery by keeping data copies close to the clients (which confirms our early statement that applications care about data), and the mismatch has started to cause unnecessary complexity. For example, when the CDN provider (*e.g.*, Akaimai) is different from the content provider (*e.g.*, ABC News), to fit today's security model like HTTPS/TLS, the CDN provider needs to impersonate the content provider by obtaining certain cryptographic keys that supposed to be known to the latter only.

In contrast, data-centric security matches application's need because it directly secures the data. For example, when using NDN, the client can directly verify the data is released by the content provider (*e.g.*, ABC News) regardless of connections. Therefore, there is no need to force the data directly downloaded from the content provider's IP in real time or to let CDN providers compromise the content provider's key.

The concept of data-centric security has also been observed in new trends in web development, confirming applications' need of securing "what" instead of "where". For example, web packages [Yas21] aim to let clients use content when they are offline and in other situations where there does not exist a direct connection to the server where the content is generated. To ensure the security of the content, signed HTTP exchange [Yas20] has been proposed to make a HTTP file just like a piece of named secured data in NDN, whose security is of independent from connections. However, while a piece of application data can be named, to fetch a piece of secured data in TCP/IP, the name must be translated to IP address where the channel-based security is required. Therefore, there will be an unnecessary overlapping between the data-centric security at the application layer and the channel-based security at the network layer, thus causing additional complexity.

1.3 Contribution

Besides the analysis of two different security models, we present a number of NDN based data-centric security systems that utilize named secured data as the building block. These systems aim to address several real-world challenges, ranging from DDoS mitigation to providing secure communication in smart home and vehicular networking. To be more specific:

- We present Sovereign [ZGM20] to build autonomous smart home systems with NDN over local broadcast media. This is in a sharp comparison with today's best practice where the home control dependents on the external cloud and consequently the user's privacy like daily operations are also disclosed to the cloud. By moving the control to the local network, naming and securing home resources, Sovereign is able to realize autonomous home control and enhance end users' privacy.

- We present FITT [ZVO19] to mitigate the current network flooding attacks by (i) incrementally deploying NDN and FITT overlay at CDN nodes, (ii) letting victims to guide the DDoS mitigation, and (iii) performing fine-grained traffic throttling distributedly near the attackers.

- We present DLedger [ZVM19] as a general logging component that is immutable, high-performance, and distributed. DLedger is built over a peer-to-peer NDN network, a synchronized data structure based on Direct Acyclic Graph (DAG) of blocks with hash, and a set of security conventions to ensure the robustness of the system. Compared with conventional logging systems and blockchain systems, DLedger ensures immutability while removes the single head of line blocking in the single chain systems.

- We present NDN-MPS [], an authentication system that support multiparty trust.

- We present RapidVFetch [ZLD19], an data fetching system to facilitate data downloading in vehicular networking.

Through the design discussions, we demonstrate how data centric security and named secured data can be applied to address some challenges that are intractable to channel-based security model. We confirm the unique advantages of the new security model and also share insightful experiences of building network systems with named secured data.

CHAPTER 2

A Perspective of Network Security

2.1 Defining Network Security from Application's Perspective

To understand the two different models of network security, it is important to first figure out the concept of network security. For this purpose, we collect several existing definitions of network security from different sources.

As defined in ISO/IEC 27033-1:2015 [ISO15]:

> "Network security applies to the security of devices, security of management activities related to the devices, applications/services, and end-users, in addition to security of the information being transferred across the communication links. It is relevant to anyone involved in owning, operating or using a network."

As defined in Wikipedia [Wik21c]:

> "Network security is a set of rules and configurations designed to protect the integrity, confidentiality and accessibility of computer networks and data using both software and hardware technologies."

As defined by SANS Institute [SAN21]:

> Network security is the process of taking preventative measures to protect the underlying networking infrastructure from unauthorized access, misuse, malfunction, modification, destruction or improper disclosure. Implementing these mea-

sures allows computers, users and programs to perform their permitted critical functions within a secure environment.

As shown, existing definitions are related to both hardware and software, and to both network communications and network participants. However, although existing definitions have a wider coverage, the two different security model are more about the *data communication defined by network protocols*. Therefore, we need to focus on and define the network security related to data and the network protocols.

For this purpose, we assume that in a network system, the local hardware platform and the local host environment, which is managed by the operating system, are secure. For example, considering a typical data communication where a node Alice wants to send a file to Bob, with our assumptions, the security holds before the data is sent out through the Alice's network interface, *e.g.*, the data will not be altered or revealed unexpectedly. It is noteworthy that our assumptions do not guarantee that Alice is truly talk with Bob instead of another host, Eve, who impersonates Bob, so it is the network security solutions' duty to help Alice to authenticate the other side of the communication.

Based on the assumptions, we highlight the importance of applications and define the network security from application's perspective. This is because the ultimate goal of network is to serve applications, confirmed by TCP/IP model, the OSI model, NDN model, and many other network system architectures. Considering the network security is to secure the networking process, eventually, the goal of network security is also to serve applications.

From there, to understand the network security, we first need to make a question clear – what is application's need? Is the need to establish a secure channel between network participants or to ensure the data sent to and received from the network is secure? The first answer leads to today's network security solutions which centers around channels while the second answer leads to data-centric security model. We argue the application's need is secure data instead of secure channels. This conclusion can be inferred from several aspects: (i)

applications serve human users and human users care about the data instead of the channels; (ii) establishing secure channels is also to securely send and receive data.

To this end, combining the goal of network security and what application's expectation of secured data, we give the following definition of network security.

Definition 1 (Network Security) *Network security refers to the protection of data transferred in the network to meet application requirement of data availability, integrity, authenticity, confidentiality/access control, and privacy considerations.*

In the rest of this chapter, we will discuss the channel-centric security model and data-centric security model, and analyze their difference.

2.2 Channel-based Security v.s. Data-centric Security

2.2.1 Defining Channel-based Security

Channel-based security model is the currently adopted model in the TCP/IP architecture. For example, the web security is largely based on the HTTPS protocol which secures the channels between clients and web service providers. With such a secured channel, a client can authenticate the server's possession of the claimed domain name, and both sides of the channel can ensure the authenticity and secrecy of the transferred data. Not only being used in web, TLS, the underlying protocol of HTTPS, has also been widely used in many other network systems, including Simple Mail Transfer Protocol (SMTP), Virtual Private Network (VPN), Onion Routing, Secure Shell (SSH), File Transfer Protocol (FTP), etc.

In channel-based security model, whether the data is secure depends on establishing the secure channel (*e.g.*, a TLS connection) with an expected host. Importantly, the security primitives are applied to data only when the data is in the channel. Once a piece data leaves the channel, the security properties of the data no longer exists. For example, after a client Alice download an HTTP file from ABC News, Alice cannot share this file to another client,

Bob, even Bob is interested in the same data. This is because Alice cannot prove to Bob that the file is downloaded from ABC News. Therefore, to get the file in a secure way, Bob needs to establish a channel to ABC News' server and download the same file again.

Based on these characteristics, we define channel-based security model as follows.

Definition 2 (Channel-based Security Model) *Channel-based security model is a security model where the security of data relies on setting up a secure channel to the node where the data originates and the security properties of data exists only in the context of the channel.*

According to this definition, we have the following remark.

Remark 1 *Channel-based security requires a synchronous data exchange.*

When asynchronous data exchange is needed under the channel-based security model, it can cause much undesired complexity in the system, *e.g.*, content providers share their private keys to CDN service providers for the sake of maintaining HTTPS.

2.2.2 Defining Data-centric Security

The second model is data-centric network security model as proposed by CCN/ICN/NDN (hereafter just NDN because NDN is the most developed version so far). The unit being secured is a piece of named data, called a Data packet. A Data packet is composed of a data name, the data payload, and a digital signature covering the name and data payload. The signature is created by the data producer at the time of Data packet generation. Since the security properties of data, *e.g.*, the integrity and authenticity ensured by the signature, the confidentiality provided by payload encryption, are independent of any channels, a Data packet can be cached by any network nodes. Even a Data packet is fetched from cache rather than the original data producer, a data consumer can still verify the data's security properties.

Besides being applied in NDN, the concept of this network security model can also be observed in a number of application layer projects such as JSON web signature (JWS) [JBS15], Verifiable Credentials [SLC19], and Signed HTTP Exchange (SXG) [Yas20] used in Web Packages [Yas21]. Similar to a Data packet in NDN, a JSON with signature, a verifiable credential, or a signed HTTP file can be cached and consumed preserving the security properties without the need to establishing a channel to the original server where they originate.

Based on these characteristics, we define data-centric security model as follows.

Definition 3 (Data-centric Security) *Data-centric security model is a security model where the security properties of data are bound to data itself and independent of the network context.*

Since data security becomes independent of direct and synchronous channels between hosts, we have the following remark.

Remark 2 *Data-centric security allows both synchronous and asynchronous data exchange.*

2.2.3 A Comparison

From the description of the two different security models, the core difference is the dependency on network of the security primitives that have been applied to the data. Under the channel-based security model, data security relies on the channel between the two hosts while in data-centric security model, the data security holds regardless of the network context. For example, a cached TLS packet with MAC and encrypted payload cannot be reused in another TLS channel but a cached NDN Data packet can be reused by other requesters.

The root cause of the difference comes from the different network abstractions used by the two architectures. TCP/IP is about connection between a pair of hosts, so the security model added to it naturally bound to the channel. NDN centers around named data and thus the security model fits it by directly securing named data instead of channels.

As stated, there is a mismatch between the application's needs of secured data and the secured channels provided by the channel-based security model. Such a mismatch was not obvious until the rise of content delivery applications like YouTube, Facebook, etc. and the wide adoption of CDN. This is because before the wide adoption of CDN, clients directly connect to the content provider for data exchange, fitting the channel-based security model. However, when data objects need to be cached in CDN nodes near the content requesters, directly applying channel-based security becomes insufficient because the CDN providers' TLS certificate do not match the content provider's certificate. As a consequent, quick-and-dirty solutions like private key sharing between content providers and CDN providers are being used. In contrast, if the data-centric security can be applied, there is no mismatch and the syste complexity can be much reduced because requesters can check the security of the content regardless whether it is fetched from a CDN provider or the content provider itself. In fact, Web Package mechanism [Yas21] was designed to solve the mismatch at the application layer, which follows the idea of data-centric security model.

Through the comparison, we can see data-centric security model matches application need better, especially for the content delivery applications. If being deployed in the current Internet, the mismatch between content delivery and channel-based security model can be well addressed. Nowadays, more than half of all web traffic have been served over CDNs [Aka21], and that percentage is predicted to grow as applications offer more varied content types. Therefore, if not well addressed, the mismatch may keep causing new complexities and security issues in the future.

It is noteworthy that the comparison does not indicate data-centric security model is always more suitable than channel-based security. A typical use case of channel-based security is the exchange of data that is customized, non-static, and not reusable, *e.g.*, user's bank account information. For such type of data, cache or re-usability is not needed and usually unwanted, and thus a secured channel directly between the client and the server can be a better choice.. Furthermore, in channel-based security model, expensive public key signature

verification is usually used only at the beginning of the connection for identity verification, and the later data is exchanged using lightweight message authentication code (MAC) and symmetric key encryption. In data-centric security, however, digital signature is widely used and thus may affect the performance of data exchange.

2.3 Forward Secrecy and Data-centric Security

Forward secrecy with its typical implementation, TLS, gives an impression that forward secrecy is designed for channel-based security. This is because in TLS, the security is bound to the connection and once the TCP connection changes, a new TLS session or a new set of keying material derived from the previously shared key (PSK) should be used.

However, does this mean the forward secrecy can only exists in channel-based security? To answer the question, we first revisit the definition of forward secrecy [Wik21b].

> "In cryptography, forward secrecy (FS), also known as perfect forward secrecy (PFS), is a feature of specific key agreement protocols that gives assurances that session keys will not be compromised even if long-term secrets used in the session key exchange are compromised."

The core of FS is the relationship between a dedicated key for a session versus a long-term private key kept by the end hosts. This does not restrict whether the dedicated key is bound to a channel identified by a pair of sockets or a group of Interest and Data packets which are independent to channels (there is no concept of a channel in NDN).

Considering a producer and a consumer want to exchange some secret data which is not supposed to be seen by third parties. To ensure forward secrecy, they can perform a Diffie-Hellman (DH) key exchange to obtain a shared secret. Then both of them derive a common key for content encryption, *e.g.*, by applying AES-GCM to the payload of the payload of Data packets. On the one hand, the Data packet carrying the encrypted payload

still promise forward secrecy because even if the long term identity key of either the producer or the receiver get compromised, the attacker cannot recover the shared secret to decrypt the data packet. On the other hand, the security is still data-centric because the security of the Data packet is independent on any underlying channels. For example, the data can be cached or stored by some storage services and be used asynchronously by the consumer.

The use of DH or other means to achieve forward secrecy do pose some limitations to the data packets. For example, the data cannot be reused by multiple consumers and there will be a very low hit rate even the packet is cached. However, in the scenario where forward secrecy is needed, the data being protected is usually not expected to be reused, so the limitation can actually be beneficial to the security of the data.

2.4 Trust as a Central Pillar of Network Security

No matter in channel-based security or data-centric security, a fundamental component is trustworthiness. The current network applications are built over the trust relationship among service providers, network service providers, and end users. This may not be obvious enough to end users because the engineering efforts, *e.g.*, the certificates are pre-installed in users' devices and browsers.

In this work, we define trust relationship as follows.

Definition 4 (Trust relationship) *An agreed upon relationship between two or more system elements that is governed by criteria for secure interaction, behavior, and outcomes relative to the protection of assets.*

Considering security properties, trust indicates both authentication and authorization. To be more specific, if Alice trusts by Bob, it means the Alice already knows Bob's identity information and has means to authenticate Bob's data. A typical implementation is public key cryptography where Alice can install Bob's certificate and verify Bob's data through

signature verification. In addition, the trust is also related to behaviors in certain boundaries. For example, Bob is only trusted to perform certain operations rather than all types of operations. This indicates authorization.

Trust relationship in cyberspace is built based on the trust relationship in the real word and out-of-band operations are usually, if not always, needed to bridge the two. One most common way is to install keys associated to the trusted party, *e.g.* public key certificate, in an out-of-band way.

CHAPTER 3

An Overview of Data-centric Security Support in NDN

3.1 A Brief Introduction to NDN

Named Data Networking (NDN) [ZAB14] changes the Internet communication model from TCP/IP's pushing packets to destination addresses to fetching data by names.

To be more specific, in NDN, an application fetches a piece of data by requesting data name. Such a request is called an *Interest packet* and the fetched data is a *Data packet*. As shown in Figure 3.1, both Interest and Data do not contain any IP addresses or other network topological identifier.

After receiving an Interest packet from the network, as shown in Figure 3.2, a router forwards the Interest by looking up the Interest name and performing long-prefix match against the forwarding information base (FIB), which is like the FIB in IP but is made up with name prefixes instead of IP prefixes. For each forwarded Interest packet, the router will set up a record of the Interest and its forwarding path and keep it in a data structured called Pending Interest Table (PIT). After the Interest packet arrives at a network node or end host that has the matched Data packet, the Data packet will be replied. To forward the Data packet back to the Interest sender, each router will look up the PIT, find the corresponding Interest record, and send the Data packet to the interface from which the Interest came. In this way, the Data packet will following the path of Interest reversely back to the application. At the same time, routers can cache the Data packets so as to satisfy future Interest packets asking for the same piece of data.

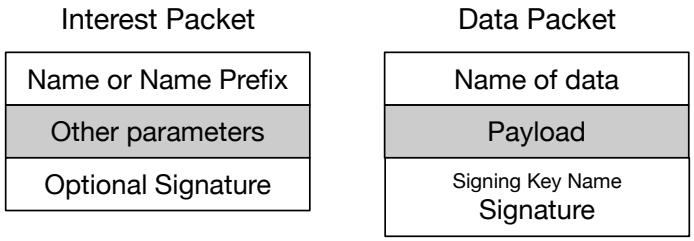
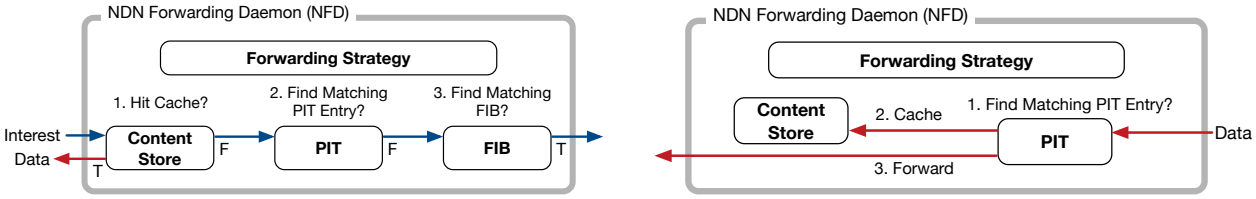


Figure 3.1: Interest and Data packets in NDN



(a) NDN Interest Forwarding (b) NDN Data Forwarding

Figure 3.2: NDN Packet Forwarding

Therefore, name is the network identifier used in NDN. However, different from TCP/IP's design, where each layer defines its own identifier space (*e.g.*, network layer uses IP while transport layer uses port number), data names in NDN are semantic meaningful, hierarchical, and most importantly, defined by applications.

3.2 Security Support of NDN

NDN as a network layer protocol itself does not provide data security. Instead, NDN builds security into its design by allowing applications to add security to their data in a easier way, for example, by cryptographically signing the packets and the signature is carried by individual network packets. In addition, cryptographic schemes and system solutions can be applied over NDN to provide data confidentiality with access control and enhance privacy.

The security support in NDN security is realized in a cross-layer manner: NDN provides a general format to carry the security properties with data while applications provides keying material and security policies.

However, NDN names are defined by applications and the keys of entities also contain application semantics.

3.2.1 Data Availability

NDN provides availability naturally by using named secured data as the basic unit. Specifically, since each Data packet is secured directly, a Data packet can be cached or stored by any other node, and at the same time, does not affect the data consumption, thus improving the availability of data. For example, considering Data packets that carries static content, even if the original producer is offline, data can still be fetched by its name if the Data packets have been cached or stored by routers or dedicated storage services.

3.2.2 Authentication and Data Authenticity

In NDN, a Data packet is not only named but also secured and the authentication of data is built over the public key cryptography. To be more specific, at the time of generation, the producer application will cryptographically sign the packet. This allows any party who knows the public key of the producer to verify the integrity and authenticity of the packet. Importantly, this verification is independent from where the packet is fetched from (*i.e.*, from a router's cache or from the producer application).

To realize the above ability, NDN producers require at least a name for generating data under and a pair of public and private key. For this purpose, NDN assumes the distributed name management and Simple Distributed Security Infrastructure (SDSI) [RL96], where the Internet is composed of a number of organizations where each organization manages their own namespace and maintain their own public key infrastructure (PKI). This is in a similar way as today's domain name system in which each authoritative server can manage their own DNS names in the zone.

Under such a model, each NDN entity is supposed to obtain a name, called an identity

name, from the local name manager. Importantly, the identity name will be associated with the entity's public key. To be more specific, the entity will generate or reuse a pair of public and private key, and apply for a public key certificate from the local certificate manager. An NDN certificate, which is also an regular NDN Data packet, binds the identity name and the public key together, and is signed by the certificate manager. As a Data packet, a certificate is named in the following way and can be fetched via Interest packets.

/<identity name>/KEY/<key ID>/<Issuer Info>/<Certificate ID>

The difference between an NDN certificate manager and a commercial TLS certificate authority (CA) is as follows.

- Since an NDN certificate certifies a name and the name is semantically meaningful. An NDN certificate can not only be used for authentication but also can indicate authorization. For example, the owner of the certificate whose identity name is /example.org/manager indicates a higher level of access rights than the owner of the certificate for /example.org/guest. In contrast, a TLS certificate usually only serves the purpose of authentication. This is because a commercial TLS CA is not in any specific application system and lacks the semantics that is required for authorization.

- An NDN certificate manager is managed by the local organization while the TLS CAs assume a global trust model because the CA's trust anchor certificates are globally installed by the Internet users.

An implementation of NDN certificate management is NDNCERT [ZAZ17, ZS20].

It is noteworthy that, though NDN provides approaches to data authentication, the security of data is more than that. Authentication and trust verification ensures the data is truly signed by the claimed private key and the key is authorized to sign such a packet. However, it does not guarantee the correctness of the content; for example, the private key of the data producer can be compromised and is used to sign incorrect content.

3.2.3 Named based Authorization

Since both Data packet and the data producer have a name, constraints on these names can be established to express trust policies for authorization. In NDN, these constraints are called trust schema [YAC15]. To be more specific, a trust schema rule contains (i) the packet name that the rule applies to, (ii) the expected signing key name or name pattern, and (iii) a number of NDN certificates as the trust anchors. If a packet's signing key does not satisfy the rule, or if the signer's certificate cannot be verified against an allowed certificate chain (*e.g.*, each certificate in the chain is allowed by the trust schema), the packet will be rejected even if the signature value itself is valid.

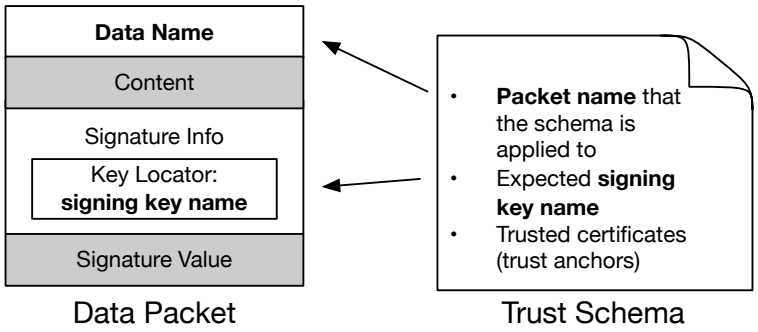


Figure 3.3: Trust Schema Defines the Expected Data Name and Key Name

In this way, besides the signature verification (*i.e.*, authentication), the trust schema further restricts which key can be used to sign which type of data (*i.e.*, authorization). The trust schema rules are supposed to be made following the least privilege principle [SS75]. For example, a temperature sensor is only allowed to sign the temperature data but is not authorized to generate and sign a command to unlock a door. An example of using trust schema in smart home is Sovereign as introduced in Section 4.1.

As a generic component, the trust schema is also used in other security supports. For example, in the data encryption based access control, the trust schema can be used by the access control manager to define which consumer can request which decryption key.

3.2.4 Data Confidentiality and Access Control

In NDN, the confidentiality can be implemented by encryption with access control based on decryption key issuance. Importantly, as stated, public keys carried by certificates are named in NDN and can be fetched as normal NDN Data packets. This is same for encryption and decryption key. The difference is that the decryption key is further encrypted so that only authorized consumers can access to the decryption key. In addition, hybrid encryption is usually used. That is, the payload is encrypted with a symmetric key for efficiency and the symmetric key is encrypted with the asymmetric key which separates the role of encryptor and decryptor (in comparison, symmetric key does not distinguish encryptor and decryptor because they use the same key for decryption and decryption).

In this way, the access control can be reduced to the management of decryption key issuance. The underlying realization varies depending on whether the decryption key issuance is synchronous or not. Here we give two possible approaches as an example.

- If such issuance is synchronous, the decryption key can be requested through a remote procedure call by the consumers. In this process, the access manager can check the trust schema to decide whether such a consumer can sign such a decryption key request.
- If the application needs the key issuance process to be asynchronous for better availability, the key manager can generated encrypted decryption key for all consumers in advance and name these keys following pre-defined naming conventions. Therefore, each consumer can follow naming conventions to fetch the key that belong to the consumer back without contacting the key manager in real time.

An implementation of the asynchronous NDN based access control is Name-based Access Control (NAC) [ZYR18] which supports RSA encryption and attribute-based encryption [BSW07] of NDN Data packets with automatic key delivery with names.

3.2.5 Content and Name Privacy

Privacy poses a higher requirement than applying encryption to the Data content. As defined in ISO 27701 [ISO19], privacy refers to the term that describes the end result of adequate controls over the 'processing' of Personal Identifiable Information (PII) [MGS10]. In the context of attribute-based authentication, privacy is defined through unlinkability, which refers to the need to be able to handle and process personal information anonymously, in a way that precludes being able to identify the original data subjects from the information being communicated and processed. Therefore, only encrypting content does not preserve unlinkability if the name of the packet can be linked or traced.

A further privacy enhancement mechanism is name obfuscation, which aims to minimize the information disclosure from names. From the use of NDN names, three type of information is revealed.

- Name or prefix that can indicate a host.
- Name or prefix that can indicate an application within the host.
- Name that can indicate a piece of content served by the application.

This is similar to the three identifiers used in TCP/IP, namely, IP address, port number, and application-layer identifier. Among the three types of information, the content name is most diverse and information-intensive. In addition, an NDN name prefix used to indicate host or application can also bring more information compared with an IP address or a port number. For example, /west-la-medical-center/emergency-service can reveal more information than "1.2.3.4:80". Even though the IP address can be mapped to the domain name "west-la-medical-center" after a reverse DNS lookup, the /emergency-service is still more meaningful than "80".

Common approaches to name privacy include (i) encryption based mechanisms where the sensitive information in name will be encrypted, (ii) proxy based mechanisms where the

proxy will send the requests like how VPN and Onion Routing work in TCP/IP.

CHAPTER 4

Build Usable Security Systems with NDN

In this section, we introduce several application scenarios where their security challenges can be addressed in a more efficient way by NDN's data-centric security model than the conventional channel-based security. Across very different network environments (*e.g.* home network, vehicular network, Internet) and different application purposes (*e.g.* access control, content delivery, DDoS mitigation), we show that NDN's named secured data provides a better foundation to address some security challenges that are nontrivial with today's network security support.

4.1 Sovereign: NDN Based Smart Home System

The majority of the materials contained in this section were modified or quoted verbatim from [ZGM20].

4.1.1 Background and Motivation

Recent studies and news [TST17, LB16, UJS13, CCJ11, SWA17] see the concern with today's cloud-based smart home systems. In these smart homes, devices are directly controlled by the cloud and the end users are authorized clients of the cloud. Under such a model, users' daily home control commands will go through the cloud to be verified and then executed, leading to undesired exposure of user privacy.

From a network perspective, today's smart home architecture is the underlying TCP/IP

communication model, in which the fundamental way of communication is to connect to an IP address. When multiple end hosts like home devices are involved, the communication model helps to form a solution of introducing a centralized rendezvous node that connects all the hosts together. Because of the big success of the cloud computing, tech giants naturally put this node at the cloud to benefit from the economies of scale and collect valuable information from end users, but the severe privacy issues also arise.

To preserve home users' privacy, we propose a NDN based smart home framework, called Sovereign, to realize user control without relying on external parties like cloud services. The main insight is to utilize a local centralized entity, called a controller, as a unified control point for end users' ease of use. Instead of disturbing direct device-to-device communication, the controller will distribute sufficient policy information and keying materials to individual devices to realize end-to-end security. Devices (and their hosted applications) can directly and securely communicate with NDN through local broadcast media without relying on a rendezvous node.

Importantly, while Sovereign requires more resources at IoT devices, rapid advances in hardware technologies make our design possible. Recent reports [Mar21, IC 15, IC 18a, IC 18b, EE 13] have shown that 32-bit microcontrollers, which provide faster speed and larger memory, have taken the IoT market and promised the strong growth in the next five years.

While moving the control from the cloud to home, our design does not prevent a smart home from utilizing external resources (*e.g.* remote backup, intensive computation), and even sharing data with external parties. Specifically, Sovereign requires these external resources and parties to be authorized by end users. The difference in use of external services between today's smart home systems and Sovereign is that Sovereign let the control fully at end users' hand.

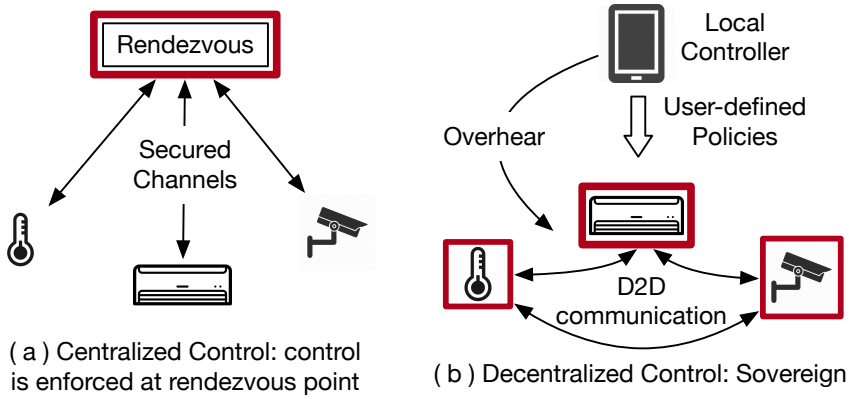


Figure 4.1: Two different models

4.1.2 Design Overview

In today's cloud-based smart home systems, the trust, control and security of the smart home are in the cloud. In comparison, Sovereign changes it in a fundamental way by by bringing the trust anchor to a local node, the Sovereign home controller. Individual devices enforce security according to the security policies by directly securing D2D communication. In addition, Sovereign gets rid of the need of a single rendezvous point by utilizing NDN based D2D communication over the local broadcast network.

Figure 4.2 shows the overview of Sovereign. Consider a home user Alice purchased a new air conditioner (AC) for her home and wants to add it to her Sovereign system. To start with, Alice initiates the device bootstrapping process by some out-of-band operations like QR code scanning or pressing some buttons. In the bootstrapping, after verifying each other, the device will register itself to the controller and the controller will (i) send the system trust anchor to the device, (ii) sends keying material and security policies needed by the device, and (iii) assigns a name to the device according to the naming conventions. In Sovereign, the security policies are based on trust schema rules that are converted from user-decided configurations collected by the controller.

After that, the AC will start running by tracking and adjusting the home temperature, where home temperature data is generated by home temperature sensors. In addition, we

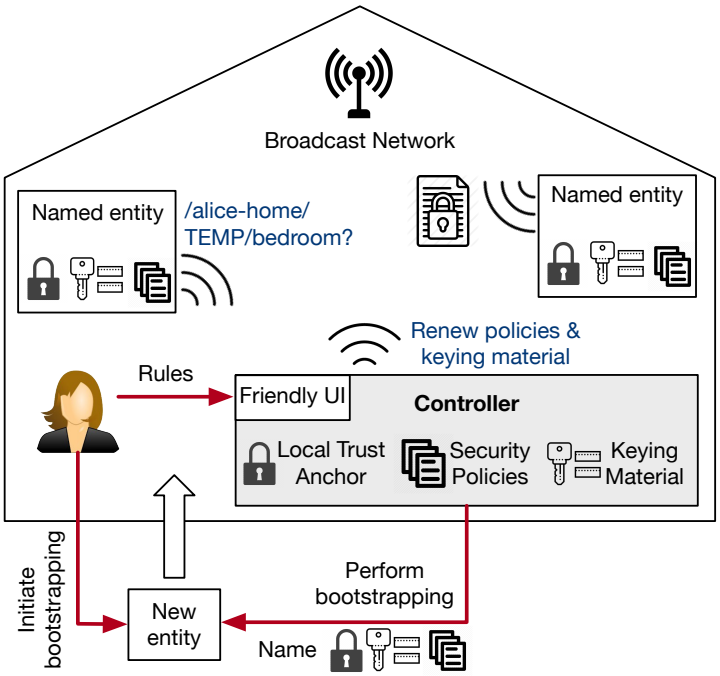


Figure 4.2: An overview of Sovereign

assume the AC can also close bedroom windows. In Sovereign, the temperature data fetching and window command sending all take place directly between the involved devices over the broadcast network in the form of NDN Interest–Data exchange. To get temperature data, the AC directly broadcasts the desired data name so that any other entities who have the data can reply. After fetching the Data packet carrying the temperature, the AC will first verify the packet is signed by another home entity, whose certificate is endorsed by the home trust anchor. Then, to access the payload, the AC needs to first decrypt it using keys obtained from the controller. When issuing a command to the window, the AC will also name this command according to the naming convention, encrypt the payload, and sign it with its own private key. Therefore, windows in the system can fetch the command by the name. If the AC is allowed by the security policies to generate such a command, home windows will successfully verify, decrypt, and execute the command

Type	Naming Convention
Device&Application	/<home-prefix>/<service>/<location>/<entity-id>* *e.g.*/alice-home/AirCon/bedroom/north-ac-1
Commands to executables	/<home-prefix>/<service>/<**scope**>/CMD/<cmd-id>** *e.g.*/alice-home/AirCon/**bedroom/north-ac-1**/CMD/set-temp – device-level command *e.g.*/alice-home/AirCon/**bedroom**/CMD/set-temp – room-level command *e.g.*/alice-home/AirCon/CMD/set-temp – home-level command
Service's Content	/<home-prefix>/<service>/CONTENT/<location>/<entity-id>/<content-id>** *e.g.*/alice-home/TEMP/CONTENT/bedroom/senor-1/temp
Encryption/Decryption Key	/<home-prefix>/<scope>/EKEY /<home-prefix>/<scope>/DKEY
Security Policy	/<home-prefix>/RULE/<location>/<entity-id>**

Notation: A component with <> represent a variable. A component without <> represent a constant string component.

∗: Actual service command, content, and policy NDN Data packets will have a timestamp suffix to achieve the data uniqueness.

∗∗: The corresponding identity key name is the identity name with a "KEY/<key-id>" suffix.

Table 4.1: Naming Conventions in Sovereign

4.1.3 Naming Conventions and Name-based Security Policies

In Sovereign, each piece of data, executable, device, and keying material is named following pre-defined naming conventions as shown in Table 4.1. The naming conventions are pre-installed with the system software and thus, each application knows how to construct an Interest packet to fetch the desired data or to invoke services.

Using the example of the AC in the bedroom of Alice's home, the AC listens to requests under three prefixes as follows so that the AC can react to commands of different levels, namely, device-level, room-level, and house-level commands. As shown, authorized entities can control the home temperature in the desired granularity.

/alice-home/AirCon/**bedroom/north-ac-1**/CMD/set-temp

/alice-home/AirCon/**bedroom**/CMD/set-temp

/alice-home/AirCon/CMD/set-temp

Naming conventions also facilitate the expression of the security policies. Security policies specify entities that are authorized to perform certain operations. In Sovereign, flexible control can be achieved by specifying the names of entities and resources in the policies. Given the name P representing one or multiple entities, and the name R representing one or multiple resources. One can limit P's authorized actions to R by defining a security policy. Such a policy can be written as a triple based on P and R, which can be either be specific names, name prefixes, or regular expressions of names.

<P's name, verb, R's name>.

For example, the verbalized policy "the controller can command all door locks" can be written as the name-based policy <controller name, produce, door lock command prefix>. As another example, the verbalized policy "temperature sensors can produce temperature data" can be represented by the policy <prefix of temperature sensors, produce, temperature content prefix>.

4.1.4 Distributed End-to-end Security Enforcement

The authentication and access control is managed by the controller but enforced distributedly in D2D communication. In general, after converting user-defined rules into name-based security policies, the controller distributes these policies among all entities. During runtime, all Data packets are signed and encrypted by producers, and verified and decrypted by consumers. In the verification step, consumers consult the available security policies and verify whether Data producers are authorized for signing before consuming the content or executing a command.

System Setup Phase: In the system setup phase, the controller generates an asymmetric key pair. The public key is bound to the home prefix and published as a self-signed certificate. This certificate represents the home's trust anchor. The trust anchor will be installed on each entity during the entity bootstrapping. The private key is kept secret by the controller

and is used to sign security policies, cryptographic keying material, and certificates for new entities.

Entity Bootstrapping Phase: In the entity bootstrapping, a new entity joins the system and learns cryptographic keying material and security policies required for later communication. The process takes place in the default broadcast media and starts with the mutual authentication between the controller and the entity, in which out-of-band operations may be needed (*e.g.* QR code scanning). Then, the following information is secretly exchanged:

- The trust anchor certificate is installed on the new entity.
- The controller assigns the new entity an appropriate name according to the naming convention (supported by additional input from the homeowner, *e.g.* entity location).
- The controller issues a public key certificate binding the entity's public key and name together. The certificate is signed with the private key of the trust anchor.
- Security policies and cryptographic keys used for access control are transferred to the new entity.

An implementation of Sovereign's entity bootstrapping process is discussed in [Ano19].

Enforcing Security Policies: While name-based security policies are maintained by the controller, individual participants enforce these security policies for the D2D communication. To elaborate on how these policies are enforced, we differentiate between two types of security policies: (i) *produce*-policies define which entities are allowed to produce data of a specific name pattern (*e.g.* sending an actuating command to a door lock), (ii) *decrypt*-policies define the entities that are allowed to access data of a specific name pattern (*e.g.* data from specific sensors).

Produce-policies are enforced by data receivers. To be more specific, after authenticating a Data packet, the receiving entity extracts the data name and producer's name from

NDN Data's name and signature fields. These names allow checking whether the data was generated by an authorized entity defined in security policies. For example, the following policy defines that temperature content can only be signed by a temperature sensor, where /alice-home/TEMP is the shared prefix of all home temperature sensors, and /alice-home/TEMP/CONTENT is the common prefix of temperature content.

</alice-home/TEMP, produce, /alice-home/TEMP/CONTENT>

Combining the Data's name, the producer's identity, and the available security policies allow receiving entities to reject temperature content produced by unauthorized parties.

The same procedure is applied for restricting entities to issue commands. For example, a *produce*-policy as follows indicates that all the automation applications running on the home hub named "hub-1" can invoke executables whose names match the specified regular expression[1].

</alice-home/AUTO/hub-1, produce, /alice-home/LOCK/<>*/CMD>

Before executing the issued command, the receiving entities first checks the verified producer name against the available security policies and rejects the command when issued by unauthorized entities.

Decrypt-policies are enforced by utilizing Data encryption. To be more specific, every entity is able to fetch Data by emitting an Interest packet carrying the desired content name. The encrypted Data, however, can only be accessed when having access to the correct decryption key. In Sovereign, the controller is maintaining all encryption and decryption keys and allows authorized entities to obtain required keys. This reduces access control to maintaining the access of corresponding decryption keys. For example, the *decrypt*-policy "bedroom AC can read the temperature" is written as follows:

[1]The regular expression <>* matches zero or more name components.

</alice-home/AirCon/bedroom, decrypt, /alice-home/TEMP/DKEY>

The subject name /alice-home/TEMP/DKEY represents the decryption key to the content produced under the temperature service.

Encryption and decryption keys are distributed during the entity bootstrapping process. However, keys can be renewed or changed (*e.g.* for access right revocation purpose), and hence, entities need to securely retrieve the updated keys from the controller. In Sovereign, decryption keys are encrypted for every authorized entity using the shared secret between an entity and the controller. Those secured decryption keys can then be retrieved using Interest-Data exchange. Renewed keys can also be kept in a storage component to make the smart home system resilient against temporary controller failures.

4.1.5 Integrating Sovereign Framework with Pub/Sub

One of the core implementation idea of Sovereign is a publish-subscribe (pub/sub) communication module that encapsulates naming, security, and networking primitives in one API. Pub/sub is a messaging pattern that categorizes messages into semantically meaningful topics and is often seen in the context of IoT. Message producers (called publishers) publish messages to topics without knowing the set of message consumers (called subscribers). Subscribers choose to receive messages under pre-defined topics without the need to know the actual message producers. Pub/sub is used for two main reasons. First, pub/sub is data-centric which matches the data-centric networking and security design of Sovereign. Second, pub/sub has been widely adopted in existing IoT frameworks, and hence, adopting pub/sub provides convenience for developers.

Pub/sub API is directly built over Sovereign's D2D communication as presented in the previous section by handling name prefixes as topic identifiers. That is, subscribers use name prefixes to decide whether a message is under a certain topic or not. For example, a message carrying temperature data of the bedroom is mapped to the name prefix /alice-

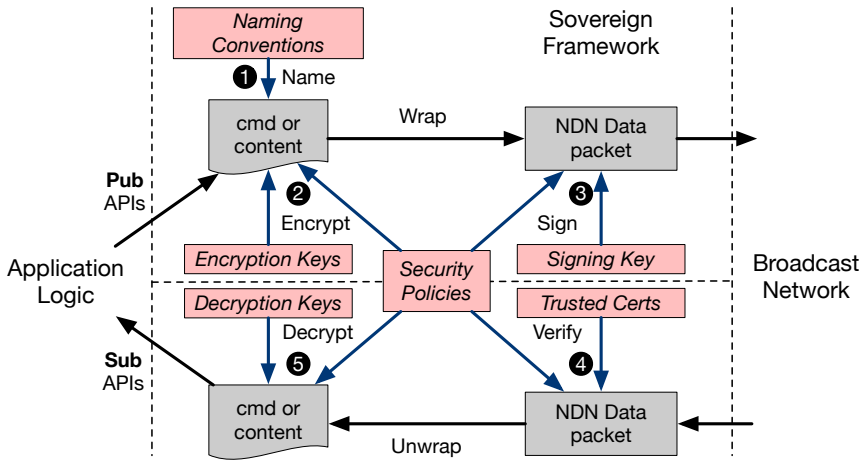


Figure 4.3: Workflow beneath the pub/sub API

home/TEMP/CONTENT/bedroom. This name prefix is further used as the pub/sub topic identifier. In this way, producers that publish content or commands under a topic is to generate Data packets named under the corresponding prefix. Subscribing to a topic is implemented by issuing Interests containing the topic's name prefix to fetch relevant data.

As indicated in Figure 4.3, in Sovereign implementation, the pub/sub API embeds naming, security, and networking considerations to make them transparent to developers. We illustrate the underlying workflow with an example, where an application issues a command to set the bedroom temperature to 70°F. After calling the publish API, Sovereign defines the command name based on application parameters following naming conventions (❶). Sovereign identifies an encryption key according to the topic and encrypts the payload (❷). Further, the API wraps the command name and the encrypted payload into a Data packet. Finally, the API uses the application's private identity key to sign the Data (❸) and makes it available on the local network. On the receiving end, after subscribing to a topic, the API automatically fetches the relevant Data packet. Once a Data packet is received, Sovereign verifies its signature and checks against security policies (❹). In the next step, the Data is decrypted with the corresponding decryption key (❺). Once verified and decrypted, the command is delivered to the application.

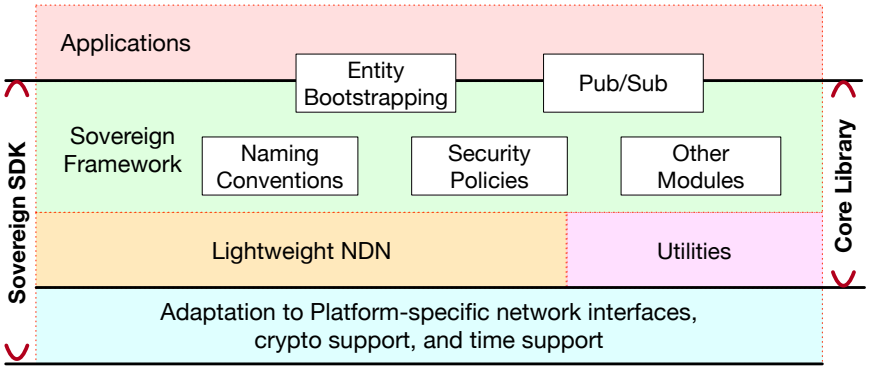


Figure 4.4: Structure of Sovereign's Smart Home SDK

Sovereign is made available as an open-source, cross-platform software development kit (SDK). To allow use over constrained devices, the SDK is built on C and uses static memory allocation only. The SDK's structure is visualized in Figure 4.4. The core library implements the Sovereign framework. As indicated, the only components exposed to developers is the *Entity Bootstrapping* and the *Pub/Sub API*. Other components are transparent for developers. Moreover, the SDK includes an adaptation layer making the core library work across different platforms and communication media. So far, the adaptation layer is tested for platforms including Linux/Unix, RIOT OS [BHG13], and Nordic NRF boards [Nor21]. Regarding connectivity, the adaptation layer allows using Bluetooth, IEEE 802.15.4, and the legacy TCP/UDP used as link layer protocols.

Also, the Sovereign SDK includes a standalone NDN stack that is lightweight enough for constrained devices. This implementation is required since the official NDN library and forwarder — ndn-cxx [Ndn21] and NFD [NFD21] — are not designed for being used on constrained IoT devices. We have verified its compatibility to the official NDN implementation with full examination.

To be used in real smart home products, device vendors and application developers need to program with Sovereign SDK and write the program together with all dependencies to the hardware platform (*i.e.* the program memory on the microcontroller).

4.1.6 Evaluation

We first discuss how Sovereign can work with security and privacy. Thereafter, we evaluate Sovereign's performance and it shows Sovereign's low overhead, low memory and flash requirements, and the ability to work with constrained devices.

Privacy and Security Assessment: Case Study and Comparison We assess Sovereign's privacy and security by analyzing packet flows in two real-world applications. We select two simple yet representative open-source applications [**?**, **?**] taken from the official SmartThings GitHub repositories [**?**], and realize the same functionality in Sovereign. The first application is designed for a smart switch and the second is an automation applet turning on a switch when a contact sensor is touched. The code snippets of the original programs (code block 1 & 3) and their Sovereign-based equivalents (code block 2 & 4) are compared in Figure 4.5. The switch device application changes its own state to "on" when it gets a turn-on command and the automation applets subscribes to the state of a contact sensor and turns on the switch when the contact sensor is touched.

In the SmartThings device application (code block 1), the first line of the code securely connects the device to the home's cloud-backend. After that, the cloud recognizes the new device, learns its profile, and registers it to the device database for the home on the cloud. In contrast, the first line of the Sovereign device application (code block 2) bootstraps the device to a local controller. All sensitive information that is transmitted in the bootstrapping phase stays in the local network and is protected by encryption.

The first line of the automation applet of the SmartThings (code block 3) and the Sovereign application (code block 4) subscribes to a given topic. However, the underlying operations differ: the SmartThings application notifies the cloud backend about its interest in the given service, while the Sovereign application starts listening to data published under the given name prefix in the home network. Similarly, when the SmartThings application turns on the switch, the command message is sent to the cloud backend, where the command

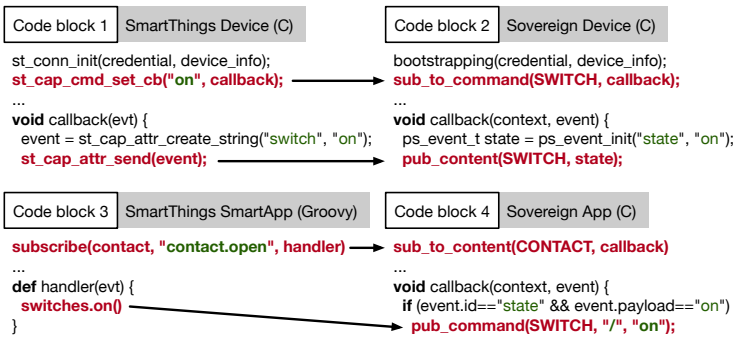


Figure 4.5: Code snippets in SmartThings and Sovereign

is verified and sent back to the switch in the local home. In contrast, publishing a command in Sovereign means producing a new Data packet and making it accessible in the local home network.

Reflecting the above comparison, we see that Sovereign provides the same functionality as cloud-based smart homes via local and secure communication. The home activities and data are not accessible to the cloud unless users explicitly grant access right to a cloud service.s

Latency and Memory Benchmark To evaluate the performance of Sovereign, we measure the latency and memory overhead of the operations *entity bootstrapping*, *content delivery*, and *command delivery*. We conduct our evaluations using a Intel Core i7 laptop, a Raspberry Pi 3B (RPI) and an nRF52840 board [Nor21] that mimic smart home entities with different capabilities. RPIs are widely used in IoT projects and often used for prototyping. Those boards are equipped with an ARM Cortex A53 @1.4GHz processor and 1GB RAM. We classify the evaluated nRF52840 chip with its 32-bit Cortex M4@64MHz CPU, 1MB ROM, and 0.25MB RAM[2] as constrained hardware. The experiments with RPI is over WiFi connectivity and use a laptop equipped with a Core i7 processor as the Sovereign controller. Evaluation involving the nRF52840 is conducted over IEEE 802.15.4 and uses a simplified

[2]We acknowledge that a part of current IoT devices shows lower capabilities. However, market studies [Mar21, IC 15, IC 18a, IC 18b, EE 13] have seen the market turning to 32-bit microcontrollers. Hence, we assume the nRF52840 as an appropriate candidate for future smart home systems.

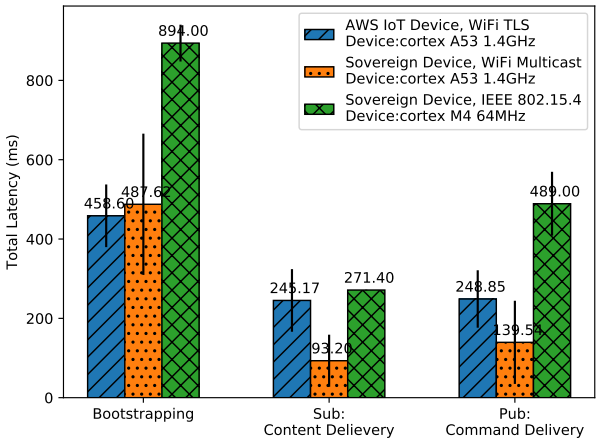


Figure 4.6: User-perceived latency of common operations in Sovereign and Amazon AWS IoT

controller installed on another nRF52840.

Note that in the experiments of content and command delivery in Sovereign, we first used the controller to bootstrap a publisher and subscriber application, respectively, and then brought down the controller. Thus, the communication happens in a D2D manner without a controller involved, showing Sovereign's ability to continue operation even when the controller is temporarily down.

Latency of Common Operations: In the first experiment, we compare the latency of the above-described operations in the Sovereign and the AWS IoT framework. Therefore, we deploy a Sovereign Controller in the local wireless network. For the AWS IoT implementation, the location of the cloud-based AWS IoT controller is not under our control. However, to provide fair conditions, the wireless network is equipped with high-bandwidth Internet connectivity.

The leftmost bar-group in Figure 4.6 visualizes the latency for entity bootstrapping. Here, no significant difference between the RPI implementations of Sovereign and AWS IoT can be observed. The slower nRF52840 chip requires about twice as much time. Here, we want to highlight the use of heavy-weight cryptographic operations, such as key generation,

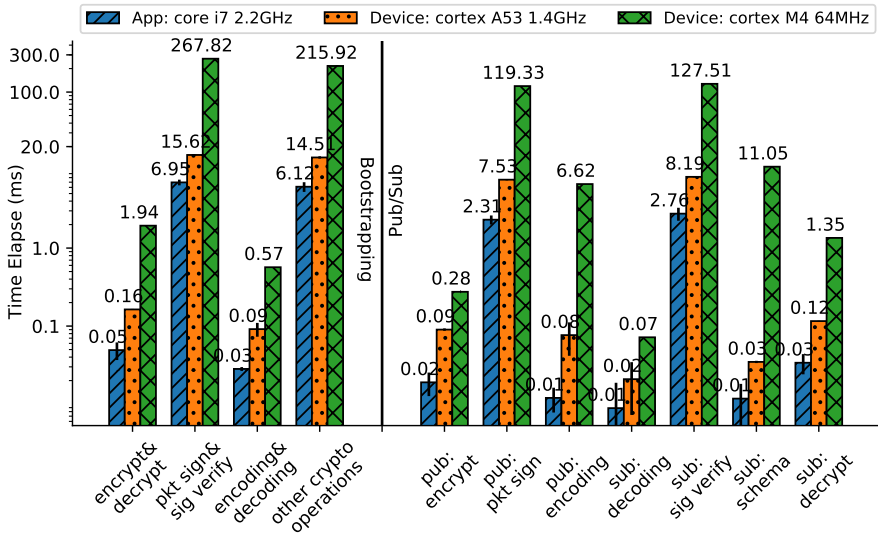


Figure 4.7: Breakdown of the execution time into computational phases in Sovereign devices/applications

that consume a significant amount of time on the constrained hardware.

The advantage of keeping communication local becomes clear when focusing on content and command delivery. Sovereign's RPI implementation is about 62% faster in content delivery and 42% faster in command delivery than the AWS IoT implementation on the same hardware. The constrained nRF52840 chip has about the same latency as AWS IoT for content delivery. For command delivery, the constrained device is slower than the RPI's AWS IoT implementation. However, a latency staying below 500 ms is assumed to be still practical for use in smart homes.

Execution Time Breakdown: A breakdown of Sovereign's runtime into individual operations is provided in Figure 4.7. The visualized operations are performed when preparing Data before broadcasting to the network and Data processing after receiving. This includes digital signature creation and verification (ECDSA), content encryption and decryption (AES CBC), security policy checking, NDN packet encoding/decoding, and other cryptographic operations (including ECDH, KDF). The results show that asymmetric cryptography consumes most of the computation time. This trend is observable across all evaluated devices.

Program/Modules	ROM Use	RAM Use
Subscriber in total	**62KB**	**47.3KB**
Publisher in total	**52.4KB**	**38.2KB**
Application	1.8%	7.3%
High-level Modules	20.7%	34.2%
Utilities	3.3%	14.4%
Crypto Tools	25.1%	0.2%
Network Forwarder	24.1%	25.0%
OS and Adaptation	25.1%	18.9%

Table 4.2: ROM and RAM Consumption

ROM and RAM Footprint: We programmed an nRF52840 chip using RIOT OS [BHG13] for measuring Sovereign's memory footprint. Table 4.2 reports the size taken by individual Sovereign modules. All the main modules together requires less than 50 KB of RAM and 70 KB of ROM. This indicates Sovereign's ability to be used on resource-constrained smart home devices.

4.2 FITT: DDoS Mitigation with NDN

The majority of the materials contained in this section were modified or quoted verbatim from [ZVO19].

4.2.1 Background and Motivation

DDoS attacks have bothered the Internet for decades and often utilize intrinsic properties of the TCP/IP architecture [OSZ20, prn18]. While the TCP/IP networking model has achieved great success, the other side of its design has also been abused by attackers to launch DDoS attacks. The ever-increasing size, frequency, and sophistication of DDoS attacks calls for new approaches that can be partially or fully deployed imminently. We propose that

this urgent need may accelerate our consideration of a new Internet architecture, and that evidence shows that there may now be economic *incentives* for large operators to upgrade their existing infrastructures to embrace it.

Indeed, starting with early DDoS attacks (*e.g.*, attacks from the Trin00 botnet in 1999 [CER99]) through to recent attacks from the Mirai botnet [AAB17], the remediation techniques used in today's Internet suggest that our defensive tactics may not be fundamentally keeping pace with attackers [OSZ20]. Rather, with attacking botnet nodes swelling in size to hundreds of thousands, and even millions, attacks have grown large enough that their attack volume rivals provisioned capacity of DDoS mitigation providers. The Mirai botnet serves as a quintessential example, in that it was used to launch some of the largest DDoS attacks in history, and it did so using compromised devices that primarily included Internet of Things (IoT) devices and household appliances that were both easily discoverable and poorly protected [AAB17]. We note that DDoS has evolved to being *more distributed* than ever, and to increasingly using application-level semantics (e.g. reflective amplification attacks using DNS, NTP, memcached, etc.). On the other hand, service operators, providers, and mitigation services [Aka19, Neu19, Clo19] have had little recourse but to *centralize defenses* and backhaul or black-hole undisrupted attack traffic ("packet love") in large DDoS mitigation service networks. These DDoS mitigation approaches haul offending packets deeper into the network and require an ever increasing amount of deep packet inspection and state in terms of flow semantics to filter out attack packets, thus they do not scale well.

In this work, we examine the basic functions in NDN, that can address the principle weaknesses in today's IP networks and try to mitigate DDoS with properties provided by NDN. Importantly, to allow this being used in real world scenarios, the new approach should be incrementally deployable and align the incentives of existing service providers.

4.2.2 TCP/IP Architecture as the Root Cause of DDoS

Most DDoS attacks that are launched on today's Internet are made possible by utilizing features in the TCP/IP network architecture. Previous work [JWS03, BCR16, YWA05, HG04, Ros14] observed that DDoS attacks often exploit the following specific properties in IP:

- *Push-model Communication*: Any Internet node can send packets to any other IP address. This leaves DDoS attack victims with no way to stop the attack traffic.
- *Destination-based Delivery*: Packet delivery is solely based on the destination address and there is no source address validation by default. Thus, source IP addresses can easily be misattributed, or spoofed, which is a primary feature used by volumetric reflective amplification DDoS attacks [OSZ20].[3]
- *Limited Expressiveness in TCP/IP protocol stack*: IP addresses, even with transport port numbers, cannot expressively describe the semantic characteristics of application-layer traffic, which makes it difficult for DDoS defense mechanisms to inspect traffic to identify attack packets.

Moreover, [HG04] proposed that a DDoS resilience architectural would need: (i) limiting the access to a server based on the server's capabilities, (ii) source address authentication to prevent source address spoofing, (iii) separating client and server address space to prevent unwanted traffic from client to client and server to server, and (iv) building symmetric traffic flows to prevent reflection attacks at the network layer.

4.2.3 DDoS in NDN

In NDN, attackers can only attempt to DDoS a target by flooding it with Interests because attackers cannot actively send Data packets to a target if the target has not send correspond-

[3]This type of DDoS attack has resulted in the largest attacks seen on the Internet to date.

ing Interest packets beforehand. Specifically, inspired by the work [GTU13], we categorize Interest packets used in DDoS attacks into three types according to (i) whether the Interest is valid, and if valid, (ii) whether or not the data requested by the Interest is dynamic (i.e., generated upon the Interest).

Valid Interest for static data (Valid-S) Valid-S Interests fetches Data packets that can be cached, e.g., Data packets for a CSS file or a video chunk. In NDN, Valid-S attacks can be mitigated because multiple Interests asking for the same data can be aggregated and satisfied by the in-network cache. NDN's intrinsic mitigation is sufficient unless attackers flood a huge amount of Interests traffic from a large spectrum of names.

Valid Interests for dynamic data (Valid-D) Valid-D Interests request data that is dynamically generated by producers upon the arrival of the Interest packets, for example, Interest packets used in a remote procedure call. This is usually reflected by a dynamic and unique data name carried by Interest packets. Since the Interest's name is customized and the Data is generated in real time, hardly any Interests arriving at a forwarder would hit cache or an existing Interest with the same name.

Invalid Interests (Invalid) An invalid Interest packet will not fetch Data packet back because of its unrecognized format, unverifiable signature, incorrect application-layer content, etc. This type of Interest is mostly useful to malicious adversaries because legitimate applications can generate correct Interests following certain naming conventions. A possible way to generate such Interests is to append non-existent name components (e.g., randomly-generated garbled bytes) to valid server prefixes.

When Valid-D and Invalid Interests are used in a DDoS attack, since their names are arbitrary and can hardly be satisfied by cache, additional DDoS mitigation services are needed over NDN.

4.2.4 Design of FITT

We demonstrate by a new DDoS mitigation solution over NDN, Fine-grained Interest Traffic Throttling or FITT, that NDN's architectural changes, even when incrementally deployed, can make DDoS attacks fundamentally more difficult to launch and less effective. FITT leverages the NDN design to enable the network to detect DDoS from victim's feedback, throttles DDoS traffic by reverse its exact paths through the network, and enforces control over the misbehaving entities at their sources.

In a nutshell, FITT reacts to feedback from a victim, traces back the specified traffic flows by checking NDN's forwarding state, and enforces traffic throttling at the edge. The design is enabled by two major architectural properties of NDN. (i) The named traffic allows an expressive way for a victim to specify the attacking traffic and for FITT to make fine-grained throttling. (ii) The stateful forwarding provides run time insights into ongoing traffic flows so that FITT can trace the traffic to precisely identify attackers.

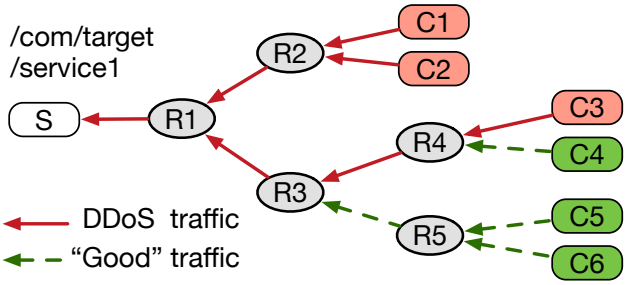


Figure 4.8: An Example Topology

FITT works in three main steps as shown in Figure 4.9. We illustrate the three steps with the example topology in Figure 4.8,

Step 1: Detection When server S's service receives traffic more than the configured threshold, it sends out the feedback to its downstream router $R1$. Once receiving the feedback, $R1$ parses the feedback and triggers the FITT reaction based on the type of the attack.

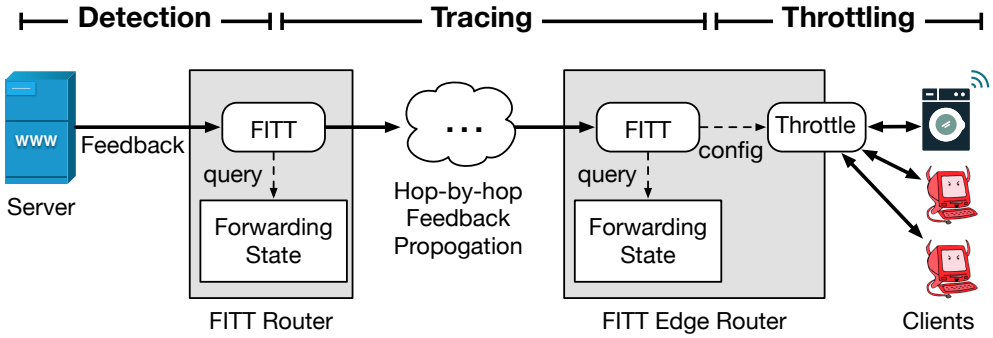


Figure 4.9: An Overview of FITT System

Step 2: Traceback In order to identify attacking sources, $R1$ checks its forwarding state. Using names carried in the feedback from S, $R1$ can identify the attacking traffic flows by their names and then know the downstream routers ($R2$ and $R3$) from which the traffic is from. $R1$ notifies the downstream routers and $R2$ and $R3$ will perform the similar procedures as $R1$ does. In this way, FITT reversely traces the attack traffic from S all the way to *edge routers* $R2$ and $R4$ where exact traffic senders are connected.

Step 3: Throttling The edge routers will first notify these clients and then perform Interest throttling on suspect downstream interfaces within the specific prefix reported by S. During the throttling, an edge router will check whether a client has changed its behavior or not (i.e. whether it lowers down its sending rate to the required value under the specified prefix). The router can then relax or reinforce the limit, accordingly.

Multiple FITT reactions can be triggered at the same time for different traffic prefixes (located on the same server or different servers) and different types of attacks. On the edge, if traffic for a specific prefix from a suspect downstream interface is being throttled by multiple FITT instances, a minimum allowed traffic value will be taken.

Since the mitigation is triggered by victim's feedback, FITT can react immediately after the DDoS attack. The latency for throttling to start is only a one-way trip time (0.5 RTT) from the victim server to clients. In addition, since the whole process entirely operates over

existing NDN forwarding plane, there is no man-in-the-loop for DDoS mitigation with FITT.

4.2.5 Incremental Deployment

We further demonstrate that service providers may implement NDN/FITT on existing CDN nodes as an incrementally deployable solution to effectuate the application level remediation at the sources, which remains unattainable in today's DDoS mitigation approaches.

In this paper, we observe that the suitability of NDN's architecture to perform DDoS remediation is not just a parallel benefit to its suitability to performing CDN functions; rather, we posit that existing CDN deployments are opportune infrastructure to enable broad deployment of NDN. Importantly, many CDNs' existing roles as MaaS providers suggest the potential alignment of costs with incentives to performing both CDN and MaaS.

- First, an upgrade of a CDN to support NDN could provide a synergistic benefit to Internet services that want protection from DDoS, and to the CDN/MaaS provider. Multiple aspects of the synergy that exists between DDoS mitigation and NDN. For example, one of the devastating aspects of volumetric DDoS attacks occurs when attack traffic is backhauled from distributed sources towards destinations (whether the destinations are victim services, or even to MaaS scrubbing centers) [OSZ20]. Deployment of NDN across a CDN/MaaS would let that provider shed attack traffic at the edges before it starts to aggregate across transit links.
- Second, CDN/MaaS providers already shoulder the computation and network requirements that NDN would require. For the TCP/IP Internet, CDN/MaaS providers have already been terminating TLS connections and proxying connections from clients to service infrastructures. An NDN upgrade would fit the operational footprint that large CDNs already have, would not necessitate deployment of additional resources, but would also fundamentally enhance DDoS service offerings.
- Furthermore, among the properties of NDN's information centric architecture is its in-

herent capability to cache data near its consumers, architecturally. This fundamental advantage is poised to not only be a pivotal feature, but also a deployment incentive for large Content Delivery Networks (CDNs) providers. In today's TCP/IP Internet, CDNs are vast networks and deployments that operate as overlay services for end-users. CDNs exist in TCP/IP networking (above the architectural layer) to efficiently deliver content to users today. CDN services typically involve large network infrastructures and deployments. They often perform caching of content to locations that are geographically distributed, as well as geographical load-balancing of requests from clients so that network latency can be minimized. In today's Internet CDNs exist as overlay technologies, and some have proposed that NDN is well suited to implement these functions in the network architecture, itself [CPZ16, JB14, MCC14]. Furthermore, many operational CDNs, today, also perform DDoS mitigation offering commercial DDoS/MaaS services: Akamai [Aka19], Cloudflare [Clo19], and Neustar [Neu19] to name a few.

The resulting NDN network would have global scope and in-network caching in the topologically distributed regions. In general, incremental deployment models become more realistic when they align their costs with incentives. That is, those who deploy new mechanisms are more likely to do so when they anticipate direct benefits from doing so. By contrast, deployments like ingress and egress filtering (BCP-38 [FS00] and BCP-84 [BS04]) illustrate slow adoption, arguably, because those deploying them do not gain any direct benefits. Conversely, service providers already expend resources and money to combat DDoS by either provisioning large amounts of excess bandwidth or by contracting with commercial DDoS mitigation providers [Aka19, Neu19, Clo19]. Service providers whose applications may already be in a position to benefit from migrating to NDN's architecture would gain *additional* benefits by deploying NDN with FITT and thereby being able to shed large amounts of DDoS traffic.

Therefore, we propose an incremental deployment (see Figure 4.10) that leverages CDN to connect NDN routers located at the edges of the network. The two ends speaking NDN

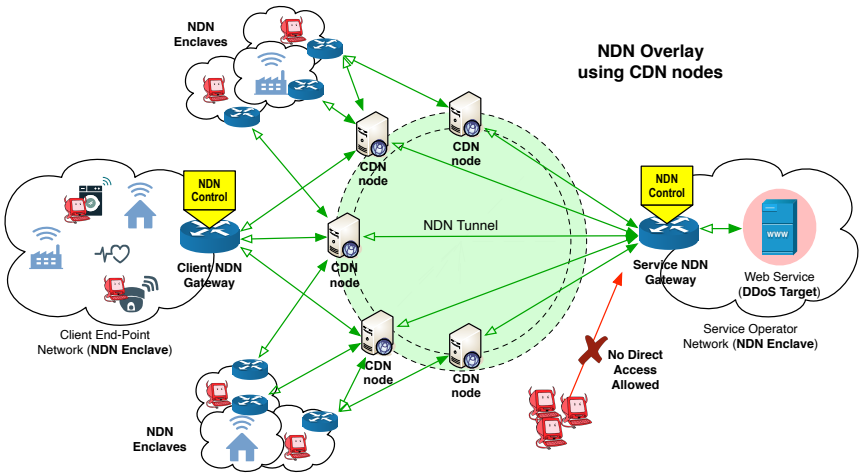


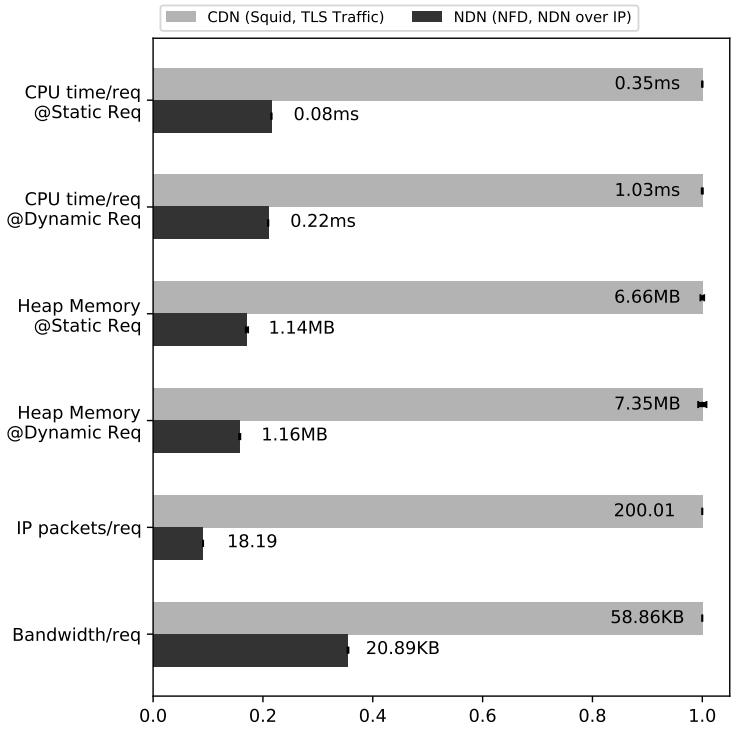
Figure 4.10: FITT and NDN Overlay in CDN

are enough for FITT to effectuate the DDoS mitigation. This means that our approach requires NDN gateways on both sides of the CDN overlay: the client end-points and the service provide/operator side. On the client end-point, the NDN gateway can both forward traffic from existing NDN devices and translate at the application level using an application proxy traffic from traditional TCP/IP devices. Such a translation is akin to having an application proxy (i.e. web proxy) and it would allow the client NDN gateway to perform enforcement without having to worry about packet classification or deep packet inspection. On the service operator side, the NDN gateway can serve traffic directly to Web Service. What's more, service operators who migrate their services to NDN bolster each others' NDN deployments, as those clients independently augment each others' deployments (through facilities like shared caching and shared routing infrastructure). In particular, we observe that serendipitous IoT deployments of NDN, which may already be underway, could benefit other services whose providers have (or will) independently embraced NDN for this reason. That is, an NDN-enabled service may shed DDoS traffic from would-be attack nodes that might otherwise be bots in Mirai. By enabling NDN at the edges in home routers and IoT deployments would place the FITT mitigation machinery very near to some of the Internet's most voluminous DDoS sources for all NDN applications (not just IoT). We believe that it is demonstrably feasible for independent service operators to overcome network protocol

ossification and migrate (at least portions) of their production traffic to NDN. Furthermore, we show later in this section that the FITT/NDN deployment outperforms existing CDN caching schemes because it is performed as a network function deeper into the stack. Thus, by enabling NDN at the edges, we get both performance and security benefits without sacrificing functionality.

The deployment of NDN/FITT may also not need to bother the change of existing end-point applications running in NDN enclaves. To be more specific, the gateway of NDN enclaves can be an NDN forward/reverse proxy. For clients, e.g., IoT devices, to talk to a remote service, the client's gateway server plays the role as a forward proxy which turns application-layer request (e.g., HTTP GET request) into an NDN Interest packet and sends it out to the server over the NDN tunnel; while for the service provider side, the server gateway router serves as a reverse proxy parsing NDN Interest packets back to normal request. When servers send back the response, the process is similar. Given the commonalities between NDN's Interest-Data exchange and today's widely-used request-response model in application layer protocols (e.g., HTTP, RPC), such proxy and reverse proxy are deployable and the cost can be reasonable because it does not require any hardware change – all NDN proxy deployment is software installation in user space.

Performance Comparison of FITT vs CDNs We simulated a topology similar to Figure 4.10 where a CDN infrastructure is used for static and dynamic page delivery for web services. In the pure CDN scenario, the CDN proxies are running the latest version of Squid [**?**] and for our approach the same CDN proxies run a version of our FITT prototype on top of NDN. Our aim was to be able to evaluated the computation and communication overhead of NDN forwarding daemon (NFD), an open-source NDN network forwarder, and Squid, one of the most widely used web proxy systems. Specifically, we performed multiple simulations using today's practice of MaaS and NDN/FITT's DDoS mitigation with the same hardware settings. In our experiments, we used computers equipped with Intel Core



(i) Heap memory consumption for static and dynamic requests (static request will be satisfied by both CDN cache and NDN cache while dynamic request will be forwarded by CDN/NDN proxy).

(ii) CPU consumption for static and dynamic requests.

(iii) Number of packets and traffic received (TLS over traditional TCP/IP versus FITT tunneling NDN packets from an NDN enclave)

Figure 4.11: Comparing a vanilla CDN proxy running Squid and the same proxy running a prototype of FITT on top of NDN

i9 4.6GHz processor with 32GB DDR4 RAM. We also used the same number of clients requesting the same amount of data and computation: six (6) clients requesting for both static content and dynamically-computed content of 2KB at the same rate simultaneously.

As shown in Figure 4.11, we first present the CPU and memory use of a single-thread Squid and NFD/FITT under the same load. The plot indicates that the combination of NFD with FITT has an advantage in terms of resource consumption when compared to Squid: under the same load of traffic, Squid consumes an estimated of about five times (5x)

more memory than FITT. We observed a similar trend when we measured the computation footprint on each of the CDN proxies: FITT requires a mere 20% of the processing time when compared to Squid. In addition, we observed the total amount of traffic at the networking layer (IP) involved assuming a CDN proxy running Squid receiving TLS traffic vs NDN traffic (similar to Figure 4.10. In both cases we made the assumption that traffic is forwarded over a single hop between the CDN and the MaaS scrubbing center. This is the worst case scenario for us because, in practice, MaaS scrubbing centers can be deeper into the network in a centralized location and several network hops away from the edge CDN proxies. Even under that assumption, our experiments indicate that in order to fetch the same amount of content and computation results, Squid over TLS requires around ten times (10x) the amount of packets resulting in an almost three times (2.8x) bandwidth overhead compared with NFD/FITT over IP-overlaid NDN. As we mentioned, since MaaS service requires TLS-terminating traffic forwarding on both the CDN proxies and MaaS scrubbing centers, the computation and communication overhead can be much higher than what we report here for the vanilla CDN implementation making FITT a much more desirable option.

4.2.6 Evaluation

Besides the prototype implementation over the latest stable version of NFD, we also implement FITT in C++ over ndnSIM [**?**], which is a NDN simulation platform based on NS-3. We first demonstrate NDN's DDoS resilience to valid static Interest flooding and then evaluate FITT under different types of attacks. The simulation results show that after the DDoS starts, FITT can effectively control the traffic to the victim as expected within seconds (less than 2 seconds under our simulation settings), and ensure that over 99% of the attack target(s) incoming traffic is from legitimate clients after a short period of time.

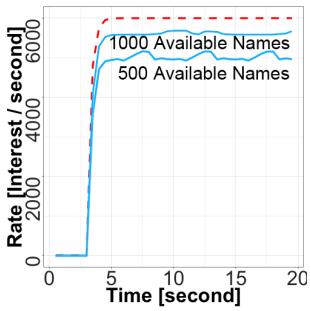
In addition, we simulate the scenarios when (i) multiple DDoS mitigation instances are happening at the same time, (ii) different traffic flows are presented and only one of them is throttled, and (iii) attackers are rogue and follow the DDoS control of FITT. The results

shows FITT can perform fine-grained DDoS mitigation and handle comprehensive attack scenarios.

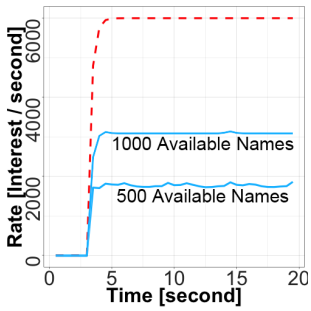
Simulation Topology We simulate the incremental deployment of NDN/FITT using a CDN as an overlay between NDN enclaves. As shown in Figure 4.24, the blue nodes are CDN nodes that are aware of NDN and FITT, the gray nodes are gateway routers of NDN enclaves. Behind each gateway router, we simulate 10 compromised IoT devices and 2 honest IoT devices. Since FITT follows a divide-and-conquer strategy in traffic throttling, the scale of the network topology does not affect evaluation results of FITT much. We make the service globally reachable, which means all users have means to learn the name and express Interest packets towards the service. For sake of simplicity, we use the prefix P to represent the service in rest of the section.

Simulation Result Notation In the simulation result plots, we use the red dashed line to represent attackers' sending rate (RPS), the blue solid line to represent DDoS target's receiving traffic rate (RPS), the green dot-dashed line to represent legitimate clients' sending rate (RPS). Therefore, when blue solid line meets the green dot-dashed line, all traffic arriving at the DDoS target is from legitimate clients. The area below the red dashed line and above the blue solid line represent the DDoS mitigation provided by FITT over NDN.

NDN: Resilience to Valid-S Interest Flooding Figure 4.12 demonstrates NDN's DDoS resilience to static (Valid-S) Interest flooding with NDN's intrinsic properties, i.e., Interest aggregation and in-network cache. To be more specific, we first disabled cache in all routers so that the result will only be affected by NDN's Interest aggregation. As shown in Figure 4.12a, NDN can withhold traffic from attackers (red dotted line) to the server (blue solid line). The more available names are used, the less traffic NDN can retain. We then introduce cache capacity of 200 data packets in Figure 4.12b. It is apparent that the number of Interests reaching P decreases because of the caching capacity, which is because



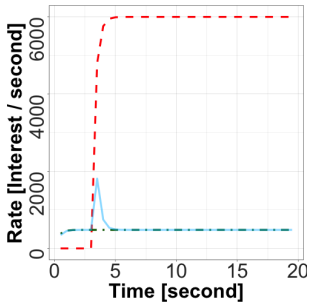
(a) Interest Aggregation



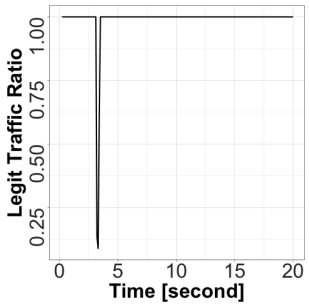
(b) In-network Cache

Simulation Settings: NDN deployment without FITT. Each attacker (60 in total) start sending Valid-S attacking Interests at 100 pkt/s from second 3 with available number of data names 500 and 1000, respectively.

Figure 4.12: NDN's DDoS Resilience to Valid-S Interest Attack



(a) Invalid Interest DDoS



(b) Legitimate Traffic Ratio

Simulation Settings: NDN deployment with FITT. Each attacker (60 in total) starts sending invalid Interests at 100 pkt/s from second 3. Legitimate clients' sending rate is 40 pkt/s starting from second 0.

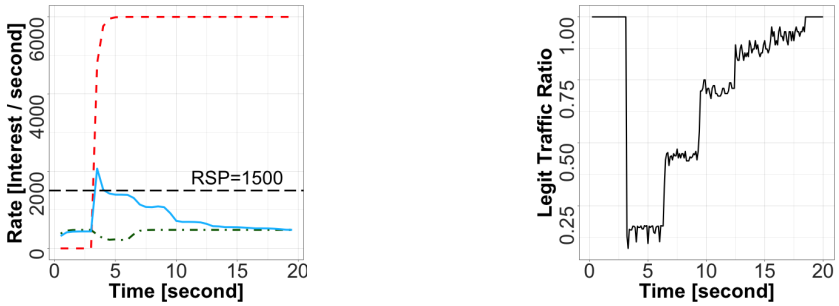
Figure 4.13: FITT mitigation of invalid Interest attack

intermediate nodes along the path will serve future same Interests with cached Data (the freshness of cached Data is 4 seconds in our simulation).

The two figures indicates that the effect of Interest aggregation and cache is lower when an attacker can use a bigger set of Interest names to attack the victim. This is because larger the name set, smaller the chance of two Interests carrying the same name and smaller the chance to hit a previous cached Data packets.

FITT: Invalid Interest Attack We first study FITT's performance against Invalid Interest (Invalid) DDoS attack. As shown in Figure 4.13a, initially, P only receives Interests from legitimate clients (green dot-dashed line). After 3 seconds, attackers start the DDoS by sending Invalid Interests (the red dashed line) to P. As depicted by the plot, P's incoming traffic line (blue solid line) immediately goes up after the attack but soon goes down and merge the legitimate clients' outgoing traffic line (green dotted line). Therefore, FITT can eliminate the invalid Interest DDoS traffic in a short time. The effectiveness is because in invalid Interest DDoS attacks, FITT can accurately identify attackers by *InvalidNames* carried in feedback messages and throttle their attack traffic. Figure 4.13b shows that after FITT reaction, all the traffic received by P is from legitimate clients.

FITT: Valid Interest Flooding We then simulate valid Interest flooding. To remove the effect of in-network cache, we disabled all router's cache. As such, Valid-S and Valid-D Interest flooding become the same because all of them will arrive at the DDoS target P. The results of Valid-D Interest flooding are shown in Figure 4.14.

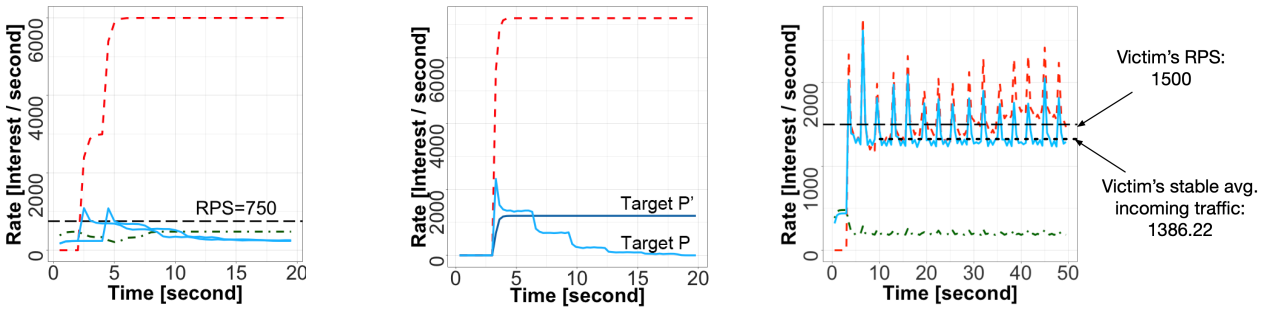


(a) Valid-D Interest DDoS (b) Legitimate Traffic Ratio

Simulation Settings: NDN deployment with FITT. Each attacker (60 in total) starts sending Valid-D Interests at 100 pkt/s from second 3. Legitimate clients' sending rate is 40 pkt/s starting from second 0. We let P's capacity be 1.5K RPS and *RateLimitTimer* be 3 seconds.

Figure 4.14: FITT mitigation of Valid Interest flooding

Since the router cannot tell good traffic from bad traffic when DDoS starts, legitimate clients are also limited. After receiving the FITT feedback messages, legitimate clients will



(a) Two-victim DDoS attack (b) Fine-grained throttling (c) "Smart" attackers

Figure 4.15: FITT mitigation under different scenarios

abide by the control placed and lower down their sending rate until the router determines them to be legitimate and free the limits, explaining why the green dot-dashed line goes down in the first several seconds of the attack and then back to the normal later. As for attackers, as shown, the traffic received by the victim drops periodically (every 3 seconds), which confirms the FITT's reinforcement throttling: FITT will halve the limit on attackers until all the attackers' traffic to the reported prefix are totally blocked. At the end of the mitigation, >99% of the Interests received by the victim are from legitimate clients (Figure 4.14b).

FITT: Multiple Attacks to Different Prefixes FITT is designed not only to handle single DDoS attack but also comprehensive DDoS attacks, i.e., attacks to different prefixes, starting at different time and using different types of Interests. In this simulation, we evaluate a more complicated attack scenario where half of the attackers attack service P with Valid-D Interests starting from second 2 and another half attack the another service P' with Valid-S starting from second 4. We found the simulation results of scenarios when P and P' are located on the same node or different nodes are almost the same. Figure 4.15a shows the result when P and P' are running on the same server.

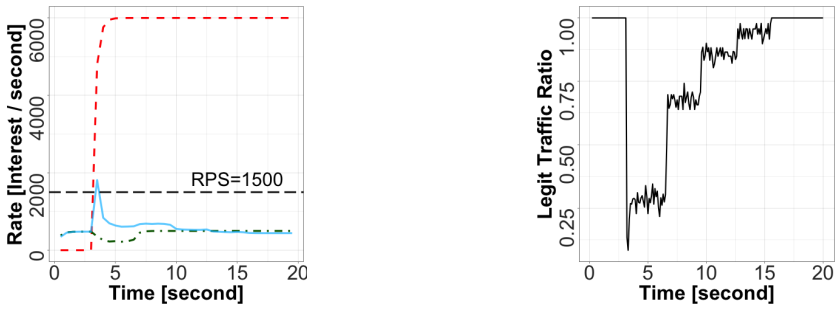
As shown, when multiple FITT mitigation instances take place, FITT can effectively control the DDoS traffic from both attacks at the same time. For each victim server, the

incoming Interests are throttled in the similar way as that when there is only one victim server under attack. Two servers' incoming traffic lines quickly go below the threshold after the attack started and soon merge the legitimate client traffic line, indicating that all the traffic received by the two servers are from legitimate clients.

FITT: Throttling Granularity In DDoS mitigation, collateral damage may ruin the legitimate traffic sent from the compromised devices. FITT throttles Interest traffic at a granularity of flow under a specific name prefix. We evaluate FITT in terms of the granularity of the traffic throttling. We reuse the simulation settings of the two-victim scenario to let all the attackers not only attack P with Valid-D Interests at 100 pkt/s but also keep the normal communication with another service provider P' at the reasonable rate of 20 Interests/s. In this case, the attacking traffic will overwhelm P but the legitimate traffic will not go beyond the capacity of P'. As shown in the Figure 4.15b, compared with the two-victim scenario where traffic to both P and P' will be throttled, in this simulation, FITT only squelches clients' traffic under P while the traffic towards P' will not be affected.

FITT: "Smart" Attackers Attackers may try to circumvent FITT's traffic throttling by complying feedback messages from edge routers temporarily and switching back to attack mode later. However, when attacks switch back to high sending rate, the traffic volume will alarm the service's pre-configured threshold again, which will in turn force the attackers to lower down the Interest sending rate again. In this way, the FITT will be triggered periodically when switches take place (Figure 4.15c). Consequently, the average attacking traffic will be kept at a certain level where the service will not be spoiled, especially when the threshold is properly set below the real capacity of the service.

FITT: Mixed Interest Attack We tune the attackers in Valid Interest flooding scenario to send both Valid-D and Invalid Interest packets to simulate a mixed Interest attack.



(a) Mixed Interest DDoS (b) Legitimate Traffic Ratio

Simulation Settings: NDN deployment with FITT. Each attacker (60 in total) starts sending both Invalid and Valid-D Interests at 100 pkt/s from second 3. Legitimate clients' sending rate is 40 pkt/s starting from second 0. We let P's capacity be 1.5K RPS and *RateLimitTimer* be 3 seconds.

Figure 4.16: FITT: Mixed Interest Attack

As shown in Figure 4.16, compared with results of Valid-D Interest flooding scenario, one obvious difference is that, after the attack starts, FITT will drop the traffic to be much lower than P's RPS (black horizontal line). This is because at the edge, invalid Interest attack will lead to a total block of the attackers, which cancels out the RPS from the valid Interest attack mitigation. After a short period, FITT will place the limit to misbehaving clients only and the legitimate clients will recover. In the end, FITT will only pass legitimate traffic to P (Figure 4.16b).

4.3 DLedger: Distributed Immutable Logging

The majority of the materials contained in this section were modified or quoted verbatim from [ZVM19].

4.3.1 Background and Motivation

The open access and lack of trustworthiness among peers in public or permissionless blockchain do not apply to permissioned application systems where strong trust relationship need to be defined and enforced among a group of entities. For example, in a smart grid network

system, the participants like power plants, public utilities, and government agencies [The14] do not fully trust each other but share the same goal of providing reliable power support. Under the same goal, each recorded data (*e.g.* logs of executed or pending commands) should be strictly validated before it can be confirmed and accepted by the system, *e.g.*, the data generator is authorized against the policies and the content is correct against some out-of-band or online query. Such a permissioned system should also provide high throughput, low latency, and fault tolerance for application use.

The state-of-the-art permissioned blockchain system, Hyperledger Fabric [ABB18], improves throughput and reduce latency by letting each transaction be executed only by a subset of the peers and thus parallel execution. Nevertheless, the writing of each block into the chain still remains a single head-of-link blocking process because Fabric uses one global blockchain and employs a fully-synchronized, conceptually-centralized ordering service to establish the order of transactions. In addition, Fabric uses TLS as the underlying communication channel for gossip and thus defeat the asynchronous communication and subject to network partition.

4.3.2 Design of DLedger

In this paper, we describe DLedger, an open-source permissioned distribute ledger system over directed acyclic graph (DAG) data structure and Named Data Networking (NDN) as a secure and asynchronous peer-to-peer (P2P) network. DLedger adopts a number of new concepts and techniques:

- Parallel block writing and hard fork avoidance allowed by the DAG.
- Reference based endorsement allows blocks to be confirmed when a subset of the peers endorse it according to the policy. This is similar to Fabric's idea but differs in the following two aspects. First, each block is endorsed differently per pre-defined policies. For example, more critical logs require a larger number of endorsements or endorse-

ments from certain entities. Second, the endorsements themselves are also recorded by the DLedger for auditing purpose.

- Co-design of communication layer using NDN and data layer using named secured block. DLedger let each block a signed and named NDN Data packet. Each block is a node in the DAG data structure in the data layer and is a self-containing unit that can be cached, fetched, and verified in the communication layer.
- No special centralized functions.
- A number of endorsement policies to improve security of the system and prevent lazy peers.
- Support of distributed programs (like smart contracts used in Ethereum) written in standard programming language with Web Assembly Interface (WASI).

Our design favors availability (*i.e.*, liveness) over consistency (*i.e.*, safety) and provide eventual consistency considering possible network partitions. Specifically, when a network partition happens, critical blocks can be written into the system without any hard forks, but depending on the endorsement requirement of different type of blocks, they may not be confirmed until the networks rejoin.

Communication Layer Entities utilizes Named Data Networking (NDN) for data dissemination and synchronization. All entities in a DLedger system, including the identity manager, together form a peer-to-peer (P2P) network. Every record are generated with a name, and other peers can fetch this data from NDN network directly using the name.

Entities advertise latest records to the whole P2P network. This process triggers both periodically and after out-of-sync events (e.g., new record generation, sleeping mode, network failures, etc.). After receiving the synchronization Interest packets, entities will fetch the

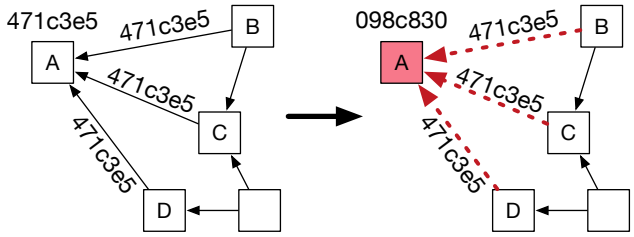


Figure 4.17: Endorsement Ensures Immutability

missing records and add it to their local ledger after record validation. Leveraging the synchronization protocol, DAG is replicated and synchronized among all entities.

The benefits of using NDN as the P2P network brings a number of benefits:

- Being an NDN Data packet, each block can be fetched, cached, verified from any peer in an asynchronous manner. This is entirely handled by the NDN protocol stack and thus there is no need to design and implement it in the application logic (*e.g.*, by building a distributed hash table to keep a block-entity mapping and implementing a open-access database of blocks on each node).
- NDN allows efficient Data multicast. When spreading a new block to the entire network, conventional gossip network will cause a large number of duplicate transmission. In NDN, this is done in a request-and-response way where only lightweight Interest may be duplicatedly transmitted and the heavy block will be shared without any bandwidth waste.

Data Layer DLedger stores all records in a Directed Acyclic Graph (DAG) instead of a single blockchain. Each vertex represents a record carrying its payload data while each edge represents an *approval* made by a record to another. In DLedger's DAG, a newly generated record (vertex) will be placed adjacent (chained) to n ($n \geq 2$) existing records in the DAG, establishing the approvals from the new record to the previous ones. *Tailing* records are those records not approved by any other records in the DAG.

When a peer P generates a new record R, the peer takes the steps as follows.

1. P randomly selects n tailing records generated by other peers.
2. It will then verify the validity of these records and the records directly or indirectly approved by these records. If any of the n records is invalid, i.e. the record is malicious or it approves an invalid record, P should drop it and repick another.
3. The peer then adds the names of the n selected records and the application payload into the new record R and sets R's name to:

   /DLedger/<Generator ID>/<Record Hash>

   where <Generator ID> is P's unique ID and <Record Hash> is the digest calculated over R. This will ensures the immutability of the blocks being referred by the current block because a modification of payload will lead to broken references in later blocks (Figure 4.17).
4. Finally, P appends a digital signature for block authenticity and integrity.

Each peer appends its new records into the DLedger after making *approvals* to other peers' records whose validity can be successfully verified. DLedger adopts a set of security policies for record receivers to check the incoming records, preventing potential threats such as spam attack and collusion attack, and misbehaving operations like laziness (contributing nothing to system security).

Specifically, besides the verification of packet format, the content, signature, and block creator's authorization, DLedger also has the following rules.

- Interlock Rule: Creator may only attach new blocks to blocks generated by other entities. This prevent a peer from maintaining a secret chain or DAG and greatly increasing the verification time of blocks (because a verifier needs to verify a entire sub DAG or chain.)
- Generation Rate Policy: Creator must generate block below a certain rate to prevent flooding.
- Contribution Rule: Creator may only attach new records to relatively new unconfirmed

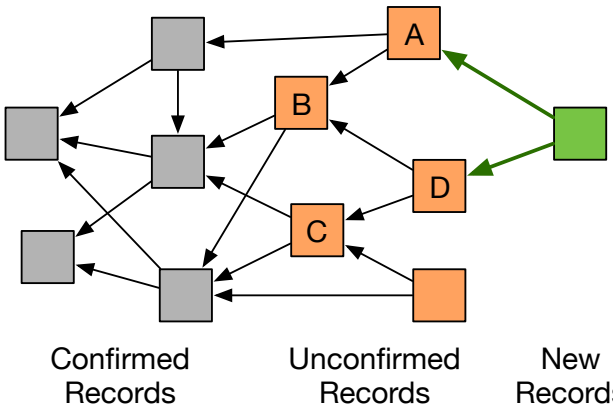


Figure 4.18: DAG, Endorsement, and Consensus in DLedger

records. This prevent lazy peers from skipping verification of unconfirmed records.

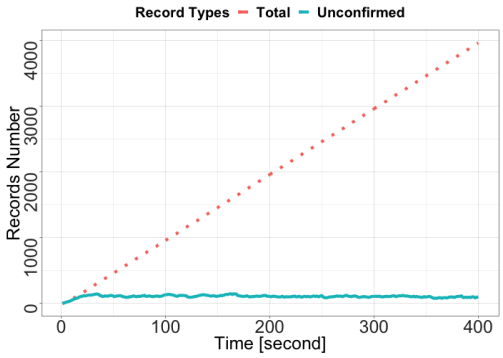
Consensus Layer The consensus on a record in DLedger is realized by enough number of endorsements made by a number of entities. The number of endorsements from *different entities* is called the *weight* of a record. As shown in Figure 4.18, the green block increases the weight of record A and D, and indirectly increase the weight of B and C. When a record gains sufficient number of endorsement according to the security policy, it is confirmed and accepted by the whole group. This assures a (k, N) $k < N$ threshold scheme where unless more than k peers of the total N peers get compromised, the system remains secured, according to the trust model.

Application Data Format Layer The distributed ledger stores multiple types of record content:

- Application records, *e.g.*, the log of executed and pending commands to a smart grid system.
- Identity management operations which include certificate issuance and revocation.
- Security policy records.
- Executables (*i.e.*, smart contract) in the compiled binary format following WASI.

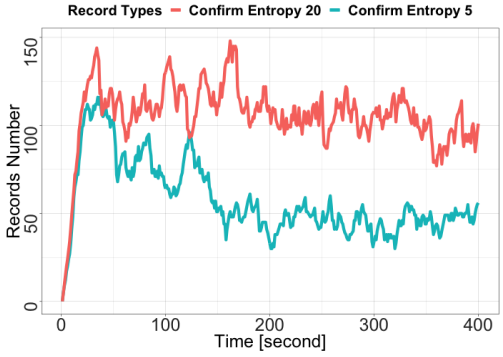
4.3.3 Evaluation

In this section, we evaluate DLedger's robustness, scalability, and ability of handling network partition through the theory analysis and our simulation results over ndnSIM [**?**], an NS-3 based simulation platform for NDN.



Simulation Settings: 50 entities P2P network with $W_{confirm} = 20$. Each entity generates a new record every 5 seconds.

Figure 4.19: Unconfirmed Records As DAG Grows



Simulation Settings: 50 entities P2P network with $W_{confirm} = 5$ and $W_{confirm} = 20$. Each entity generates a new record every 5 seconds.

Figure 4.20: The Unconfirmed Records With Different Weight

The Unconfirmed Records Size An essential concern of DLedger is its robustness: as system runs, will the size of unconfirmed records go infinitely large resulting in the system crash? As proved in [ZVM19], the tailing record size is concentrated around the constant:

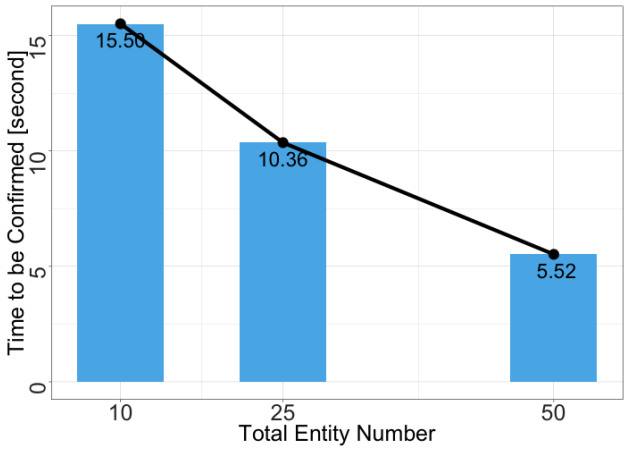
$$\frac{n\lambda T}{n-1}$$

where T is the average latency for a new record to be accepted by the system and λ is the new record generation rate of the system following a Poisson distribution. Moreover, the proof in [ZVM19] shows that the time for a record being confirmed is also a constant value. Therefore, since both the frontier (tailing record) and the depth (time to be confirmed) of DAG's unconfirmed records keep constant, the unconfirmed record size is also a constant value.

We evaluate DLedger by the simulation in terms of the unconfirmed record size as the DAG grows. To be specific, we set up a 50 entities P2P network with $W_{confirm} = 20$, where each entity's record generation rate is 0.2 records/sec. As shown in the Figure 4.19, the unconfirmed record number will not increase with the DAG's growth. Instead, the unconfirmed record number line will increase for a short time after bootstrapping, and then fluctuate around a constant value, confirming the mathematics proof.

The Figure 4.20 reveals the relationship between the $W_{confirm}$ setting and the unconfirmed record size when the total entity number is the same: by adjusting the $W_{confirm}$ from 5 to 20, the unconfirmed record size will increase by only about 100%.

System Scalability We argue that DLedger has better scalability because of the following two reasons. First, DLedger leverages DAG to keep the records. Instead of allowing only one successful next block in the blockchain, DLedger accept multiple valid next records, thus allowing multiple entities contributing to the system in parallel. This gets rid of the computation waste and frees the limit on record processing speed. Second, PoA is used as the gating function. Unlike PoW which is highly CPU bound, PoA is much cheaper and efficient even for constrained IoT devices, greatly improving the system efficiency.



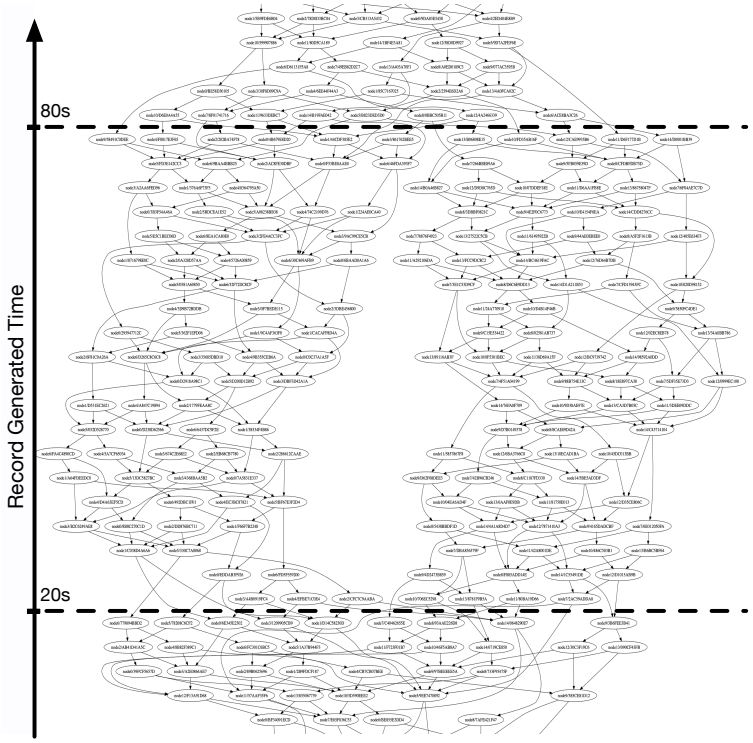
Simulation Settings: $W_{confirm} = 5$ with total entity number 10, 25, and 50.

Figure 4.21: Unconfirmed Records As DAG Grows

We also show DLedger's performance when the number of entities grows. As shown in Figure 4.21, with fixed $W_{confirm}$, the more entities participate, the faster a record will be confirmed. Because for any specific record, more entities means more efforts on approving it.

Network Partition We evaluate DLedger's performance in the case of network partition: in the simulation, we split the network into two independent subnets by taking down the links between them. The partition takes 100 seconds and two networks unit again after that. Figure 4.22 shows a DAG maintained by an entity at the end of the simulation. As shown, two branches have formed and then merged, showing DLedger's ability to recover from the network partition, eventually confirming records from each subnet.

Note that each branch of the partition needs to have more peers than confirmation weight for this branch to be confirm and recognized by other branch; this prevents entities collude and flood bad records using a branch.



This shows a DAG from the simulation. Each circle in the figure is a record. The links among records are the approvals. The recent records are higher in the DAG as shown in the figure.
Simulation Settings: 15 entities P2P network. The partition takes place at second 20, dividing the network into a 7-node subnet and a 8-node subnet. Two networks rejoin at 80 seconds.

Figure 4.22: Unconfirmed Records As DAG Grows

4.4 NDN-MPS: NDN Multiparty Trust Support

The majority of the materials contained in this section were modified or quoted verbatim from [].

4.4.1 Background and Motivation

Since relying on a single party for critical decision making is prone to errors and single point of failures, many real world systems requires jointly decision making process where multiple parties are involved. For example, in the critical smart grid system, to ensure power supply

safety, a command to turn on or off a power plant can only be executed if it has been approved by a number of related parties, including the power plant manager, the public utility service providers, the authority agent from government, etc.

Recently, Named Data Networking (NDN) [ZAB14] has started being explored and experimented by the industry to provide network and security support for smart grid systems. While NDN provides adequate mechanisms for the producer-consumer security model and supports signature schemes like RSA and ECDSA, little work has been done to support multiparty authentication and authorization. To fulfill the gap, our goal is to design and implement the components for NDN to provide secure, efficient, and usable multiparty signature support.

4.4.2 A New Trust Model

In a multiparty signing system, there are three types of parties.

- **Initiator**. An initiator is called by the data-producing application to generate a multi-signature for the data to be signed. The initiator will find a group of signers based on application's requirements (*e.g.*, defined in the trust schema), collect signature pieces from individual signers and aggregate them into a multiparty signature. Finally, the initiator will return the signature back to the application.

- **Signers**. Multiple signers need to jointly generate a multiparty signature. Specifically, when the data to be signed can be approved, each signer will make a signature piece that can be aggregated with pieces from other signers.

- **Verifier**. A verifier is called by the application to verify the multiparty signature along with the data covered by the signature. If the signature is valid and its signers satisfy the application's requirements, the verifier will return positive results to the application.

We separate them based on their functional logic. In practice, a single identity can play

multiple roles. For example, the initiator can also be one of the signers.

To ensure the security of the system, the following policies should be explicitly and rigorously defined by the trust schema.

- From a signer's perspective, who can be initiators? In other words, how can the signer know whether a given entity can make a signature request?
- From a verifier's perspective, which combination of signers can generate a legitimate signature for a given named data?

Note that whether a signer will accept a signature request does not only depend on whether the requester can be an initiator, it also requires approval from the application logic. Using the power plant turn-on command as an example, each signer should evaluate the result of the command before signing it.

In a multiparty signing scenario, the existing producer-consumer model cannot directly fit because (i) there are multiple signers of each signature, (ii) there is a new type of party, initiator, which should be clearly defined by the trust schema, and (iii) signers are no longer the producers. Thus, the existing trust scheme is insufficient to clearly describe the policies the new model.

The mismatch is also reflected by the current specification and implementation of NDN libraries. For example, the current key locator cannot carry sufficient meta information of multiple signers and the trust schema does not support the signing and verification of multisignatures.

4.4.3 Design of NDN-MPS

We propose NDN-MPS, an NDN based multisignature signing and verification tool set. These tools provide applications with the support of initiator, signer, and verifier in the context of multiparty signing scenario:

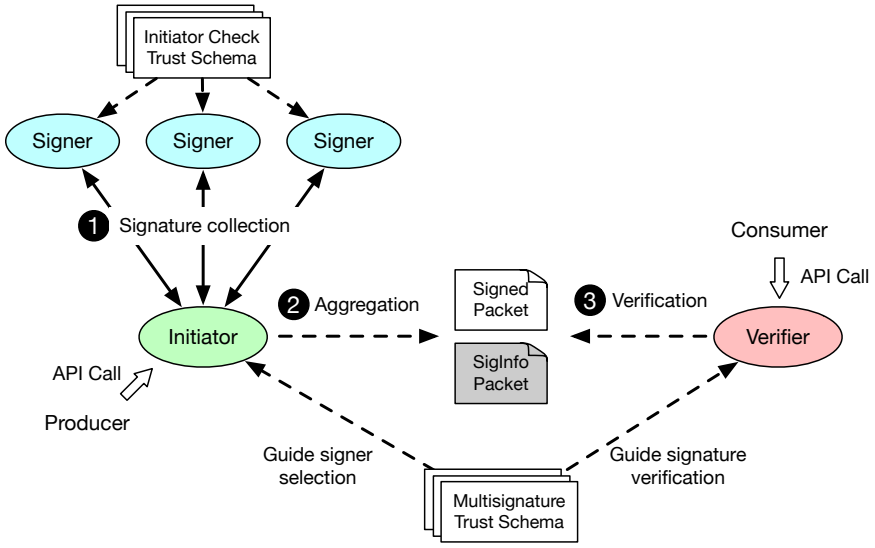


Figure 4.23: An Overview of NDN-MPS

1. New trust schema that supports the semantics for multiparty signing and verification by defining the policies of each party.

2. A signature collection protocol for an initiator to collect signatures from multiple signers.

3. A new type of key locator scheme to ensure the data integrity in the multiparty signing process and to accommodate complex signature information.

4. A new type of signature scheme, BLS signature.

NDN-MPS is designed as a security support tool set for applications. NDN-MPS consists of three main components (Figure 4.23). First, a new type of trust schema that allows applications to express the signing and verification policies of multisignatures. Second, an extension of the key locator in NDN Data packet to keep the data to be signed consistent across signers and allow complex signature information. Third, a signature collection protocol for an initiator to collect signatures from multiple signers.

These three components are used in the whole process of the multisignature signing and verification.

System Bootstrapping (Assumptions) Before an initiator starts a new signing process, we make the following assumptions about the system.

- Reachable prefix. The initiator I knows the reachable prefix of each signer S_i in the network. If a signer's prefix is not publicly reachable, the initiator will also need a forwarding hint to help reach the signer.
- Public keys installation. The verifier V has installed the public key along with the identity name of each signer. Such information can be obtained in a form of an NDN certificate with inline or out-of-band means.
- Proof of possession. When installing a public key of a signer, a proof of possession (PoP) or equivalent (*e.g.* a self-signed certificate) should be provided by the signer to prevent rogue key attack [BGW20].
- Awareness of the trust schema. The initiator I is aware of the trust schema to generate a sufficient signature for verifier V to validate. In addition, each signer has the trust schema for validating the identity of the initiator.

Process Initiation The initiator starts the signing process when the application provides it with the unsigned data and the multisignature trust schema. After that, the initiator will first work out a list of signers whose aggregate signature can satisfy the trust schema.

Signature Collection The initiator then starts collecting signature pieces by performing a RPC with each signer. The RPC protocol used in NDN-MPS that is based on NDN RPC [] but provides stronger security and privacy.

In a nutshell, the initiator will send a signed request to the signer and present the unsigned Data packet as the RPC parameter. Importantly, to ensure the consistency of the data to be signed among signers, the initiator will fill the key locator field with a placeholder name.

Upon receiving the request, each signer will first verify it against the trust schema to ensure that its sender is legitimate to act as an initiator. After that, the signer can make choice on whether to sign the data or not. In this process, the signer may need to query the application logic for approving the data to be signed before making a decision. If the signer agrees to sign the data, it will generate the signature over the data as it is without modifying the key locator or any other fields.

If any signer is not available or reject to sign the data, I will try to reach an alternative signer if it exists. Note that, the change of signers will not affect the value of the late-binding key locator, thus ensuring the consistency of the data being signed among signers.

Signature Aggregation After collecting sufficient signature pieces, the initiator I aggregates them to generate a single signature and attach it to the original unsigned Data packet. In addition, the initiator generates another Data packet, called a SigInfo packet, that contains the information of the signers, and names it under the prefix of the placeholder key locator name.

Signature Verification Before verifying a multiparty signed Data packet, the the verifier needs to obtain both the signed data packet and the SigInfo packet. Note that the SigInfo packet can be fetched after obtaining the signed packet by using the key locator name as an Interest packet.

After that, the verifier first checks the signer list from the SigInfo packet against the multisignature trust schema. If the signer combination does not generate a legitimate signature, the signature will be rejected. Then, the verifier verifies the multisignature using the public keys of the signers.

4.5 RapidVFetch: Secured Data Fetching in Vehicular Network

The majority of the materials contained in this section were modified or quoted verbatim from [ZLD19].

4.5.1 Background and Motivation

In real world, due to the cost of deployment and support, RSUs can be deployed in many different environments with various densities and topologies [RST11].

In this paper, we target the problem of degraded data downloading performance for vehicles in areas with limited RSU coverage. To illustrate our problem better, we use a simplified example scenario; that is, a vehicle drives out of a previous RSU's coverage, loses the RSU connectivity, and then drives into the next RSU's range (Figure 4.24). In addition, we separate the target applications into two categories. For non real-time applications, the problem is how to improve data downloading rate by fully utilizing the limited connectivity time. For real-time applications, the problem is how to ensure a consistent downloading rate for the user.

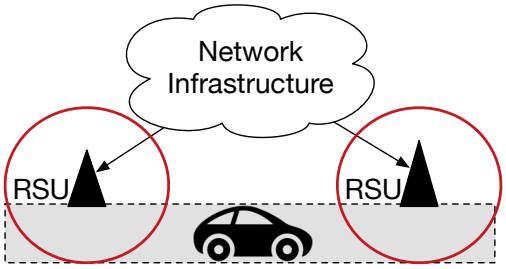


Figure 4.24: Example Scenario of Vehicular Prefetching

Content prefetching in mobile environments to reduce latency in data download has been well explored in a number of existing works [JK98, NN08, Cho02]. In the context of vehicular networking, due to the intermittent connectivity between vehicles and RSUs, prefetching of data content becomes vital for ensuring the quality of information services.

Prior works such as [DK09, GS18, HZ16, KL15] calculate the possible trajectory of the vehicle and content popularity, then prefetch different chunks of content to RSUs to maximize data retrieval rate. Fundamentally, RSUs fetch the required content from the cloud server based on a prediction model, and vehicles fetch desired content from RSUs when moving inside their signal range. The prediction and caching strategy aims to facilitate this content fetching process with minimum delay.

These approaches were proposed to run over IP, but its design lean towards a data-centric solution as their proposed mechanisms make data independent of the IP's point-to-point channels. Following this direction, it becomes ideal to apply a data-centric solution to the vehicular content prefetching problem to avoid the overhead of address allocation and connection state maintenance.

A common issue with existing IP-based and NDN-based prefetching approaches is that their performance is dependent on using a centralized server for accurate predictions, as inaccurate estimation causes waste in storage resources, and hurts download performance. In this paper, we propose a requester-initiated prefetching solution to better satisfy the real-time needs of vehicles, and increase the utilization of in-network storage. Our solution does not make predictions based on predefined models, neither does it depend on prior travel paths nor request history, but prefetches content on RSUs based on real-time user requirements in a timely manner.

An existing approach following a similar idea was [RZ13], which allowed vehicles to signal the currently connected RSU when a handover process was going to happen, and notify the following RSU to prefetch the required content. Therefore, the protocol is not transparent to RSUs. Other than [RZ13], existing approaches like [AB17] [GS18] also make use of RSU and infrastructure side routers for content caching. This means RSUs and wired infrastructure side routers need software upgrades to support new protocols, which increases deployment difficulties. Rather than relying on RSUs or infrastructure routers to cache content, in RapidVFetch, we take an alternative approach by making use of V2V

communication to initiate content prefetch requests and content delivery. In doing so, we do not need to upgrade existing RSU infrastructure, and only require the vehicles involved in the communication process to support our protocol. This also reduces the attack surface on RSUs as we can authenticate the communicating vehicles through physical connectivity to the RSUs.

Another issue with existing work is the lack of security considerations in prefetching. The prefetching process is asynchronous and how to ensure integrity, authenticity, and confidentiality of prefetched data becomes a main challenge. In RapidVFetch, we provide an approach to protect sensitive information in NDN names and Data content from eavesdropper (e.g., other vehicles, RSUs).

4.5.2 Design of RapidVFetch

We propose RapidVFetch to facilitate vehicular data downloading by prefetching named, secured data via V2V communication over NDN. Intuitively, since it is the vehicle who best knows its future locations and desired content, RapidVFetch lets each vehicle, called a *requester*, express their needs by Interest packets which are small in size and solicit help from other vehicles, called *forwarding vehicles*, through V2V communication. Vehicles at future locations can simply express these small Interest packets to the RSUs, where the prefetched packets will be cached at the RSUs and will soon be used by the requester when it moves into the RSU's range. For real-time applications, the data can also be forwarded back to the requester.

RapidVFetch provides the following desirable features.

- Having each requester take care of their own needs, RapidVFetch provides accurate content prefetching at desirable RSUs without reliance on centralized services.
- By utilizing forwarding vehicles, RapidVFetch allows a requester to maintain stable data downloading for real-time applications.

- Enabled by NDN's built-in security, RapidVFetch ensures data authenticity and confidentiality regardless of the trustworthiness of other vehicles or RSUs involved in the downloading process.
- RapidVFetch is transparent to the infrastructure and RSUs, that is, RSUs and backbone routers do not need to be aware of RapidVFetch.

The goal of RapidVFetch is to facilitate and secure the data downloading from the infrastructure to vehicles by utilizing broadcast V2V communication and NDN's content-centric networking model. To be more specific, for non real-time applications, RapidVFetch aims to improve data downloading rate by fully utilizing limited connectivity time; for real-time applications, the goal is to keep the requester online and ensure a smooth downloading rate.

In a nutshell, as shown in Figure 4.25, a *requester* contacts a *forwarding vehicle* in the target RSU's range through V2V communication (which may contain single or multiple hops) to prefetch data (❶). For non real-time applications (e.g., video downloading), when the requester drives into that RSU's range, it can re-send Interests for the prefetched content and benefit from proactive caching (❷), improving utilization of the limited connectivity to RSU. In the case when real-time communication is required (e.g., online conferencing), the forwarding vehicle can also forward the replied content back to the requester in order to keep the requester online (❸). Importantly, RapidVFetch considers each Data's security in the downloading process. The downloaded Data packets are signed and can also be encrypted using NDN's built-in security primitives and name-based access control (NAC) [ZYR18].

By letting each requester initiate their own prefetch Interest packets, RapidVFetch always provides accurate prefetched data at target RSUs, getting rid of the centralized service for content/location prediction. In addition, this allows the requester to have a better control on when to trigger the prefetching, so RapidVFetch has more flexibility to cope with complicated road conditions (e.g., unexpected delay). For example, if there is a traffic congestion before entering the next RSU, the requester can delay the prefetch. At the same time, since a RSU

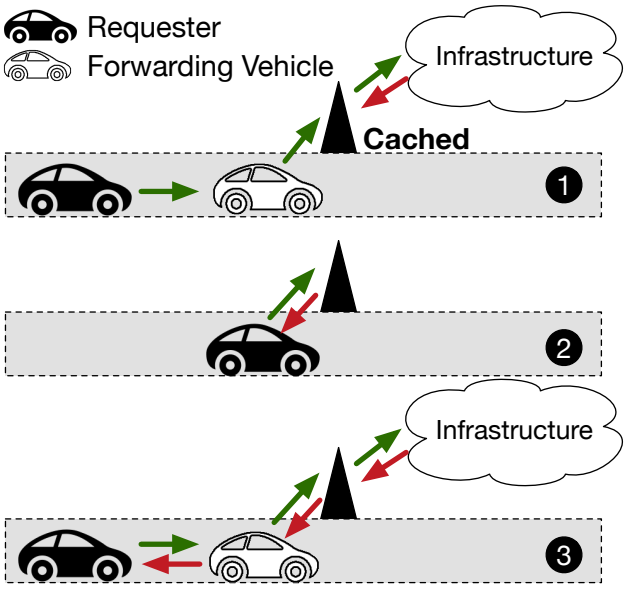


Figure 4.25: An Overview of RapidVFetch

does not need to treat prefetching Interests differently, the RSUs do not need to be aware of the RapidVFetch protocol. Consequently, RapidVFetch can be deployed at vehicles only without bothering RSUs and the infrastructure behind RSUs.

Our work illustrates the superiority of using a data-centric network architecture in vehicular networking: NDN enables RapidVFetch to let vehicles themselves pick what to cache in desired RSUs without reliance on a predictor; at the same time, the prefetched data is secured in a data-centric way and the whole process can be transparent to the infrastructure.

4.5.3 Securing Vehicular Data Prefetching and Downloading

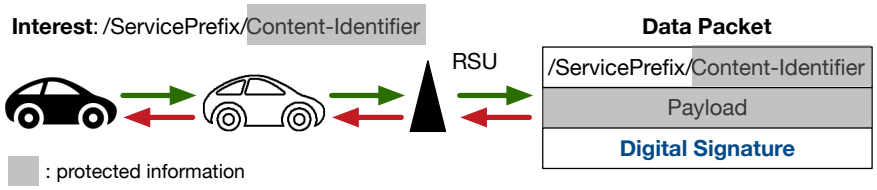
In RapidVFetch, we consider the security of both the data prefetching process and the downloading process, which further includes (i) authentic use of RapidVFetch on RSUs, (ii) integrity and authenticity of downloaded data, and (iii) user privacy and the confidentiality of downloaded data.

Authentic Use of RapidVFetch on RSUs It is nontrivial to authenticate users in an open wireless network system. To date, a number of vehicular security architectures have been proposed to employ a Public Key Infrastructure (PKI) in vehicular networks. However, these approaches provide means to ensure a vehicle has a valid plate, the driver has a valid license, or the vehicle is made by a verified manufacture, but they are not sufficient to prove the application is honest, e.g., whether the application will send spam or DDoS traffic to a RSU. Therefore, in this work, to mitigate spam Interest packets towards the RSU, we consider a simple but reliable means to authenticate a user: only forwarding Interest packets sent by users within RSU coverage.

This feature also distinguishes our work from existing works. For example, [RZ13] proposes a solution where a previous RSU issues an Interest packet to the next RSU. However, it is difficult for a RSU to verify that the Interest is from another RSU and is truly originated by a vehicle that will later drive into its own coverage. In contrast, RapidVFetch reduces the attack surface and RSUs can keep the strict access policy (i.e., traffic within coverage) without being aware of the RapidVFetch protocol.

Data Integrity and Authenticity Acting as an Interest forwarder, the forwarding vehicle may direct the Interest to a fake service provider maliciously; as can the RSU, which may alter replied Data packets in their cache. Such alteration or fake Data packets will degrade the downloading and may cause damage to the requester. It is difficult for today's channel-based security model (e.g., TLS, QUIC) to ensure the data integrity/authenticity because the data will be cached at the RSU. This is because by the time the data is used by the requester, the secured channel does not exist any more and thus no security can be guaranteed.

However, empowered by NDN's built-in security, in RapidVFetch, every Data packet is signed (Figure 4.26) by its original producer, e.g., the video streaming service provider. Consequently, such alteration or fake Data packets will be detected by verifying Data packet



Each Data packet carries a digital signature. Sensitive information in Interest name, Data name, and Data content will also be protected.

Figure 4.26: Content-centric Security in RapidVFetch

signatures. If the data has a bad signature, the requester should resend the Interest packets and fall back to the normal downloading mode.

User Privacy and Data Confidentiality A forwarding vehicle is able to know the Interest packets from the requester and the RSU can also grab information from the replied Data packets. This raises the user privacy concern for potentially leaking information of requester's ongoing actions and desired content. However, NDN names are not necessarily readable to eavesdroppers and the Data payload can also be protected through cryptographic techniques (Figure 4.26).

As a simple and effective solution to prevent information leakage, the requester and the service provider can negotiate a shared secret (e.g., through Diffie-Hellman protocol) asynchronously (e.g., ahead of vehicular communication). The shared secret can then be used to keyed hash or encrypt the sensitive name components in the Interest packets and content in the Data packets. For example, assuming the service provider's routable prefix is /service1 and the full name prefix of the downloaded data is /service1/movie/movie1/frame, the shared secret can then be used to hide the name components after /service1 and the replied Data payload.

A more systematic data confidentiality solution is to apply name-based access control (NAC) [ZYR18]. To be more specific, the service provider acts as an access manager who grants the application the access (i.e., decryption keys) to certain content. The access right

granting is based on the service provider's policy and application's identity, which is represented by a public key certificate. By the NAC scheme, the content will be encrypted with encryption keys while the decryption keys will be distributed to authorized applications secretly (i.e., in ciphertext). In addition, different encryption keys protect data under different name prefixes, allowing a fine-grained and flexible access control.

CHAPTER 5

Conclusion

We highlight that what applications need is named secured data and it is an observed new trend that both the network community and application community move towards the same direction from channel-based security to data-centric security. Before the time when data dissemination applications are emerging, the network was thinking how to provide better networks instead of how to better serve applications.

For each piece of named secured data, its name allows applications to locate and retrieve the data and the security protection ensures the security properties. When implementing named secured data in the application layer, it can be a signed HTTP file or JSON with an URL, an EL PASSO certificate with an URL and so on. When implementing it coherently in the network and application layer, it can be an NDN Data packet. Through the systems we explored in this paper, we show that NDN provides a solid platform for applications to secure their communications. Specifically, we confirm the desired properties of NDN security: building security in a data-centric way, NDN (i) reduces dependencies on the channel context, (ii) allows asynchronous data dissemination without relying on additional infrastructure (*e.g.*, today's CDN), and (iii) allows security policing directly with names.

While data-centric security and the concept of named secured data can be applied purely at the application layer to simplify the system and provide better security, a coherent network and application security support can further simplify system design by removing conflicting underlying component (*e.g.*, the mismatch between CDN and its underlying channel-based security).

We believe NDN's security support of naming and securing data directly can be the key advantage to push NDN's wide deployment in today's Internet. While directly modifying the Internet routers across the ASes lacks incentives for first movers, NDN and NDN security can be rolled out by two ways. The first way is to start deployment from edge networks, including smart home networks, vehicular networks, data center networks, etc. Importantly, NDN and NDN security should be able to facilitate services deployed in these edge network system to create incentives for stakeholders. An example is the Sovereign, which can facilitate security and privacy of smart home systems and the deployment cost is low because it requires no infrastructure change. When scattered end systems start to adopt NDN, it can push a larger deployment and deep into the Internet. Another way is to build an overlay NDN network to support distributed application systems. Well-known overlay networks that have been successfully deployed include (i) the overlay P2P networks deployed with BitCoin, Ethereum and other crypto currency systems, (ii) the P2P network used for onion routing, The commonality of these overlay network systems is they are needed by a killer application (*i.e.*, crypto currency and onion routing). Therefore, to push a large-scale NDN overlay deployment, a most important driver is the application. The design of DLedger has showed some of the potential benefits of using NDN as the underlying P2P network for a distributed system.

Future works can continue to focus on these two aspects, namely, building secure and usable edge network systems and supporting distributed applications with NDN. Incentives should be considered for the deployment and compared with existing infrastructure, NDN should provide better application support to allow killer applications in the future.

REFERENCES

[AAB17] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. "Understanding the mirai botnet." In *USENIX Security Symposium*, 2017.

[AB17] Noor Abani, Torsten Braun, et al. "Proactive caching with mobility prediction under uncertainty in information-centric networks." In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 88–97. ACM, 2017.

[ABB18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." In *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[Aka19] Akamai. "Akamai DDoS Protection.", 2019.

[Aka21] Akamai Technologies. "What does CDN stand for? CDN Definition.", 2021. [Online; accessed 15-Mar-2021].

[Ano19] Anonymous authors. "A Secure Sign-On Protocol for Smart Homes over Named Data Networking." *IEEE Communications Magazine*, **57**(7):62–68, July 2019.

[BCR16] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. "Off by default!" 2016.

[BGW20] Dan Boneh, Sergey Gorbunov, Riad Wahby, Hoeteck Wee, and Zhenfei Zhang. "BLS Signatures." Internet-Draft draft-irtf-cfrg-bls-signature-04, IETF Secretariat, September 2020. `https://tools.ietf.org/html/draft-irtf-cfrg-bls-signature-04`.

[BHG13] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas C Schmidt. "RIOT OS: Towards an OS for the Internet of Things." In *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 79–80. IEEE, 2013.

[BS04] F. Baker and P. Savola. "Ingress Filtering for Multihomed Networks." BCP 84, RFC Editor, March 2004.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. "Ciphertext-policy attribute-based encryption." In *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334. IEEE, 2007.

[CCJ11] Eun Kyoung Choe, Sunny Consolvo, Jaeyeon Jung, Beverly Harrison, and Julie A Kientz. "Living in a glass house: a survey of private moments in the home." In *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 41–44, 2011.

[CER99] CERT Coordination Center. "CERT Incident Note IN-99-04.", 1999.

[Cho02] Gihwan Cho. "Using predictive prefetching to improve location awareness of mobile information service." In *International Conference on Computational Science*, pp. 1128–1136. Springer, 2002.

[Clo19] CloudFlare. "CloudFlare Advanced DDoS Attack Protection.", 2019.

[CPZ16] Jianxun Cao, Dan Pei, Xiaoping Zhang, Beichuan Zhang, and Youjian Zhao. "Fetching popular data from the nearest replica in NDN." In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9. IEEE, 2016.

[DK09] Pralhad Deshpande, Anand Kashyap, et al. "Predictive methods for improved vehicular WiFi access." In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pp. 263–276. ACM, 2009.

[EE 13] EE Times. "MCU market turns to 32-bits and ARM." `https://www.eetimes.com/mcu-market-turns-to-32-bits-and-arm/`, 2013. Accessed: 2021-01-24.

[FK11] S. Frankel and S. Krishnan. "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap." RFC 6071, RFC Editor, February 2011. `http://www.rfc-editor.org/rfc/rfc6071.txt`.

[FKK11] A. Freier, P. Karlton, and P. Kocher. "The Secure Sockets Layer (SSL) Protocol Version 3.0." RFC 6101, RFC Editor, August 2011. `http://www.rfc-editor.org/rfc/rfc6101.txt`.

[FS00] P. Ferguson and D. Senie. "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." BCP 38, RFC Editor, May 2000.

[GS18] Dennis Grewe, Sebastian Schildt, et al. "ADePt: Adaptive Distributed Content Prefetching for Information-Centric Connected Vehicles." In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pp. 1–5. IEEE, 2018.

[GTU13] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang. "DoS and DDoS in Named Data Networking." In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, July 2013.

[HG04] Mark Handley and Adam Greenhalgh. "Steps towards a DoS-resistant internet architecture." In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pp. 49–56. ACM, 2004.

[HZ16] Zhiwen Hu, Zijie Zheng, et al. "Game theoretic approaches for wireless proactive caching." *IEEE Communications Magazine*, **54**(8):37–43, 2016.

[IC 15] IC Insights, Inc. "Microcontroller Sales Regain Momentum After Slump." `https://www.icinsights.com/news/bulletins/Microcontroller-Sales-Regain-Momentum-After-Slump/`, 2015. Accessed: 2021-01-24.

[IC 18a] IC Insights, Inc. "MCUs Sales to Reach Record-High Annual Revenues Through 2022." `https://www.icinsights.com/news/bulletins/MCUs-Sales-To-Reach-RecordHigh-Annual-Revenues-Through-2022/`, 2018. Accessed: 2021-01-24.

[IC 18b] IC Insights, Inc. "Microcontrollers Will Regain Growth After 2019 Slump." `https://www.icinsights.com/news/bulletins/Microcontrollers-Will-Regain-Growth-After-2019-Slump/`, 2018. Accessed: 2021-01-24.

[ISO15] ISO/IEC JTC 1/SC 27 Information security, cybersecurity and privacy protection. "ISO/IEC 27033-1:2015 Network Security Overview and Concepts." Technical report, ISO Standard, 2015.

[ISO19] ISO/IEC JTC 1/SC 27 Information security, cybersecurity and privacy protection. "ISO/IEC 27701:2019: Extension to ISO/IEC 27001 and ISO/IEC 27002 for privacy information management — Requirements and guidelines." Technical report, ISO Standard, 2019.

[JB14] Xiaoke Jiang and Jun Bi. "ncdn: Cdn enhanced with ndn." In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 440–445. IEEE, 2014.

[JBS15] M. Jones, J. Bradley, and N. Sakimura. "JSON Web Signature (JWS)." RFC 7515, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7515.txt`.

[JK98] Zhimei Jiang and Leonard Kleinrock. "Web prefetching in a mobile environment." *IEEE Personal Communications*, **5**(5):25–34, 1998.

[JST09] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. "Networking named content." In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 1–12, 2009.

[JWS03] Cheng Jin, Haining Wang, and Kang G Shin. "Hop-count filtering: an effective defense against spoofed DDoS traffic." In *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 30–41. ACM, 2003.

[KL15] Ryangsoo Kim, Hyuk Lim, et al. "Prefetching-based data dissemination in vehicular cloud systems." *IEEE Transactions on Vehicular Technology*, **65**(1):292–306, 2015.

[LB16] Huichen Lin and Neil W Bergmann. "IoT privacy and security challenges for smart home environments." *Information*, **7**(3):44, 2016.

[Mar21] MarketWatch, Inc. "IoT Microcontroller Market 2020 Global Industry Size, Opportunities, Key Vendors Analysis, Business Strategy, Future Plans, Competitive Landscape and Outlook 2023." `https://www.marketwatch.com/press-release`, 2021. Accessed: 2021-01-22.

[MCC14] Ge Ma, Zhen Chen, Junwei Cao, Zhenhua Guo, Yixin Jiang, and Xiaobin Guo. "A tentative comparison on CDN and NDN." In *2014 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 2893–2898. IEEE, 2014.

[MGS10] Erika McCallister, Tim Grance, and Karen Scarfone. *Guide to protecting the confidentiality of personally identifiable information*. National Institute of Standards and Technology (NIST) Special Publication 800-122, 2010.

[Ndn21] Ndn-cxx Contributors. "ndn-cxx: NDN C++ library with eXperimental eXtensions." `https://github.com/named-data/ndn-cxx`, 2021. Accessed: 2021-01-21.

[Neu19] Neustar. "Neustar Defense and Performance.", 2019.

[NFD21] NFD Contributors. "NFD - Named Data Networking Forwarding Daemon." `https://github.com/named-data/nfd`, 2021. Accessed: 2021-01-21.

[NN08] Anthony J Nicholson and Brian D Noble. "Breadcrumbs: forecasting mobile connectivity." In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 46–57. ACM, 2008.

[Nor21] Nordic Semiconductor Inc. "Nordic nRF52840." `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840`, 2021. Accessed: 2021-01-26.

[Orm03] H. Orman. "The Morris worm: a fifteen-year perspective." *IEEE Security & Privacy*, **1**(5):35–43, 2003.

[OSZ20] Eric Osterweil, Angelos Stavrou, and Lixia Zhang. "21 Years of Distributed Denial-of Service: Current State of Affairs." *Computer*, **53**(7):88–92, 2020.

[prn18] prnewswire.com. "The DDoS protection and mitigation market size is expected to grow from USD 1.94 billion in 2018 to USD 4.10 billion by 2023, at a Compound Annual Growth Rate (CAGR) of 16.1." https://www.prnewswire.com/news-releases/the-ddos-protection-and-mitigation-market-size-is-expected-to-grow-from-usd-1-94-billion-in-2018-to-usd-4-10-billion-by-2023–at-a-compound-annual-growth-rate-cagr-of-16-1-300645633.html, 2018.

[Res18] E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, RFC Editor, August 2018.

[RL96] Ronald L Rivest and Butler Lampson. "SDSI-a simple distributed security infrastructure." 1996.

[Ros14] Christian Rossow. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse." In *NDSS*, 2014.

[RST11] Andre B Reis, Susana Sargento, and Ozan K Tonguz. "On the performance of sparse vehicular networks with road side units." In *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pp. 1–5. IEEE, 2011.

[RZ13] Ying Rao, Huachun Zhou, et al. "Proactive caching for enhancing user-side mobility support in named data networking." In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 37–42. IEEE, 2013.

[SAN21] SANS Institute. "SANS: Network Security Resources.", 2021. [Online; accessed 15-Mar-2021].

[SLC19] Manu Sporny, Dave Longley, and David Chadwick. "Verifiable Credentials Data Model 1.0." Technical Report W3C Recommendation 19 November 2019, W3C, November 2019.

[SS75] Jerome H Saltzer and Michael D Schroeder. "The protection of information in computer systems." *Proceedings of the IEEE*, **63**(9):1278–1308, 1975.

[SWA17] Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. "Breaking out of the cloud: Local trust management and rendezvous in Named Data Networking of Things." In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pp. 3–13, 2017.

[The14] The Smart Grid Interoperability Panel – Smart Grid Cybersecurity Committee. "NISTIR 7628 Rev. 1: Guidelines for Smart Grid Cybersecurity." Technical report, NIST, 2014.

[TST17] Samuel Tweneboah-Koduah, Knud Erik Skouby, and Reza Tadayoni. "Cyber security threats to IoT applications and service domains." *Wireless Personal Communications*, **95**(1):169–185, 2017.

[UJS13] Blase Ur, Jaeyeon Jung, and Stuart Schechter. "The current state of access control for smart devices in homes." In *Workshop on Home Usable Privacy and Security (HUPS)*, volume 29, pp. 209–218. HUPS 2014, 2013.

[Wik21a] Wikipedia contributors. "ARPANET.", 2021. [Online; accessed 15-Mar-2021].

[Wik21b] Wikipedia contributors. "Forward secrecy.", 2021. [Online; accessed 15-Mar-2021].

[Wik21c] Wikipedia contributors. "Network Security.", 2021. [Online; accessed 15-Mar-2021].

[YAC15] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. "Schematizing Trust in Named Data Networking." In *Proc. of ACM ICN*, 2015.

[Yas20] Jeffrey Yasskin. "Signed HTTP Exchanges." Internet-Draft draft-yasskin-http-origin-signed-responses-09, IETF Secretariat, July 2020. `http://www.ietf.org/internet-drafts/draft-yasskin-http-origin-signed-responses-09.txt`.

[Yas21] Jeffrey Yasskin. "Use Cases and Requirements for Web Packages." Internet-Draft draft-yasskin-wpack-use-cases-02, IETF Secretariat, April 2021. `https://www.ietf.org/archive/id/draft-yasskin-wpack-use-cases-02.txt`.

[YWA05] Xiaowei Yang, David Wetherall, and Thomas Anderson. "A DoS-limiting network architecture." In *ACM SIGCOMM Computer Communication Review*, volume 35, pp. 241–252. ACM, 2005.

[ZAB14] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. "Named data networking." *ACM SIGCOMM Comp. Comm. Review*, 2014.

[ZAZ17] Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. "NDNCERT: Universal Usable Trust Management for NDN." In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ICN '17, pp. 178–179, New York, NY, USA, 2017. ACM.

[ZGM20] Zhiyi Zhang, Yu Guan, Xinyu Ma, Tianyuan Yu, and Lixia Zhang. "Sovereign: User-Controlled Smart Homes." *arXiv preprint arXiv:2006.06131*, 2020.

[ZLD19] Zhiyi Zhang, Tianxiang Li, John Dellaverson, and Lixia Zhang. "RapidVFetch: Rapid Downloading of Named Data via Distributed V2V Communication." In *2019 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8. IEEE, 2019.

[ZS20] Zhiyi Zhang and Junxiao Shi. "NDNCERT Specification 0.3." https://github.com/named-data/ndncert/wiki/NDNCERT-Protocol-0.3, 2020. Accessed: 2020-06-01.

[ZVM19] Zhiyi Zhang, Vishrant Vasavada, Xinyu Ma, and Lixia Zhang. "DLedger: An IoT-Friendly Private Distributed Ledger System Based on DAG." *CoRR*, **abs/1902.09031**, 2019.

[ZVO19] Zhiyi Zhang, Vishrant Vasavada, Eric Osterweil, Lixia Zhang, et al. "Expect more from the networking: Ddos mitigation by fitt in named data networking." *arXiv preprint arXiv:1902.09033*, 2019.

[ZYR18] Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik Ramani, Alex Afanasyev, and Lixia Zhang. "NAC: Automating access control via Named Data." In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pp. 626–633. IEEE, 2018.