DISSERTATION

MEASURING NAMED DATA NETWORKS

Submitted by

Chengyu Fan

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2020

Doctoral Committee:

Advisor: Craig Partridge
Co-Advisor: Christos Papadopoulos

Shrideep Pallickara
Sangmi Pallickara
J. Rockey Luo

ABSTRACT

MEASURING NAMED DATA NETWORKS

Named Data Networking (NDN) is a promising information-centric networking (ICN) Internet architecture that addresses the content directly rather than addressing servers. NDN provides new features, such as content-centric security, stateful forwarding, and in-network caches, to better satisfy the needs of today's applications. After many years of technological research and experimentation, the community has started to explore the deployment path for NDN.

One NDN deployment challenge is measurement. Unlike IP, which has a suite of measurement approaches and tools, NDN only has a few achievements. NDN routing and forwarding are based on name prefixes that do not refer to individual endpoints. While rich NDN functionalities facilitate data distribution, they also break the traditional end-to-end probing based measurement methods. In this dissertation, we present our work to investigate NDN measurements and fill some research gaps in the field.

Our thesis of this dissertation states that **we can capture a substantial amount of useful and actionable measurements of NDN networks from end hosts**. We start by comparing IP and NDN to propose a conceptual framework for NDN measurements. We claim that NDN can be seen as a superset of IP. NDN supports similar functionalities provided by IP, but it has unique features to facilitate data retrieval. The framework helps identify that NDN lacks measurements in various aspects. This dissertation focuses on investigating the active measurements from end hosts. We present our studies in two directions to support the thesis statement.

We first present the study to leverage the similarities to replicate IP approaches in NDN networks. We show the first work to measure the NDN-DPDK forwarder, a high-speed NDN forwarder designed and implemented by the National Institute of Standards and Technology (NIST),

in a real testbed. The results demonstrate that Data payload sizes dominate the forwarding performance, and efficiently using every fragment to improve the goodput.

We then present the first work to replicate packet dispersion techniques in NDN networks. Based on the findings in the NDN-DPDK forwarder benchmark, we devise the techniques to measure interarrivals for Data packets. The results show that the techniques successfully estimate the capacity on end hosts when 1Gbps network cards are used. Our measurements also indicate the NDN-DPDK forwarder introduces variance in Data packet interarrivals. We identify the potential bottlenecks and the possible causes of the variance.

We then address the NDN specific measurements, measuring the caching state in NDN networks from end hosts. We propose a novel method to extract fingerprints for various caching decision mechanisms. Our simulation results demonstrate that the method can detect caching decisions in a few rounds. We also show that the method is not sensitive to cross-traffic and can be deployed on real topologies for caching policy detection.

ACKNOWLEDGEMENTS

I want to show my gratitude to all the people who helped me on the path towards this dissertation.

I am particularly grateful to my advisor, Prof. Craig Partridge, and my co-advisor, Prof. Christos Papadopoulos, for their invaluable guidance and constant support during my Ph.D. study at Colorado State University (CSU). Along the journey, they have always been encouraging and provided insights to help me develop research ideas. This dissertation is impossible without their patience and persistent help.

I would also like to thank my other committee members, Prof. Shrideep Pallickara, Prof. Sangmi Pallickara, and Prof. J. Rockey Luo. Their constructive suggestions and comments are valuable for improving this dissertation.

My sincere thanks to our collaborators from the NDN project team and the SANDIE project team. The designs involved in this dissertation would not have been accomplished without their contribution and hard work.

I also want to thank my colleagues at CSU, Susmit Shannigrahi, Manaf Gharaibeh, and Dimitrios Kounalakis. I benefited tremendously from the stimulating and insightful discussions with them. It is a great pleasure to have worked with them.

Thanks to all my friends for making the past few years in Fort Collins enjoyable and memorable.

Finally, I owe my utmost gratitude to my family for their unconditional love.

DEDICATION

To my beloved wife Jin who is always encouraging and supportive, my angel Rocky who gives me immense motivation and strength, and my amazing parents who love me more than themselves...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Named Data Networking (NDN) [1] is a promising instance of Information-centric networking (ICN) Internet architectures that address data directly rather than addressing servers. Unlike host-oriented IP networks, data on NDN is the thin waist. The access to data in NDN networks is by name rather than by its location. NDN provides content-centric security. Every NDN content is named and signed. All nodes in NDN networks can cache data, and thus the network can exploit the in-network storage for efficient data delivery. NDN also provides advantages such as stateful forwarding, user mobility, and others, which make it better able to satisfy the needs of today's applications.

After many years of fundamental research and experimentation, researchers gained promising results in large-data applications [3], building automation systems [4], vehicular networks [5], and IoT applications [6], and other fields. Recently, people started to deploy NDN in the real-world in some international cooperative projects [7–9].

However, deploying NDN in the real world is difficult. One challenge is network measurement. People conduct various measurements to gain insights into networks. Understanding the deployed NDN network is beneficial to operators, end-users, and researchers. As the basis of network management [10], network measurement can quantify how networks are being used and how networks are behaving [11]. Knowing the network status and behavior, operators can efficiently manage a network and diagnose performance issues [12] in the network. Network measurement also provides feedback to help network applications, services development, and new approaches to exploit network resources [13–15].

Interest in measuring networks has existed since the early days of the Internet. After many years of development, today's IP-based Internet has a set of measurement tools available in various aspects (Chapter 2). For a specific purpose, one could choose one tool from the set of available

tools to conduct measurements on the network. For example, tools for measuring available bandwidth could be pathload [16], pathrate [17], and other tools.

In contrast, NDN measurement is still at its early stage. The new name-based communication paradigm changes the way to explain network behaviors. New features, such as in-network caching and smart forwarding, not only bring benefits to data retrieval but also complicate NDN measurement. Researchers try to replicate the rich and sophisticated measurement methods of IP networks in NDN networks, but only a few NDN measurement achievements [18–21] exist.

1.1 Problems Addressed

The shift to a content-centric communication mechanism fundamentally changes the way to measure networks. The traditional end-to-end measurement methods do not fit NDN, where no address is assigned to end-hosts. Specifying the source and destination addresses in measurement packets does not work in NDN. In addition, NDN has some new features to facilitate data retrieval. Each feature introduces some complexity in measurement. The intertwined NDN features change the way we understand network performance. Those features introduce new measurement requirements.

In the rest of this section, we present the main problems and gaps we identify and address in this dissertation in relation to NDN measurement.

NDN lacks measurement framework

Previous work in NDN measurement focuses on specific applications [18–21]. However, the community does not have a clear picture of NDN measurements. NDN needs a conceptual framework that provides a guideline for measurements. The framework should give an overall picture to capture high-level measurement categories. It needs to cover existing NDN measurement achievements and provide a link between these measurement types. The classification, as well as the prior work, should be able to help identify gaps in NDN measurement.

Need to investigate the feasibility to replicate measurement approaches from IP

Researchers have investigated measurement approaches in IP networks for decades. A set of measurement tools are available to measure networks from various aspects in IP networks (Chapter 2). To gain insight into the network, users could choose one or a few tools from the available tools as their need.

NDN and IP have similarities in various aspects. They both forward packets based on the information carried within each packet. In terms of the provided functionalities, NDN is a superset of IP. NDN network layer not only supports functionality that is provided by IP but also has new features that are performed at other upper layers in IP networks.

The similarity between the two architectures makes it possible to replicate at least some IP-based measurement approaches in NDN networks. However, no prior work in NDN measurement investigated the feasibility of replicating measurement approaches from IP.

Need approaches to measure in-network state

NDN provides new features to improve data retrieval performance. In NDN, the Interest/Data packet exchange is symmetric, which induces a hop-by-hop control loop. The hop-by-hop control allows NDN routers to use historical statistics to make smarter forwarding decisions. NDN routers also can follow the caching rules to save received Data packets in its Content Store (CS) and use them to satisfy future requests.

The in-network state is critical to data retrieval performance. Forwarding Interest to the "best" producer could result in better performance, while choosing an incorrect outgoing interface may introduce delays or even packet losses. Retrieving data from a cache on the path is much faster than that from the original producer. However, suboptimal caching decisions may lead to the inefficient use of cache resources. Moreover, multiple forwarding strategies and caching policies may exist in NDN networks, and they may interact poorly. Measuring the in-network state is essential to troubleshoot the above performance issues. Nevertheless, no measurement approach exists until this thesis.

Need approaches to measure networks at end-hosts

The NDN community needs a set of measurement approaches to help understand networks in various aspects. IP networks use both passive measurement and active measurement to fulfill the measurement purpose.

Passive measurement techniques, such as NetFlow in IP networks, are typically used to monitor enterprise networks [22]. However, monitoring functions on routers requires special protocols and access privileges to monitoring nodes. Inferring the state of one connection from a large number of connections is not straight-forward.

On the contrary, active measurements at end-hosts do not need any management protocols or access privileges to any intermediate nodes. They inject traffic into the network and collect statistics locally on end hosts to infer in-network states.

NDN also needs both passive measurement and active measurement to diagnose network issues. The passive techniques in NDN networks appear similar to IP networks, but the active measurement is challenging in NDN networks. Both forwarding strategies and caching policies introduce in-network states that are not preserved in IP networks. The injected traffic may change the in-network states and cause a dramatic change in network behaviors. Developing an active measurement approach while minimizing the overhead is potentially a difficult problem.

1.2 Thesis Statement

This dissertation states that **we can capture a substantial amount of useful and actionable measurements of NDN networks from end hosts**. In this statement, the measurement from end hosts indicates that we do not need any management protocols, the help from the third party or special access privilege to intermediate nodes. The measurements can capture the information inside networks to help identify network issues or detect network states. The challenge is to minimize the overhead in measurements. NDN routers contain some functionalities, such as caching policies, to adapt to changing network conditions. These functionalities are stateful and sensitive to network traffic. A large amount of traffic may introduce congestions or delays, which could change

the network states. The forwarding strategy may adapt to the change and stick to another "idle" interface. The excessive traffic may occupy the precious cache spaces and force routers to evict cached copies. We minimize the overhead by using a small number of probing packets for NDN measurements. The statistics collected on end hosts help the tools infer network states.

We aim to gain useful measurements about the NDN network states from end hosts. The state in NDN networks could be the topology, routing, and configurations. We present two studies to support our thesis statement. In the first supporting study, we present an devised packet dispersion method to estimate the capacity between the consumer and the producer. In the second study, we present an efficient delay-based method to detect the deployed caching decisions mechanisms. We show how we can use a small number of probing packets to disclose useful information about network state. To the best of our knowledge, those measurement methods are not investigated in NDN networks in prior work.

1.3 Overview of the Dissertation

The previous section presented some open problems and gaps in the NDN measurement field. In this section, we present an overview of studies in this dissertation to address these problems and how they support the statement in Section 1.2.

1.3.1 NDN measurement framework (Chapter 3)

NDN adopts the content-centric communication mechanism, which fundamentally changes network measurements. IP and NDN have many differences, which lead to different measurement requirements. The way to explain measurements in NDN networks also differs from that in IP networks. Previous work [18–21] in NDN measurement addresses issues in specific applications. This dissertation presents a conceptual framework for NDN measurement, aiming to provide a guideline for NDN measurement research.

We start with comparing IP and NDN, as today's Internet is IP-based, and the IP measurement framework is well-established. Comparing the layers that provide similar functionalities, we show

that NDN contains rich functionalities that were performed at other upper layers in IP networks. We can roughly think of NDN as a superset of IP. Based on the observations, we present a conceptual NDN measurement framework that covers needed NDN measurements in three categories. People could use the framework to identify the gaps in NDN measurement.

Moreover, we list available measurement tools as well as their measurement targets in both IP and NDN networks. We show that gaps exist in various types of measurement types. Our work in this dissertation focuses on the measurement from end hosts.

1.3.2 Replicating IP measurements in NDN networks

Since NDN can be seen as a superset of IP, and IP networks have a set of well-designed measurement tools, we could replicate IP measurement approaches in NDN networks to fulfill some measurement requirements. In this section, we present an overview of our work to explore the feasibility of replicate IP measurements in NDN networks and how they demonstrate the thesis statement.

Performance benchmark for a high-performance NDN forwarder (Chapter 4)

Before replicating the IP measurement approach (i.e., packet dispersion techniques in this dissertation), we study the relationship between packet parameters and NDN performance using an NDN forwarder, NDN-DPDK [23]. We choose NDN-DPDK for reasons. NDN-DPDK is a high-performance router built on the Data Plane Development Kit (DPDK) [24], which allows NDN packets to be directly encapsulated in Ethernet frames. Since fewer modules are involved in packet forwarding, our analysis could focus on these modules, and the goodput could be better as well.

Unlike the prior work [23] to study the NDN-DPDK router's throughput, the purpose of this work is to benchmark the data retrieval performance under a specific name prefix using the NDN-DPDK forwarder. We use the NDN-DPDK forwarder to build a dedicated NDN testbed, and then use the provided ndnping-dpdk tool for the benchmark. We manipulate both Interest and Data parameters to generate traffic. The reported performance results, along with the associated parameters, are used to evaluate the effect of parameters on performance.

Specifically, we investigate the effect of name components and Data payload sizes on various performance metrics. Our results show that both name component length and Data payload size affect data retrieval performance. Longer name components decrease forwarding performance, as they require more iterations during table lookups. In general, larger Data payload sizes generate better goodput and throughput, but hop-by-hop fragmentation may hurt the performance. The results indicate that applications must carefully choose the Data payload size to utilize the hop-by-hop fragmentation for better NDN performance efficiently.

Packet dispersion techniques in NDN networks (Chapter 5)

We choose to replicate packet dispersion techniques (Section 2.7) in NDN networks. The packet dispersion techniques are used to estimate the end-to-end capacity of a path in IP networks by measuring the interarrivals of packets. When an NDN network adopts the best-route forwarding strategy, the communication between one consumer and one producer is the same as the end-to-end communication in IP networks. The end-to-end capacity estimation techniques are useful to estimate the capacity under a name prefix. Moreover, the packet dispersion techniques introduce small overhead, which makes it a good fit for NDN measurements.

We conduct experiments on the same dedicated testbed in Chapter 4. Since NDN uses hop-by-hop fragmentation, and the data processing pipeline dominates the performance, we devise the packet dispersion techniques to use the interarrivals of Data packets. The testbed machines contain two types of network cards, which allow us to investigate the feasibility of replicating packet dispersion techniques on both of them. To allow probe packets to detect potential interrupt coalescence (IC) (Section 2.7.3), we use a sequence of consecutive Interest packets to generate a sequence of Data packets. Our experiment results over the common NICs demonstrate that the NDN packet train can detect IC. In the presence of IC, we can estimate the capacity based on the interarrivals between successive Data packet bursts. The above work demonstrates that estimating the capacity of an end-to-end path from end hosts is feasible.

We also investigate the results of high-speed NICs. The results from end hosts indicate that there are issues in applying packet dispersion techniques in NDN networks over high-speed NICs.

We then profile the NDN-DPDK forwarder to identify potential performance bottlenecks in its design. After checking the interarrivals on the forwarder, we show that the variance of packet interarrivals increases when Data payload sizes become larger. We argue that the potential issues may be the scheduling overhead in the operating system and the use of hop-by-hop fragmentation in NDN networks.

1.3.3 Detecting NDN-specific network state (Chapter 6)

Caching is critical to data retrieval performance in NDN networks [1]. Various caching policies may exist in the same network, and they may interact poorly. Misconfiguration of caching policies may introduce performance degradation. Detecting active caching policies is useful for managing NDN networks and improving network performance.

A caching policy contains the caching decision mechanism and the cache replacement mechanism. In this work, we focus on detecting caching decisions. Specifically, we propose a novel method to build the fingerprints of caching decisions. To start, the end host sends out a number of Interests, and each Interest carries a unique name. Upon the arrival of Data chunks, the end host saves the delays. After repeating the above steps for ten times, we plot the results in figures. We use two metrics to identify a caching decision mechanism uniquely. The first metric is the hop count distribution of Data packets in one round. The second metrics is the changes of the hop count distribution across multiple rounds.

Our simulation results demonstrate that the method detects various caching decisions successfully. Estimating the probability value in static probabilistic caching decisions is also possible. We also study the effect of cross traffic. The results show that cross traffic does not change the method's correctness.

We realize that the NDN stack does not give the hop count information to applications. To address this issue, we propose to use clustering algorithms to map delays to hop counts. Experiment results show that the mapping could give us the correct fingerprint of caching decisions.

1.4 Research Contributions

This dissertation presents three studies to address some research gaps in NDN measurement and to support the thesis statement. In this section, we list these studies and their contributions.

The first study proposes the first conceptual NDN measurement framework (Chapter 3). We compare IP and NDN to point out the similarities and differences. Then we propose the conceptual framework to help classify existing measurement achievements. The work also establishes a connection between IP and NDN. The framework includes the needed measurement categories and existing measurement work in each category. We envision the framework can help researchers to identify gaps in NDN measurements. Specifically, in our case, we point out the need to measure the NDN network from end hosts.

Our second study studies the feasibility of replicate IP measurements in NDN networks. We present the first performance study for NDN-DPDK on a dedicated testbed (Chapter 4). Different from the prior work [23], we focus on the data retrieval performance under a name prefix, not the throughput of the NDN-DPDK forwarder. We show the effect of NDN packet parameters on performance on the testbed. We point out that carefully choosing Data payload size could utilize every fragment efficiently, which is critical to get better goodput for applications. Using the results in Chapter 4 as the baseline, we present the first work to investigate the feasibility of replicating packet dispersion techniques (Chapter 5). We demonstrate that the packet dispersion techniques can estimate the capacity when 1Gbps NICs are used. Our measurements from end hosts also show that the techniques do not apply to NDN-DPDK forwarder with 40Gbps NICs. After investigating the interarrivals on the forwarder and analyzing the forwarder code, we point out both the scheduling and hop-by-hop fragmentation introduce variance. We also profile the NDN-DPDK forwarder to identify potential bottlenecks in the design.

The final study proposes a new approach to detect the deployed caching decision mechanisms at end hosts. To the best of our knowledge, this is the first work to detect caching configurations in NDN networks. We show that we can detect the active caching decisions for a name prefix using the hop count distribution in one round and the changes across multiple rounds. Our results also

show that the proposed method is robust, and cross traffic does not change its correctness. We are aware of the issue that the NDN stack does not expose the hop count information to applications. In our simulation using a real topology, we demonstrate that mapping delay to hop counts can address the above issue.

Chapter 2

Background and Related Work

To better serve users, computer networks are constantly growing in both size and complexity. Managing such large and complex networks becomes more and more challenging. Due to various reasons, the Internet serves as a black-box to hide problems inside. When users experience excessively slow downloads on the Internet, the causes may be the congested paths, mistuned TCP parameters at the end-hosts, or routing changes. Understanding how networks are being used and why they behave the way they do [10, 11] helps operators to manage networks efficiently. Network measurement is the way to gain useful information about the underlying network.

Each network architecture has distinct measurement requirements. Named Data Networking (NDN) introduces new features in the network layer to facilitate data distribution. The rich in-network features lead to complexity in measurement. Understanding the NDN mechanisms helps outline the requirements of NDN measurements. This chapter describes the NDN architecture in section 2.1 and explains NDN features with specific examples as the background of this dissertation. For instance, we introduce the caching policies in section 2.3, as the background knowledge for caching policy detection in Chapter 6.

IP networks have been deployed to serve people for decades. After years of development, a set of IP measurement approaches and tools are available to help users and operators in various purposes. Revisiting these approaches could inspire new measurement ideas in NDN networks. To this end, we introduce the taxonomy of IP measurement approaches in section 2.6. Since we think NDN as a superset of IP, we believe that some existing IP measurement approaches have the potential to be used in NDN networks. We choose packet dispersion techniques as a specific example in end-to-end path capacity measurement. We explain how it works in section 2.7.

2.1 Named Data Networking

This section briefly summarizes the NDN architecture. First, we explain what is NDN and why it was developed. Then we introduce its features using examples. A more detailed description can be found elsewhere [1, 25, 26].

2.1.1 What is NDN

Today's Internet architecture advocates the hourglass model where the thin waist, IP layer, implements the global interconnectivity [1]. The IP layer allows applications to run on top of it without the need to address the interaction with lower-layer technology. The simplicity of the IP layer provides a friendly environment for the emergence of tremendous Internet applications. After decades of development, more and more applications, such as digital media, file sharing, social networking, and mobile applications, has been introduced. The sustained growth in these IP applications has changed the Internet's task from supporting communication to distributing content.

The IP-based Internet is not designed to support content distribution. It supports end-to-end communications by carrying endpoints' addresses in packets. This host-centric semantic leads to the incompatibility between the networking model and the application requirements: applications request data by names, but network names endpoints by IP addresses. For example, to ensure content availability, people use awkward, pre-planned, application-specific mechanisms like content delivery networks (CDN) [27, 28] and P2P networks [29, 30]. Many applications rely on complex host-based mechanisms to secure connections [31, 32], but content security issues still exist. Additionally, mapping content to host locations complicates network configurations as well as the implementation of network services. In this context, researchers proposed Information-centric networking (ICN) to shift from the traditional host-centric networking model to a novel content-centric paradigm.

In this dissertation, we focus on Named Data Networking (NDN), a promising instance of ICN architectures. Comparing with other ICN architectures, NDN stands out in several aspects. First,

NDN provides reasonably mature content-based communication mechanisms, and researchers have built various applications [19, 33–36] over it. Second, the NDN team maintains an open-source code-base [37]. Third, NDN is one of four projects selected by the NSF Future Internet Architectures (FIA) program [38].

2.1.2 NDN Architecture

Similar to IP networks, NDN retains the same hourglass model (Figure 2.1). The major difference is that NDN changes the centerpiece of the network stack from IP to named data. IP's hourglass model help the upper and lower layer technologies evolve without changing the universal packet forwarding mechanism. The named data can run over any transport technology, including IP and others. Using data names instead of IP addresses for delivery offers a new set of minimal functionality based on content names.

NDN way better satisfies the requirements of today's applications. Today's applications are typically written for retrieving data. On IP networks that address hosts, those applications themselves or specific middleware need to map the required data to the host-based model. NDN changes the semantics of network service from delivering packets to a destination to retrieving content. Therefore, applications can focus on data retrieving, without implementing the functionality of mapping data to hosts. NDN introduces two new layers: the strategy layer and the security layer. The strategy layer allows the network to adapt to the changing network conditions. The security layer secures the content itself rather than the communication channels as in IP. With the help of those new features, NDN can eliminate the need for complex middleware and the need for applications to perform common network functions.

2.1.3 NDN Packet Types

On NDN networks, data is accessed by name rather than through the host where it resides. To achieve this, NDN uses two types of packets for data transport - **Interest** and **Data** (Figure 2.2). Both types of packets carry a name. The name uniquely identifies a piece of data on NDN networks.

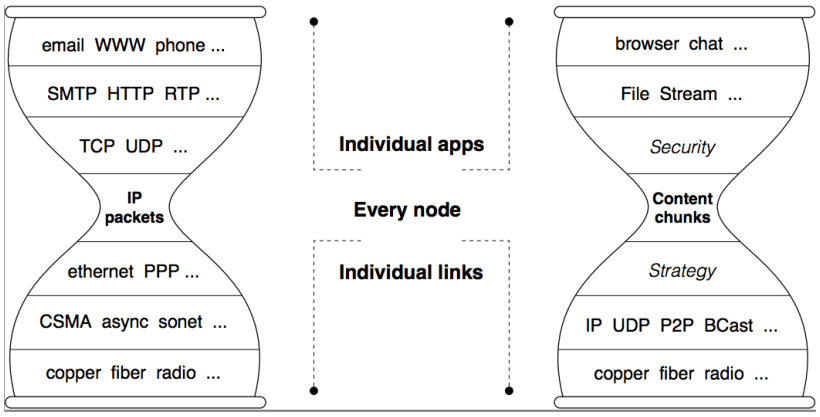


Figure 2.1: Internet and NDN Hourglass Architectures [1]

When a publisher makes a piece of content public, a unique name is assigned to it. If a data receiver (a.k.a, data consumer) wants to fetch this content, he/she sends an Interest to the network. Routers use the carried name (or name prefix) to forward the Interest packet towards the data producer. A Data packet is returned to the consumer only when the Data packet name matches the name in the Interest.

Both Interest and Data contain multiple fields to facilitate data retrieval. A Nonce must be included in the Interest to help NDN to detect loops. Other elements are optional. The selector field in an Interest could specify the restrictions on the returned data when more than one Data packet satisfies an Interest. Guilder fields affect Interest forwarding behavior. For instance, the InterestLifetime indicates the (approximate) time remaining before the Interest times out.

A data packet carries the name, actual data, metainfo, and signature that binds the data to the name. The signature field allows consumers to verify if Data packets are originally from a valid producer. The metainfo field contains descriptions about the data. They help data retrieval in various ways. For instance, the freshness field helps node decide when to mark a cached Data packet as stale. The content type allows applications to choose corresponding modules to handle incoming Data packets.

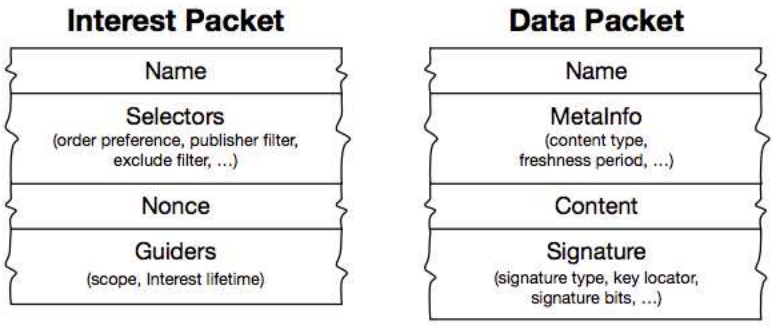


Figure 2.2: Packets in the NDN Architecture. [1]

2.1.4 Names

Names are carried in both Interest and Data packet. Routers do not understand the meaning of the names. The name opacity allows applications to define their own naming schemes independent from the network.

Similar to URLs, NDN names are hierarchical. The hierarchical structure has several benefits. First, applications could use the hierarchy to represent the relations between data pieces. If this dissertation is published on NDN networks, it may have the hierarchical name "/ndn/colostate/cs/student/chengyu/dissertation.pdf", where "/" is the delimiter between name components. Given the name, people could understand this is the dissertation of student Chengyu at the computer science department of Colorado State University (CSU). NDN names conceptually contain two variable parts, the routing prefix and the application-specific portion. Similar to IP routings, NDN routing protocols can propagate the routing prefix "/ndn/colostate" to ensure CSU's reachability. Hierarchical names are natural to aggregate so that the name-based routing system could scale. Furthermore, specifying the scope and contexts in the hierarchical names is much easier. Specifically, a trust model [39] could specify the scope to help authentication.

2.1.5 Data-Centric Security

In contrast to TCP/IP that secures the communication channels, NDN secures the content itself. To secure the content "/ndn/colostate/cs/student/chengyu/dissertation.pdf", the producer can generate the signature using his signing key "/ndn/colostate/cs/student/chengyu/signing-key". The

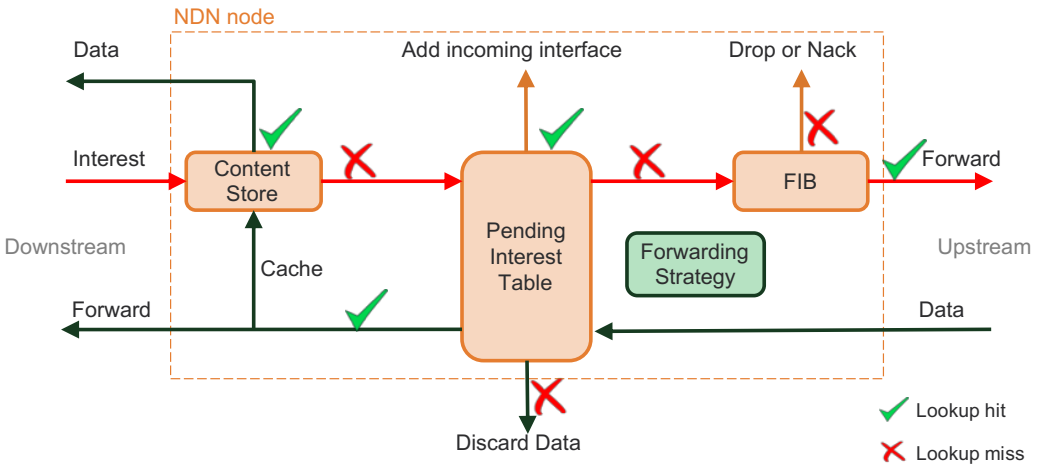


Figure 2.3: Forwarding Process at an NDN Node. [1]

signature binds the content producer, content name, and content data together, enabling determination of data provenance and integrity. Upon the arrival of the Data packets, a consumer can verify the packet's authenticity and integrity regardless of how or from where the packet is retrieved. This feature facilitates data distribution as well. Since consumers do not care where the packet is retrieved, NDN networks could cache Data packets on intermediate routers. When another consumer sends out an Interest with the same name, the Interest will be satisified by the cached Data packet, without the need to reach the producer.

Moreover, NDN allows consumers to choose or customize a trust model [39]. The trust model specifies whether a public key owner is an acceptable producer for a packet or content in a specific context. For the NDN content "/ndn/colostate/cs/student/chengyu/dissertation.pdf", a trust model could require the publisher's key must have the same prefix "/ndn/colostate/cs/student/chengyu/".

2.1.6 Packet Handling

The name-based packet handling needs the help of three major data structures on each NDN router (Figure 2.3): a Forwarding Information Base (FIB), a Pending Interest Table (PIT) and a Content Store (CS).

- FIB: a forwarding table used for instructing packet forwarding. The FIB on an NDN router is similar to the FIB on an IP router except that it contains name prefixes instead of IP address

prefixes. Moreover, an NDN FIB maintains the list of potential outgoing interfaces for a specific name prefix.

- PIT: a table saves all Interest packets that are not yet satisfied. PIT entries contain not only the Interest name but also the incoming interfaces of this Interest. Routers then could use those incoming interfaces to forward the returning Data packets to the consumers.

- CS: a structure that temporarily holds data packets that pass through this router. Since NDN Data packets contain all the information to authenticate the provenance and integrity, Interests do not need to reach the original producer for Data. Routers can utilize the storage space to satisify future Interests.

Before forwarding packets, an NDN router must contain forwarding rules in its FIB. For instance, an FIB entry "/ndn/colostate/cs" is added to ensure the reachability of the computer science department at CSU. Routing protocols propagate the forwarding information to the rest of the network. FIB aggregation may happen to make the routing table scalable. In the computer science department, the user's name prefix, "/ndn/colostate/cs/student/chengyu/", may be registered for publishing data.

NDN communication is driven by consumers. The Interest packet sent by consumers contains the name or prefix of the desired data. In our example, the Interest needs to include the name "/ndn/colostate/cs/student/chengyu/dissertation.pdf". Upon receiving the incoming Interest packet, an NDN router first checks whether the local CS contains a matching Data packet. If it is found, the corresponding data packet is forwarded back to the same interface where the Interest packet arrived. Failure to find a matching Data packet triggers a PIT search. If a matching entry is found but the arriving interface is not listed, the new interface is added to the same entry. This design removes the duplicate load on network nodes and servers when consumers request the same content. The nonce in Interest packets prevents the forwarding loops. When no matching PIT entry is found, a new PIT entry is created. After that, the FIB will instruct how to forward this Interest based on longest name prefix matching (LNPM) results. Specificaly, the FIB takes the

requested content name "/ndn/colostate/cs/student/chengyu/dissertation.pdf" as the lookup key to find the LNMP result. The name's hierarchical structure helps the FIB identify the LNMP entry "/ndn/colostate/cs" on that router and use the outgoing interface to forward Interests.

When a Data packet that contains name "/ndn/colostate/cs/student/chengyu/dissertation.pdf" arrives, an NDN router first checks if the local PIT contains the carried content name. If there is no PIT entry found, the Data packet is discarded. If it is found, the caching policy on that router decides if it saves the Data packet in the CS. Then the router forwards this Data packet to all the listed interfaces and removes the corresponding PIT entry. This design ensures NDN routers forward data along the reverse path and multicast the data without requiring the consumer to acquire a network address.

2.1.7 Intelligent Forwarding Strategy

NDN has a unique feature called the forwarding strategy that can intelligently use available network resources for data retrieval. Over each interface, every out-going Interest packet pulls back exactly one Data packet, except in cases where packets get lost. Symmetric traffic enables intelligence. For every pending Interest packet, an NDN router maintains an entry in its PIT. The maintained state allows the router to monitor the two-way traffic and be aware of packet arrivals/losses and network condition changes. With this closed-loop formed by the two-way symmetric packet flow, routers can adapt to the network changes based on those observations.

The intelligent forwarding strategy is one of the key features that facilitate data distribution. Unlike IP networks where applications must set up another channel to resume the broken data retrieval, NDN routers could switch over to another available link. For instance, a strategy can balance the load among multiple servers using round-robin Interest forwarding in the multi-producer scenario. In our example of retrieving this dissertation, routers could maintain multiple retrieval paths and rank them according to performance. When the performance over one path becomes low, the strategy could switch over to another path. The switch-over is much faster, as the data plane has direct knowledge about network conditions.

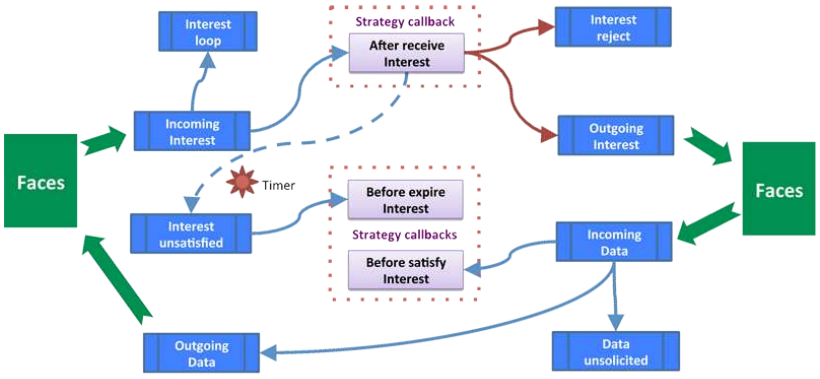


Figure 2.4: NDN forwarding strategy adapts to network states. [2]

To allow applications to choose appropriate forwarding strategies, NDN makes the strategy modules pluggable. A forwarding strategy could leverage callback functions to process Interest and Data packets. Various parameters could be used in forwarding strategies, including values in a measurement table, current network condition, or pre-defined rules. Figure 2.4 demonstrates how strategies utilize in-network states to support intelligent data retrieval. Network operators can configure static or dynamic per name-prefix strategies, and thus various strategies could co-exist in NDN networks.

To sum up, the NDN design focuses on data distribution. The NDN network layer contains more features than the IP network layer. Although those features improve data retrieval performance, they complicate measurements. We further discuss the complexity in Chapter 3.

2.1.8 Hop-by-hop Fragmentation

The Internet is made of many heterogeneous networks. Since the appearance of the Internet, more and more computing devices, including mobile phones, vehicles, and other IoT devices, connected to the network. Consequently, various maximum transmission unit (MTU) sizes may exist at link-layer across different links. To this end, networks must fragment packets to fit the MTU.

IPv4 [40] solves this issue by letting each hop (re)fragments the packet as needed to fit the next MTU size. The final destination reassembles the fragments. IPv6 uses a different method to handle

the heterogeneity. The source discovers the smallest MTU over the path to the destination and then fragments data from higher layers to fit the MTU [41].

However, the concept of path MTU, like IP, does not apply to NDN. As a new Internet architecture, NDN focuses on data and data retrieval. To retrieve the desired data, a consumer needs to express an Interest, and the network will find and return the requested data. These Interests are processed by intermediate routers. Intermediate routers decide whether the Interests can be satisfied with cached data, or they should be aggregated with similar interests from another consumer, or forwarder upstream according to the FIB. Upon the receipt of Data packets, routers try to match them against pending interests. This match is based on the data name. In the above communication model, each router along the path must have fully assembled Interest and Data packets for making forwarding decisions. Moreover, data is not necessarily retrieved from the original producer with the help of in-network caching. Any future consumers could retrieve data from an in-network cache. Therefore, producers cannot pre-size data packets to avoid fragmentation, as the path to retrieve data may change from one time to another. To sum up, the traditional "end-to-end path" concept and the "path MTU" concept no longer apply to NDN networks.

To accommodate fragmentation, NDN adopts Hop-by-hop Fragmentation and Reassembly (HBHFR). Alexander [42] pointed out that HBHFR not only ensures the correct delivery of Interest and Data packets but also brings a number of advantages. First, HBHFR can reduce packet delivery failures. The traditional fragmentation requires end-hosts to retransmit packets when losses happen. HBHFR has the potential to recover from lost or corrupted fragments locally. Additionally, HBHFR opens the possibility to utilize the underlying advanced link-layer networking technologies efficiently. Since HBHFR can handles MTU constraints on the data retrieval path, applications could use larger MTU sizes without worrying about the details. One should also notice that using a large packet size can increase the store-and-forward delay across every hop. Alexander [42] claim that it depends on the network speed to determine if a noticeable store-and-forward delay would happen. We show the effect of hop-by-hop fragmentation on network performance in chapter 4.

2.2 Publish/Subscribe Systems

The publish/subscribe system is widely-used for propagating data from publishers to interested subscribers in IP networks [43–47]. In publish/subscribe systems, messages are not directly exchanged between publishers and subscribers. If subscribers want to fetch data in particular topics, they can register their interest to a logical intermediary, the event dispatcher. Upon the arrival of an event, the dispatcher follows the registered event patterns to forward the event to the matching subscribers.

Leveraging the event dispatcher, publish/subscribe systems decouple publishers and subscribers. Both publishers and subscribers communicate only with the event dispatcher [48]. Publishers do not need to know how the event will be distributed or to whom. Subscribers do not need to have the knowledge of where the event came from or who produced it.

Although the publish/subscribe systems is similar to NDN, caching as a mechanism for storing data in publish/subscribe systems has not received attention until the work published by Vasilis et al. [49, 50]. They assume that some nodes have a limited capacity cache that decides upon a policy for caching and prioritizing messages. They consider some basic caching policies, such as caching in all event dispatchers for yielding high survivability and caching in leaf dispatchers for maintaining low overhead and querying complexity [49]. The focus is on the performance of the system regarding information survivability, delay, overhead, search complexity, and scalability. To the best of our knowledge, no work is proposed to detect what caching policy is deployed on publish/subscribe systems.

Caching in NDN exhibits fundamental differences from the traditional caching schemes and poses new challenges [51]. NDN assumes that every NDN node can be equipped with the cache. Each NDN node can make cache decisions and replace cached items at line-speed. Researchers have proposed various caching policies to improve network performance (see Section 2.3), which also introduces potential configuration issues. Chapter 6 presents the work to detect caching policies in NDN networks.

2.3 NDN Caching Policy

Since Jacobson et al. proposed Content-Centric Networking (CCN) [25], the research community reached the consensus that the naive caching policy is not the most optimal. Researchers have shown that more advanced caching policies can utilize caches more effectively [52–56], but there is no consensus on which caching policy is optimal. In each application field, there may be a customized caching policy deployed to help data distribution. Typically, NDN caching policies contain two aspects: the caching decision and the cache replacement decision. Some existing approaches focus on only one of these aspects, but others take both into account. This subsection introduces the existing caching policies as Pfender et al. outlined [57, 58].

2.3.1 Caching Decision

The caching decision is to determine whether the packet will be stored in the CS or discarded. Typically, an NDN node needs to make such a decision when a Data packet that the node does not save in its CS arrives at the node.

Cache Everything Everywhere (CEE)

CEE is the most straightforward caching decision strategy. When CEE is using, NDN nodes attempt to cache every incoming Data packet that is not already in their CS. CEE provides the rapid replication of all available Data packets through the network. Due to its simplicity, CEE strategy is widely used in traditional NDN networks.

However, simplicity leads to high redundancy, especially in a strongly connected network [58]. The redundancy could hurt caching efficiency. If every router stores the same Data packets, we are not using the network's caching capabilities to its full extent. In a network with severely limited caching capabilities, CEE will waste the precious network cache resources.

To this end, researchers propose new strategies to reduce overall redundancy by ensuring that different router saves different Data packets. That leads to the definition of diversity [58], which is the ratio of unique content objects in all caches to unique content producers. Other advanced caching strategies attempt to maximize this metric.

Leave Copy Down (LCD)

NDN routers with the LCD caching policy always cache Data packets only at the first node to receive the Data [57]. The so-called "first node" is the next hop from the node where the cache hit occurred. When data become more and more popular, the saved Data chunks will gradually move towards downstream as LCD caches responded Data chunks one hop closer to the requesting consumer.

Generally, LCD is conservative as it tends to keep Data packets close to the producer in the retrieval process. No matter where it happens, the saved copy could alleviate the load on the producer as long as it stays on the path. LCD is chosen when we do not want to cache everything at every node, but also do not want to introduce complex additional steps into the caching process.

Static Probabilistic Caching

Caching content with high diversity is essential to improving the cache hit rate. The easiest solution to decrease redundancy is to apply a probability when a router needs to decide whether it should cache the given Data chunk. The probability that an NDN router will store the incoming data chunk could be decided before the router is running. Upon the arrival of a Data chunk, a router generates a random number between 0 and 1. If the generated number is smaller than the pre-set probability p, the chunk is saved in the CS on that router. Otherwise, this chunk is forwarded downstream toward the consumer without being cached.

Since not every Data chunk is cached at every router, the cache diversity with probabilistic caching across the network will definitely be higher than that with CEE. Some Data chunks are requested and sent more often than others, and we call these chunks "popular" Data chunks. Static probabilistic caching could be aware of "popular" Data chunks. Since each incoming Data chunk triggers the process of caching decision, "popular" Data chunks have a higher chance of being stored at more points in the network. Let n be the number of times that a node receives the same Data chunk, the caching probability at that node is $1 - (1 - p)^n$ [59].

It is obvious that the value of probability p has an essential effect on the diversity and redundancy cross the network's caches. A higher p will result in higher cache redundancy, and CEE

could be treated as a special case of probabilistic caching with $p = 1$. Previous studies have found that a lower p correlates with better performance, but too low a value for p may result in under-utilization of available resources. Researchers have not determined the lower bound for p [58]. The most commonly chosen value is $p = 0.5$ [60]. However, a wide range of values has been considered [61], such as 0.7, 0.5, 0.3, and other values.

Dynamic Probabilistic Caching

Some caching strategies adopt dynamic probability to better adapt the caching behavior to the network state. In these strategies, each router computes a caching probability individually for itself or even for each Data chunk, based on available information. The information could be node-local information, Data chunk information, or even the information from the wider network. For example, local router ID, Data chunks properties (age, type, or producer), the position of the caching node in the network topology, and other information could be used for caching strategies to determine the probability.

ProbCache [53] is a dynamic probabilistic caching strategy that computes the caching probability of a given Data chunk based on the position of the caching node. The position is decided by the total number of hops between its producer and the consumer that requested it, as well as the number of hops remaining on the path to the consumer [57]. Specifically, ProbCache extends the Data packet with two new fields, Time Since Birth (TSB) and Time Since Inception (TSI). TSB counts the number of hops since the creation ("birth") of the Data packet. TSB's initial value is 1. Before forwarding the Data packet, a router increments the value. TSI is defined as the number of hops between the creation of the corresponding Interest packet and the cache hit. ProbCache calculates the cache weight as TSB/TSI. Therefore, the network edges would have a higher probability of storing Data chunks. One should notice that the caching probability is re-calculated each time. That is because the TSI is not the hop counts between the consumer and producer when a Data chunk is cached.

Some researchers believe that caching closer to the server could be more beneficial in some topologies [57]. ProbCache has a variant, called ProbCache-inv, to store more data chunks at the producer side. The caching probability is simply inverted to $rand() < (1 - CacheWeight))$.

Other Strategies

Many other caching decision strategies consider various types of information in their mechanisms. In the following, we will briefly cover a number of alternative strategies. However, due to their complexity and overhead, we do not investigate them in this dissertation.

Cooperative caching takes more than local information into account. The way to cooperate could be implicit or explicit. Implicit coordination strategies set up prior rules on nodes to make the decision for Data chunks. Routers follow the rules to make decisions, avoiding the need for explicit coordination. The label-caching decision proposed by Li et al. [62] is an example. This strategy eliminates the overhead of coordination between nodes by pre-assigning each router a fixed label $l < k$ at setup. Routers are only allowed to cache chunks whose segment ID modulo k is equal to l. This design evenly distributes chunks across the network. In practice, users could adjust k to stratify chunks into equal subsets automatically. Explicit cache coordination strategies tend to be more complex than their implicit counterparts. In explicit coordination, routers require more communication among the nodes and more calculations to maintain a consistent state [57]. The communication should contain information about cache states or cached chunks in some manners to help make decisions. An example is a strategy proposed by Liu et al. [63]. The strategy coordinates between nodes to construct a virtual backbone network with core nodes responsible for caching.

Other information can be used in cache decisions. Vural et al. [64] propose a strategy that uses the popularity to calculate the cost function for the caching of incoming chunks. Chai et al. [65] utilize the topology of the network to save chunks at the most central node. All caching strategies claim to generate reasonable good performance, but there is no consensus on what information a good strategy should have.

2.3.2 Cache Replacement Decision

The cache replacement decision is another component of caching policies. Routers need to evict chunks whenever a content chunk is to be stored in a CS that has reached its capacity. When the CS capacity is reached, the cache replacement policy needs to determine how to choose chunks that need to replace. Caching decision is irrevocable. Once the caching decision has been made, the new chunk is guaranteed to be stored, and the chosen chunk has to be evicted from the local CS.

Some commonly used cache replacement decisions are Least Recently Used (LRU), Least Frequently Used (LFU), and Random Replacement (RR). In LRU, each node keeps track of the Interests it serves and when it served them. Whenever the CS reaches its capacity, the router always evicts the data chunk that was least recently requested. LRU aims to keep the freshest and most popular chunks in the CS so that removing unpopular or outdated chunks is naturally adopted. In practice, this approach has been shown to be effective [58]. LFU, as a variation on LRU, keeps track of how often the cached chunks are requested. The least popular chunk is chosen when eviction occurs. It was reported that LFU generates better cache diversity than LRU [52]. However, LFU tends to overly favor items that experience spikes in Interest frequencies, which will eventually result in higher server load [58]. This is because caches accumulate stale items instead of storing fresher content, and thus more Interests are routed to the original producers [61]. As the least complex of the basic cache replacement policies, RR evicts random content chunks whenever the replacement occurs. It tends to be used more as a benchmark for evaluating other replacement policies than as an actual policy.

Detecting deployed caching policies is critical to understanding the retrieval performance. Misconfiguration of caching policies may lead to performance degradation. Comparing to cache replacement, caching decision decides the efficiency of using network caches. In this dissertation, we propose a novel caching decision detection approach in Chapter 6.

2.4 Existing NDN Measurement Tools

Network measurement in NDN is an open research area. As a new networking architecture, NDN changes the communication model from host-centric to content-centric. It also introduces new features to improve data transfer. Those changes add new use cases and complicate network measurements. Currently, only a few achievements are available [18–21]. This section gives a brief description of those tools.

ndnping [18] is a reachability testing tool for NDN. It is used to test the reachability between two nodes as IP-based ping does. The ping server is listening on a name prefix. Upon the arrival of an Interest that carries the name prefix, the server sends Data in response. The client then reports the RTT for the Interest-Data exchange.

Contrace (also called CCNinfo) [66, 67], NDN-trace [19] proposed the path tracing tools to determine the characteristics of available paths to a given name prefix in NDN networks. While implemented on different platforms, both of them deploy an agent on each network node to help measurements. Contrace requires PIT entries created by trace requests to be kept alive until a configured timeout expires. NDN-Trace follows the "flow balance" principle. The NDN-Trace agent that is deployed on each NDN node attaches local parameters to the trace requests sent by clients. This whole process is repeated at every node along the path(s) until the request either reaches the target prefix or times out. Then a trace reply is an NDN Data packet sent by an NDN-Trace agent. Upon receiving a trace Data packet, the agent generates a Data packet that contains timing information for the local node. Eventually, the client will receive a reply containing all the information generated by each hop along the path (or paths).

Marchal et al. [21] investigated the server-side performance for generating NDN Data. The proposed tool, NDNPerf, focuses on the server-side performance evaluation. It could help users have an idea of the throughput a server can achieve when it has to generate and transmit NDN Data packets.

NDN measurement is still at its initial stage. Only a few measurement achievements exist in NDN networks. Without a sufficient number of measurement approaches or tools, operators would

have difficulties in understanding the network, and researchers would have trouble to conduct their experiments. Fortunately, measurements in IP networks are well-established. We may learn from IP measurements and figure out the similarities and differences between IP and NDN. To this end, we describe the taxonomy of IP measurements later in this chapter. Leveraging the IP measurement framework, we outline some required measurements that are useful to NDN networks in the next chapter.

This dissertation focus on approaches that users could measure network state from end hosts. Specifically, we investigate the feasibility of using packet dispersion techniques, an IP measurement approach for estimate end-to-end path capacity, in NDN networks (Chapter 5). Moreover, NDN has unique features. To address the differences, we propose the first caching policy detection method in this dissertation (Chapter 6).

2.5 The NDN-DPDK Forwarder

Internet users and applications have demonstrated an ever-increasing need for high-performance packet forwarding [68]. As the experimental NDN forwarder, Named Data Networking Forwarding Daemon (NFD) is widely used in the past years to prove its potential to improve data distribution. However, NFD is aimed at allowing easy experiments with new protocol features and algorithms. Its implementation emphasizes modularity and extensibility, not for performing at high-speed packet forwarding rates. Therefore, researchers still struggle to improve data distribution, especially scientific data distribution. To this end, the National Institute of Standards and Technology (NIST) is developing an NDN forwarder, NDN-DPDK [23], with Data Plane Development Kit (DPDK) [24]. The DPDK framework provides a set of data plane libraries and network interface controller polling-mode drivers for offloading packet processing from the operating system kernel to processes running in userspace. Using dedicated CPU cores and directly talking to network card drivers allow DPDK to achieve higher computing efficiency and higher packet throughput. With DPDK, NDN-DPDK leverages concurrency to improve forwarding throughput.

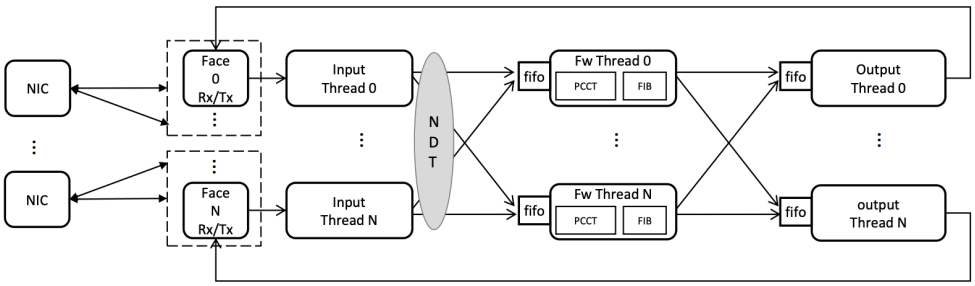


Figure 2.5: Overview of the NDN-DPDK forwarder [23].

Figure 2.5 shows the high-level design of the NDN-DPDK forwarder. First, the NDN-DPDK forwarder creates virtual interfaces, called Faces, over the network interface controllers (NICs, also known as physical network interfaces), to send and receive NDN packets. The data plane has three main stages, namely, input, forwarding, and output. The input stage receives packets from an NDN Face, decodes them, and dispatches them to the forwarding stage. The forwarding stage is responsible for processing Interest, Data, or Nack packets according to the NDN protocol. The output stage is to queue packets for transmission. Leveraging DPDK, NDN-DPDK could use dedicated CPU cores and large buffer to speed up forwarding. Specifically, each stage in NDN-DPDK may contain one or more threads pinned to CPU cores. Besides, a large memory pool is attached to each physical NIC during initialization.

With DPDK, an input thread continuously polls the NIC to obtain a burst of packets (typically 64 packets) and place them in the large memory pool. Such bulk access greatly improves performance by reading several elements with only one costly atomic operation. Before dispatching packet, the input thread needs to decode received packets, reassemble fragmented packets, and drop malformed ones. Upon the arrival of an Interest, the input thread computes the hash of the first two name components. The input thread then selects the forwarding thread by querying the last two bytes of the hash value in the Name Dispatch Table (NDT). Since NDN routers are stateful, the incoming Data and Nack packets must be dispatched to the same forwarding thread that previously processed corresponding Interests. NDN-DPDK achieves this by letting Data and Nack carry a 1-byte field in the packet header that identifies the forwarding thread.

Each forwarding thread is equipped with a queue to receive packets dispatched by input threads. This stage follows the NDN protocol to process every packet. Each thread contains two structures: the Forwarding Information Base (FIB) and PIT-CS Composite Table (PCCT). Both are implemented as hash tables. FIB records where the content might be available and which forwarding strategy is responsible for the name prefix. PCCT combines the Pending Interest Table (PIT) and Content Store (CS) found in a traditional NDN forwarder. It records which downstream node requested a piece of content, and also serves as a content cache.

Each output thread also has a queue to buffer outgoing packets from forwarding threads. Packets are queued for transmission on a Face. Depending on the MTU size, fragmentation may happen to align with links' configurations. After transmission, the NIC driver automatically frees packet buffers for newly arrived packets.

NDN-DPDK is confirmed to improve the forwarding throughput up to 60Gbps using 100Gbps NICs [23]. However, to the best of our knowledge, NDN-DPDK has not been evaluated on real networks. In Chapter 4, we present the first benchmark results for NDN-DPDK in a real testbed. We then use the benchmark results in Chapter 5 for evaluate the work in capacity estimation.

2.6 IP Measurement

People have an interest in measuring networks since the early age of Internet. To better serve customers, Internet service providers (ISPs) continuously upgrade devices and deploy new network technologies. The introduction of new devices and technologies lead networks to grow continually in both size and complexity [69]. Managing such large and complex networks, therefore, becomes more and more challenging. Internet serves as a black-box to hide problems inside. When users experience excessively slow download, the causes may be the congested paths, mistuned TCP parameters at the end-hosts, or routing changes. Researchers and operators are eager to understand how networks are being used and why they behave the way like that [10, 11]. In particular, researchers want to characterize traffic, topology, and performance of a network or a protocol [70].

These numbers could help operators to manage networks efficiently, i.e., detect, diagnose, and fix potential network issues [70].

2.6.1 Measurement Methods

In practice, users may use various approaches to measurement the network. The two common approaches are the passive and active approaches, and both have their values [71]. In fact, they are complementary with each other, and thus some tools use a hybrid way.

Passive measurements are carried out by observing network traffic at a single or multiple critical locations in a network [72]. Passive measurements may be used for various purposes. To detect intrusion, operators could count the number of packets and bytes traveling through routers or links between specified sources and destinations, monitor bandwidth usage and protocol distribution. Passive measurements do not introduce new traffic to the network [69] and thus have no impact on network performance and measurement analysis. However, passive measurements need special privileges to access monitoring nodes. Investigating a large amount of data limits the use of passive measurements as well.

Active measurements, on the other hand, are conducted by injecting probe packets, which are specially crafted for a test, into the network [72]. Leveraging these probe packets, one could measure a path's metrics, such as delay, throughput, and so on. For example, one might measure a network's maximum carrying capacity by sending packets through it and increasing the sending rate until the network is saturated. Active measurements introduce additional network traffic. Since the injected traffic may disturb on-going traffic flows and impact measurement results, network administrators may object to probe packets. Another issue is that end-host based inferences may not be as accurate as directly monitoring on intermediate nodes.

Hybrid measurements combine active and passive measurement methods [72]. Using hybrid measurements, one could actively push probe packets into the network and passively monitor the testing path to measure the end-to-end delays. It is obvious that hybrid measurements have both the advantages and disadvantages of active and passive measurements.

2.6.2 Taxonomy of Measurements

After decades of development, researchers and engineers have a set of tools to measure IP-based networks in various aspects. Widely used measurement tools include Ping, Traceroute, and others. These tools report various metrics to quantify network conditions or performance, offering some insight into the network. One or multiple measurement tools may be used to locate network issues and fix them. In general, measurements could be classified as network state measurement, performance measurement, or traffic measurement.

Network State

The change of network state, such as network topology, path, routes, and configurations, could directly impact network connectivity, available bandwidth, and packet delays. For instance, link failures or ISP policy changes may cause routing protocols to recalculate paths. After the recalculation, the path that traffic flows travel through the Internet may change or break. Since the selected paths may contain more routers, the time spent in routers and hops between them would increase accordingly [71]. In other words, the path recalculation inevitably changes the packet delays and impact the network throughput. While determining the reason for a network change is critical, the Internet does not provide any explicit information about the network structure.

To this end, researchers proposed path change monitoring and diagnostic tools to detect path anomaly. Traceroute, for example, is a path measurement tool to get the sequence of routers from the source to the destination. Due to its simplicity, Traceroute is widely used on the Internet. Operators use it to investigate routing failures [73]. Researchers use it as the basis for ISP performance analysis [74], anomal detection [75] and others.

Performance

Network performance is crucial as it could directly impact users' experience. Networks have no guarantee of constant packet delays, and thus several metrics are used to capture the path properties, such as bandwidth, delay, jitter, packet loss rate, and so on.

Bandwidth and latency are the two most common metrics. Bandwidth is a concept that relates to the amount of data a link or network path can deliver per unit of time. The associated bandwidth metrics limit the performance of data-intensive applications, such as scientific data transfer or multimedia streaming. Latency is defined as the time that a packet takes to travel from the source host to the destination host. This metrics is essential to latency sensitive applications. Latency and bandwidth are correlated. The transport layer has difficulty in sustaining high bandwidths when the delay is high [76]. Latency sensitive applications could benefit from the lower end-to-end delays associated with high-bandwidth links [77].

Latency could be two-way or one-way. To measure Round Trip Time (RTT), a single host needs to capture the time between the initial packet and the response. The most widely used method to investigate network delay is to construct and transmit an ICMP packet [40] as Ping does. One should note that the path from a source to a destination may differ from the path from the destination back to the source [76] in IP networks. To help applications whose performance depend mostly on one direction, one-way delay metric and the associated tool, OWAMP [78], is proposed.

Bandwidth includes available bandwidth and bulk transfer capacity (BTC). Available bandwidth stands for maximum unused bandwidth of a link or path during a specific period, while BTC refers to achievable throughput of an established TCP connection. IP networks have a set of tools to measure available bandwidth and BTC, including pathload [16], pathchar [79], iPerf, and others.

Traffic Monitoring

Traffic monitoring is essential in understanding and characterizing users' activities. Characterizing users' activities allows for reproducing traffic with similar characteristics in some controlled network environment (e.g., network simulations). Moreover, monitoring traffic could help identify the on-going hidden attacks.

Monitoring has no impact on network performance as it does not introduce or generate new traffic. However, users need special privileges to monitor traffic in critical locations of a network. It is also critical to minimize both the amount of measuring devices and the amount of data collected

while maintaining the accuracy of measurements adequate [69]. Tcpdump, NetFlow [80], and SNMP [81] are some of the widely used monitorning tools in IP networks.

The taxonomy of IP network measurement is essential to NDN measurements. NDN is still a new network architecture. It adopts name-based communication, and NDN routers are stateful. Leveraging the taxonomy of IP measurement could help us identify the similar requirements and the missing pieces in NDN measurements. We discuss the needed NDN measurements in Chapter 3.

2.7 Packet Pair/Train Dispersion Probing

Researchers have a great interest in measuring the the end-to-end capacity of a path from end-hosts. Packet dispersion is one of the most famous approaches. We introduce some well-stuied packet dispersion approaches in this section as a prior knowledge for other chapters. Interrupt coalescence (IC) has significant effect on the capacity estimation results, and thus we decribe how it works as well.

2.7.1 Packet Pair Technique

The packet pair techniques are proposed to measure the end-to-end capacity of a path (bottleneck bandwidth) from the dispersion of two equal-sized probing packets sent back to back [82]. The dispersion of a packet pair is defined as the time distance between the last bit of each packet. The source sends those packet pairs to the destination, and then the destination could estimate the capacity based on the dispersion.

Figure 2.6 demonstrates the dispersion of a packet pair before and after the packet pair goes through a link of capacity C_i. For simplicity, we assume that the link does not carry other traffic. Without losing generality, if the dispersion is Δ_{in} before a link of capacity C_i, the dispersion after the link, noted as Δ_{out}, will be estimated as Equation 2.1, where L is the size of probing packets.

$$\Delta_{out} = \max\left(\Delta_{in}, \frac{L}{C_i}\right) \tag{2.1}$$

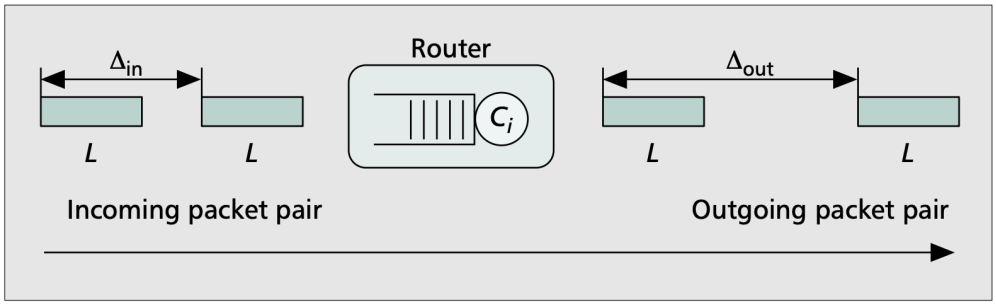


Figure 2.6: Packet pair dispersion. [77]

On a path with multiple links, the destination could estimate the dispersion as Equation 2.2 after a packet pair goes through the path, where C is the end-to-end capacity of the path. Therefore, the estimated path capacity on the destination is $C = L/\Delta_R$.

$$\Delta_R = \max_{i=0,\ldots,n} \left(\frac{L}{C_i} \right) = \frac{L}{\min_{i=0,\ldots,n} (C_i)} = \frac{L}{C} \tag{2.2}$$

One major issue of packet pair techniques is the unrealistic assumption, and no other traffic exists on the target path. Even worse, cross-traffic can either increase or decrease the dispersion. Therefore, the measurement results either underestimate or overestimate the path capacity [77]. In practice, cross-traffic packets may appear before the first packet or between the two of the probing packet pair. If cross-traffic packets are inserted into the FIFO queue between the probing packet pair at a specific link, underestimation happens. That is because any inserted packet inevitably increases the dispersion to more than L/C. Cross-traffic packets could appear before the packet pair. If cross-traffic delays the first probe packet of a packet pair more than the second packet at a link that follows the narrow link of the path, capacity overestimation occurs.

To address the above issue, researchers proposed various methods. Sending more packet pairs and using statistical methods to filter out erroneous bandwidth measurements seem to have the potential to mitigate the effects of cross traffic. Unfortunately, standard statistical approaches do not always lead to correct estimation [77]. Other methods [17, 83] are proposed in the literature perform capacity estimation using packet pair measurement as well. However, to the best of our knowledge, no investigation into the relative merits and drawbacks of these techniques.

2.7.2 Packet Train Probing

Packet train probing is an extension of packet pair techniques. Unlike packet pair techniques that only use one packet pair, a packet train uses multiple back-to-back packets. The dispersion of a packet train at a link is defined as the amount of time between the last bit of the first and last packets [77]. For simplicity, let us assume no other traffic appears on the link. If the destination receives a packet train of length N, and the end-to-end dispersion is $\Delta_R(N)$, the dispersion rate D is equal to the path capacity (as shown in equation 2.3).

$$D = \frac{(N-1)L}{\Delta_R(N)} \tag{2.3}$$

Packet train probing could increase train length N to decrease the variance of the dispersion rate D. To illustrate this, we use the example in [77]. The example considers the case of a two-hop path, where the capacity of the first link and the second link are C_0 and C_1 respectively ($C_1 < C_0$). The source sends packet trains of length N, and each packet has a size of L bytes. Assuming that the links use first come first served (FCFS) buffers, and cross-traffic (X_c in bytes) will arrive at the second link during the measurement, the dispersion of the packet train after the first link and second link is as shown in equation 2.4 and 2.5. From Equation 2.5, we can find that the variance of dispersion rate D decreases as the train length N increases.

$$\Delta_1 = \frac{(N-1)L}{C_0} \tag{2.4}$$

$$\Delta_2 = \frac{(N-1)L + X_c}{C_1} \tag{2.5}$$

Since the destination could measure the dispersion after the second link, the average dispersion rate (ADR) measured at the destination is

$$ADR \triangleq \frac{(N-1)L}{E\left[\Delta_2\right]}. \tag{2.6}$$

Given that the expected X_c is

$$E[X_c] = R_c\Delta_1 = R_c\frac{(N-1)L}{C_0}, \tag{2.7}$$

we can get the conclusion as Equation 2.8.

$$ADR \triangleq \frac{(N-1)L}{E[\Delta_2]} = \frac{C_1}{1+\frac{R_c}{C_0}} \leq C_1. \tag{2.8}$$

Prasad et al. [77] pointed three important properties for ADR:

1. ADR is less than the path capacity when $R_c > 0$;

2. ADR is not related to the available bandwidth in the path, but it is larger than the available bandwidth;

3. ADR is independent of the packet train length N, but larger N reduces the variance in the measured dispersion rate D.

Typically, packet train dispersion requires measurement software running at both the source and the destination of the path. In IP networks, measurement software can force the destination to send some form of response (e.g., ICMP port-unreachable or TCP RST packets) to eliminate the need for access at the destination. However, this approach needs the help of underlying protocols, and the reverse path capacities and cross-traffic may affect the results.

In this dissertation, we treat NDN as a superset of IP. Therefore, the end-to-end capacity in NDN networks is also a desired measurement objective. Since packet train dispersion is more robust to address cross-traffic, we use it for capacity measurements in our experiments.

2.7.3 Interrupt Coalescence

Some high-performance NICs adopt Interrupt Coalescence (IC) to reduce the per-packet interrupt processing overhead. In IC mode, NICs delay the generation of an interrupt for a few hundreds

of microseconds. In this case, packets received in the time interval are grouped, expecting the performance is improved.

Prasad et al. [84] pointed out although IC decreases the per-packet interrupt processing overhead, it introduces queueing delays and alters the "dispersion" of packet pairs or trains. They proposed to use the dispersion of consecutive packet trains to detect the presence of IC. The length of a packet train should be sufficiently long to observe at least two bursts. The approach also assumes that no significant cross-traffic arrives at any links. The path capacity estimation relies on the interarrivals between successive bursts in the presence of IC. Specifically, the dispersion between the first packet of two consecutive bursts Δ is equal to BL/C, where B is the length of the first burst, L is the packet size, and C is the path capacity. After observing the dispersion, the capacity can be easily calculated as $C = BL/\Delta$.

Our testbed contains two types of NICs, which allows us to investigate the IC effects on NDN measurements. In Chapter 5, we demonstrate that the above approach still works on NDN networks.

2.8 Summary

This chapter presented the background and related work of NDN measurements. We introduced NDN as well as its features with specific examples and compared them with IP counterparts to highlight the changes. We focus on the classification of widely-used caching policies in NDN networks. We also touched the high-performance NDN forwarder, NDN-DPDK, and the existing NDN measurement tools. Additionally, we outlined the taxonomy of IP measurement approaches in this section to help understand NDN measurement. We described the packet dispersion techniques in detail as prior work, as we investigate the feasibility of replicating IP measurement methods in NDN networks.

In our discussion of prior work, we see NDN measurement is still at its early stage. Researchers have not outlined the measurement required by NDN networks. In Chapter 3, we compare IP and NDN, and borrow the well-established IP measurement framework to propose a conceptual NDN

measurement framework. The framework helps us identify possible required measurements in NDN networks and propose solutions to address them.

Chapter 3

What Measurement Does NDN Need

This chapter outlines the measurement needed by NDN networks. Network measurement techniques are crucial for obtaining insight into networks. The insights includes network performance, a network profile, the logic behind the behavior of network control mechanisms, or any other things related to network researches and operations. For researchers, measurement helps them understand the behavior of the Internet and the traffic characteristics on the Internet. For network operators, measurement helps detect network anomaly, diagnose network problems, and fix network issues.

Today's IP networks have a set of measurement tools. Many measurement tools and approaches successfully help IP network users and operators diagnose network issues and improve network performance. We need to be able to measure NDN networks as effectively as we measure IP networks.

However, NDN measurement is still at its early stage. A few NDN-based measurement tools are proposed, but they only cover a small number of measurement types. Without sufficient measurement tools and approaches, researchers have difficulty in troubleshooting network issues and improving application and network performance. To this end, we outline the needed measurements for NDN networks in this chapter. To start, we compare the supported functionalities in IP networks with the ones in NDN networks. We point out that NDN is a superset of IP, as the NDN network layer not only supports functionality that is provided by IP but also has new features that do not exist in the IP network layer. According to the observations, we propose a conceptual NDN measurement framework. We then identify the needed measurements in measuring network state from end hosts. We also state that researchers must take the name-based communication paradigm into account when measuring NDN networks.

3.1 NDN vs. IP

We start by comparing NDN with IP. IP measurement is well established. Comparing NDN with IP allows us to find out both the similarities and the differences between the two architecture. Leveraging the existing classification of IP measurement, we can identify needed NDN measurements.

NDN and IP adopt different communication mechanisms. IP was designed to create a communication network, where packets named only communication endpoints. In IP networks, each packet contains both the source and destination addresses. Routing and forwarding in IP networks are based on IP addresses. When IP addresses are decided, the path is relatively constant. To retrieve data, applications in IP networks usually need additional services or systems. A simple use case is retrieving an online file in IP networks. After we publish this dissertation in my personal account, its URL may be "https://www.colostate.edu/c̃hengyu/dissertation.pdf". To access this file, a local IP application needs to ask the local DNS server, which is an application-layer system, for the addresses of the school's web servers. As a response, the DNS server may return a list of IP addresses that belong to a CDN network to redirect the user to one of the web servers. After the HTTP connection is established for the data transfer, the server sends data to the application.

On the contrary, NDN is based exclusively on named content, without referring to unique endpoints. Specifically, NDN is a consumer-driven architecture that focuses on retrieving a piece of content according to its name. When requesting a content chunk, a consumer sends out an Interest packet, which is forwarded in the network until it can be satisfied by a Content Store (CS), or by an application that serves as the content producer. The actual bits are returned to the consumer in a Data packet that follows the reverse path of the Interest packet. As in IP, the Forwarding Information Base (FIB) table is used to determine an Interest's next hop. An entry in the NDN FIB consists of a namespace and one or more possible faces. Each face represents an interface to a possible next hop. When the face list consists of multiple faces, the forwarding plane needs to decide on which face(s) to forward the Interest. The forwarding decision is determined by the selected forwarding strategy. Depending on the network states, a consecutive Interest packet

Table 3.1: NDN has functionalities provided by other layers on IP networks

NDN	IP
Discovering the "best" location	DNS
Content caching	Web caching, CDN, etc.
Stateful forwarding	Services at higher layers

with the same name or prefix may be forwarded along different paths and may reach different data sources. As a result, a Data packet may be retrieved from various data sources, a content publisher, a content repository, or the CS of one of the on-path nodes. The file mentioned in the example in IP networks could be named as "/colostate.edu/chengyu/dissertation.pdf" in NDN networks. To retrieve data, an NDN application simply sends out the Interest packet that contains the name of the desired content to the NDN network. The network then leverages the deployed forwarding strategy to discover the "best" location (the origin, the cache, or the repository) for retrieval. If the previous path is congested, the forwarding strategy may choose another idle path to retrieve data.

In the above examples, we can find NDN has functionalities that were performed at other upper layers in IP networks (Table 3.1). We can roughly think of NDN as a superset of IP. Typically, IP is designed to address hosts to form communication networks. NDN could name data in various ways, and the IP address is one feasible format. IP applications need help from DNS and/or other mechanisms to locate where the data is. NDN networks, in contrast, could find the "best" location that hosts the data without applications' engagement.

Caching happens at a different layer in the two architecture. Caching in IP networks happens at the application layer, while NDN routers cache Data packets with various caching policies at the network layer.

Another sharp contrast between IP and NDN is the forwarding. The typical IP forwarding follows a single path chosen by the routing process. IP can support complex forwarding functionalities by binding the service to the IP address. For example, assigning a special anycast address for the DNS root servers. However, the solution is architecturally awkward as the service is a layer four or even higher concept. NDN eliminates this awkwardness by inherently supporting complex forwarding functionalities based on names. Each NDN router is equipped with multiple forward-

ing strategies, and thus it can provide complex network layer functionalities to adapt to network changes.

3.2 Needed NDN Measurement

Interest in measuring networks has existed since the early days of the Internet. After many years of development, today's IP-based Internet has a set of measurement tools available in various aspects (Section 2.6). Since IP networks are address-based, those tools were typically developed to measure network metrics between endpoints. For a specific purpose, for example, measuring available bandwidth, one could choose pathload [16], pathrate [85], or something else from the existing tool set to conduct measurements in the network. Those tools generate reports about the target network in various aspects so that people could gain insight into networks.

The shift to a content-centric based communication mechanism fundamentally changes the way to measure NDN networks. The rich in-network functionalities not only improve data distribution performance but also introduce complexity in NDN networks. Those in-network functionalities have significant effects on data retrieval performance. Measuring their behaviors could help peope understand networks. However, due to the complexity, only a few achievements are currently available in NDN measurements [18–21]. These tools give users limited capabilities to identify and troubleshoot network issues. Users are unable to verify the deployed NDN networks are running as intended.

To classify NDN measurement, we borrow the IP measurement framework [70]. NDN measurement types could drop in categories like performance, traffic, and network state (Figure 3.1). Some measurements may drop into the intersection of multiple categories. Link-layer measurement is one of the examples. Similar to IP, link utilization in NDN networks depicts the state of link usage. Link-layer measurement could affect each other. On the one hand, the network state could change link utilization. Forwarding strategies may balance the load and let traffic equally distribute on available links. On the other hand, link utilization changes could affect performance. Transferring data on idle links could have low retrieval delay and loss, leading to high performance. A high

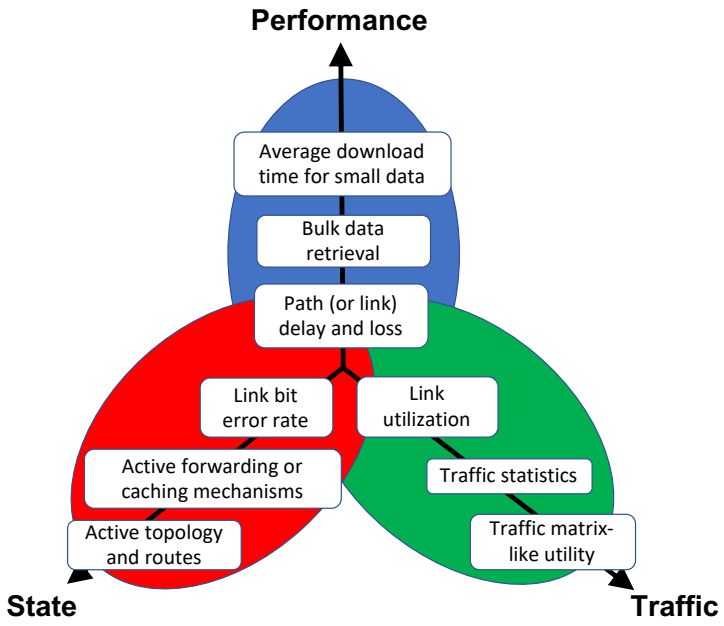


Figure 3.1: NDN measurement types.

link bit error rate will hurt delay and increase the loss rate. Especially, the forwarding strategy in NDN networks may adapt to the bad link state and then change the link utilization. Researchers should investigate the relationship between various objects and propose approaches to identify the cause of network issues.

When we look at the higher layer, communication in NDN networks always happens under a name prefix. Given a specific name prefix, users may want to know the retrieval performance over NDN networks. The retrieval performance gives people results for specific data transmission directly. Typically, the performance measurement utilizes active measurements to quantify performance. The performance could be for bulk data retrieval or for a piece of content. Metrics could be a delay, throughput, and others. Researchers need to choose performance metrics for NDN data retrieval carefully. Traditional metrics like delay, throughput, packet loss rate, and others may still be meaningful in NDN measurement. Some other metrics like jitter may be misleading, as NDN has caching and forwarding strategies, and both may add or reduce delays. Researchers need to investigate performance metrics to point out valid metrics and propose new metrics. Tools to mea-

sure performance in NDN networks should give meaningful metrics that could help understand the effect of NDN features.

Detecting network state is essential to understand networks' behavior and fix potential network issues. In IP networks, a network state could be topology, configuration, and routing. In NDN networks, active routes and active topology are name-based. Even when one node is the origin of multiple name prefixes, routes and topology states vary from one name prefix to another. Therefore, we need to note that NDN measurement is the name prefix specific. Besides, NDN is a superset of IP. In addition to the traditional IP network configurations, NDN has more configurable elements, including forwarding, caching, and their attributes. Detecting network configurations in NDN networks is critical, as they are tightly correlated to data retrieval performance. Similar to IP networks, measuring network state in NDN networks could be direct or indirect. Direct techniques rely on network routers as measurement nodes, which require some management protocols to give special privileges to the outside node(s). The outside node then could collect data directly from routers for analysis. Indirect techniques rely on other active or passive measurements at some endpoints to infer network states.

NDN routers are stateful. A large amount of probing traffic will definitely change the state of target networks. Therefore, proposed approaches should minimize the introduced overhead when measuring NDN networks.

Traffic measurement is as important as the other two categories. Monitoring traffic in networks allows people to learn what healthy networks should be. From there, operators could detect network anomaly by monitoring on-going traffic. Researchers could extract statistics from traffic at various levels (Interest, Data, etc.). Those realistic traffic statistics will be used in simulations to evaluate applications or other network solutions. Researchers should also pay attention to packet fields. For example, NDN allows content publishers to decide the lifetime of Data packets. Instant messages usually have a smaller lifetime so that applications could reuse the packet names later. Videos and scientific datasets barely change, so their lifetime could be longer or even unlimited. Mistakenly set a short lifetime to scientific data may make them evicted too soon. Researchers

Table 3.2: A list of measurements in IP and NDN

Measurement Types	IP	NDN
Availability/Latency	ping, sting [86], fping [87], echoping [88], owamp [89], etc.	ndnping [18], ndnping-dpdk [90]
Bandwidth/Throughput	iperf/iperf3 [91], Netperf [92], bing [93], clink [94], pchar [95], pathrate [85], SProbe [96], etc.	ndn-traffic-generator [97], **Chapter 4**, **Chapter 5**
Path/Topology/Routing	traceroute, Rocketfuel [98], Mtr [99], BGPmon [100], etc.	NDN-trace [19], ccninfo [67]
Forwarding State	N/A	None
Caching State	N/A	**Chapter 6**
Monitoring/Analyzers	tcpdump [101], IPTraf [102], tcpdpri [103], SNMP [104], argus [105], NetFlow [106], etc.	ndndump [18]
Network simulation	dummynet [107], ns-3 [108], OPNET Modeler [109]etc.	ndnSIM [110], Icarus [111]
Infrastructure	perfSONAR [112], RIPE Atlas [113], Sam Knows [114], etc.	NDN Measurement Framework [115]

could put collected statistics into a traffic matrix. A traffic matrix is a tool that gives the traffic volumes between origin and destination in an IP network. Operators use the traffic matrix for IP network capacity planning and management. The traffic matrix is also important to NDN network management. However, NDN communication is name-based, and NDN routers are stateful. Researchers should redesign the traffic matrix that fits NDN's needs.

Since IP and NDN have both similarities and differences, we use a table to help identify research gaps in NDN measurement. As shown in Table 3.2, IP is well established. It has a set of achievements in all the IP measurement types. On the contrary, NDN only has a few tools available. NDN lacks measurement tools almost in every measurement type. IP has tools to measure either one-way or two-way latency, but NDN tools only provide the round trip delay time (RTT). Both IP and NDN have traffic generators to measure bandwidth, but NDN does not have packet dispersion techniques, such as pathrate [85], to estimate the end-to-end capacity with minimal overhead. Additionally, NDN has no tools to measure NDN-specific network states, i.e., forwarding state,

and caching state. NDN networks need tools to monitor, measure, and analyze networks in most measurement types.

This dissertation fills two gaps in the table. We leverage the similarity to replicate IP measurement approaches in NDN. We also address the differences by detecting NDN specific state from end hosts. The first work drops in the measurement type of bandwidth/throughput. It enriches the measurement type by replicating IP-based packet dispersion techniques in NDN networks (Chapter 5). As the prior work, Chapter 4 provides the insights of NDN-DPDK [23] forwarder performance on a dedicated testbed. The other work (Chapter 6) is the first approach to measure caching state in NDN networks.

3.3 Summary

In this chapter, we outline the required measurements in NDN networks. Specifically, we compare NDN and IP mechanisms and point out that NDN contains functionalities that are performed at other upper layers in IP networks. Therefore, NDN can be seen as a superset of IP. We then borrow the taxonomy of IP measurements to propose the conceptual NDN measurement framework. The framework inspires us to compare existing IP and NDN methods to identify research gaps in NDN measurement.

We point out NDN networks require tools to enrich the measurement in almost every measurement type. This dissertation focuses on measuring networks from end hosts. Chapter 5 enriches the work in bandwidth/throughput measurement by replicating IP-based packet dispersion techniques in NDN networks. Chapter 6 present the first caching decision detection approach to fill the gap in caching state measurement.

Chapter 4

NDN-DPDK Performance

This chapter presents an experimental performance evaluation of the NDN forwarder (NDN-DPDK) that is built over the Data Plane Development Kit (DPDK) [24]. The goal is to provide throughput results and performance insights for experiments in capacity estimation in NDN networks (Chapter 5). We conduct experiments on the NDN-DPDK forwarder rather than the commonly used NDN Forwarding Daemon (NFD) [37] for several reasons. First of all, NFD is well-known for its low-performance as the primary design goal of NFD is to support diverse experimentation of NDN technology. The design emphasizes modularity and extensibility to allow easy experiments with new protocol features, algorithms, and applications [37]. In contrast, DPDK leverages concurrency so that NDN-DPDK significantly boosts packet processing performance and throughput in content distribution. Moreover, the overlay approach that NFD adopts introduces complexity in measurements. The underlying TCP/IP modules may hurt network performance, and measurements must exclude these modules' effects to evaluate NDN performance. In this case, we do not need to address the potential issues introduced by other modules in overlay approaches. Encapsulating NDN packets in Ethernet frames also introduces less overhead, and thus the goodput of the NDN forwarder could be better. Most importantly, NDN-DPDK is developed at the National Institute of Standards and Technology (NIST). It has the potential to be deployed in the real world. Investigating its performance could have a significant contribution to future projects.

4.1 The Benchmark Tool - ndnping-dpdk

To benchmark the NDN-DPDK performance, we use the traffic generator, ndnping-dpdk, included by the NDN-DPDK repository.[1] Unlike ndnping [18] and ndn-traffic-generator [97] from

[1]The data of this dissertation are using commit a10184933e9481312383777db38ced7549e77f07

named-data.net, this implementation does not use a local forwarder. The application directly sends NDN packets through the network interfaces without introducing additional overhead.

The traffic generator contains a benchmark module to allow users to measure the minimum sustained interval (MSI) for a name prefix. MSI indicates the ability of a forwarder to satisfy the specified Interest satisfaction value. Users can use MSI to calculate other metrics, such as the packet per second (PPS), the throughput of a forwarder, and others.

Khoussi et al. [23] use the tool to evaluate the NDN-DPDK forwarder throughput on a server with two NUMA nodes. Their goal is to verify that NDN-DPDK can perform at high packet forwarding rates. The parameters to achieve higher performances are also given in their work. While their work uses multiple data retrievals to show the forwarder throughput, we are more interested in learning the performance of the single data retrieval through the NDN-DPDK forwarder. The single data retrieval throughput provides the baseline for the experiments in Chapter 5. Our works differ in experiment settings as well. Their evaluation is on a standalone server, but we build a dedicated testbed with three physical machines. Since NDN-DPDK does not support the network cards used on our testbed, our experiments use the AF_PACKET socket to receive and send packets. Therefore, the maximum transmission unit (MTU) is 1500B in our experiments, which inspires us to evaluate the effects of MTU in data retrieval performance.

To start the benchmark, users must specify several parameters, such as the target name prefix, some Interest related parameters, the Interest interval range that is used for benchmark, and the target Interest satisfaction ratio. With all those parameters, ndnping-dpdk can adjust the Interest sending rate in the given range with provided interval steps. The tool leverages the binary search to narrow down the Interest sending interval, such that Interest satisfaction ratio stays near 100% within a period (typically 60s). Samples during the warm-up period are discarded to ensure the correctness of the measurement.

Figure 4.1 demonstrates the process to benchmark an NDN-DPDK forwarder. The MSI search starts at 6250ns. Since it can satisfy the specified Interest satisfaction, ndnping-dpdk continues to try the next MSI by halfing the previous MSI, 3125ns in the figure. The failure to find the MSI will

trigger an increase in the interval. After several rounds, ndnping-dpdk could report the measured MSI in the specified interval range. It is quite evident that the measurement is time-consuming. Although the binary search shortens the measurement time, ndnping-dpdk typically needs several minutes to inject a large amount of data into the network. Moreover, given that NDN forwarder is stateful, injecting traffic into the NDN network would be more likely to introduce competition among on-going flows for the scarce network resources or even interrupt the on-going cross traffic. Despite ndnping-dpdk has above drawbacks, it is a handy tool to generate the performance results, helping users study the forwarder performance.

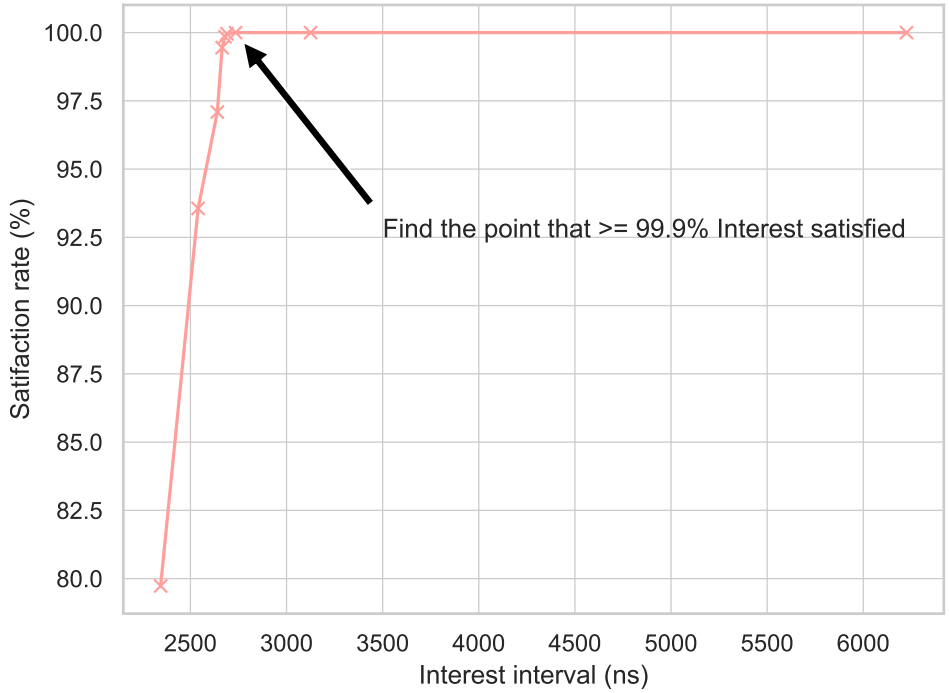


Figure 4.1: ndnping-dpdk needs several rounds to complete benchmark.

4.2 Experiment Settings

To validate the performance of NDN-DPDK without introducing any issues to the local network, we set up a local testbed. Our local testbed contains three machines to form a linear topology

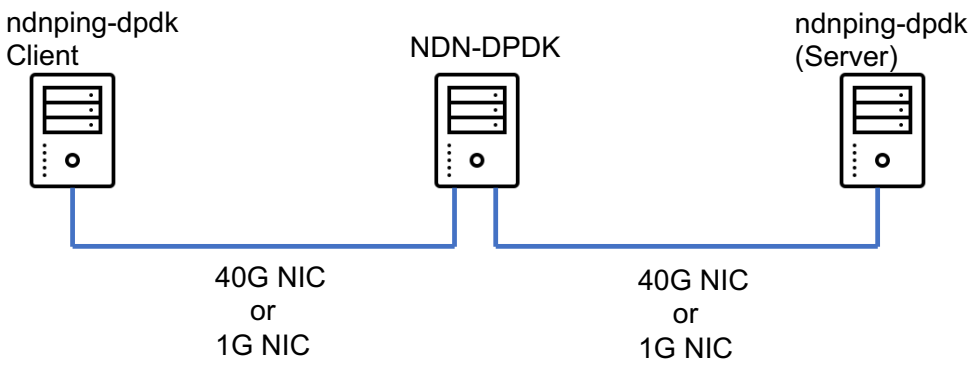


Figure 4.2: Local Testbed Topology.

(Figure 4.2). Each machine has 10 Intel(R) Xeon(R) E5-2660 v3 CPUs and 128G memory to allow the forwarder to have sufficient resources to work efficiently. Additionally, two types of network cards (NICs) are used on these machines to help us understand the forwarder's behavior over various NICs. The maximum transmission unit (MTU) for both NICs is 1500B, and thus fragmentation will happen when large Data payload lengths are used. We summarize the hardware specifications in Table 4.1. Note that we were running CentOS 7.6 on all the machines.

Table 4.1: Hardware Specifications

Operating System	CPU	Memeory	Numa Node	Network Cards
CentOS 7.6	Intel Xeon E5-2660 v3 (10 cores)	128GB	1	Broadcom BCM5720 (1Gbps) Chelsio T580-CR (40Gbps)

Unfortunately, NDN-DPDK does not directly support either of the local NICs. That means NDN-DPDK cannot use dedicated NICs to process NDN packets. Therefore, we use the raw socket to allow NDN-DPDK to send and receive NDN packets through the kernel to collect the initial NDN-DPDK performance results. Unlike the traditional way to use the raw socket, DPDK uses PACKET_MMAP, which provides a mmap'ed ring buffer to improve performance. The buffer is shared between user space and kernel for sending and receiving packets [116]. Although using raw socket cannot produce the best performance of NDN-DPDK, it is useful to help us learn the lower bound of NDN-DPDK forwarder performance. More importantly, many deployed NICs do

not support DPDK, and using raw socket may be the most widely-used approach to deploy NDN-DPDK. Additionally, letting packets go through the kernel allows us to utilize the widely-used packet capturing technology to dump traces for study (Chapter 5).

Khoussi et al. [23] listed several factors that could potentially affect NDN-DPDK forwarder's performance. The forwarder can control some factors, and the application configuration can control others. Forwarder-related factors include the number of forwarding threads, placement of forwarding threads onto non-uniform memory access nodes (NUMA), forwarding thread's queue capacity. Applications can decide the number of name components, the data payload length, and Interest sending rate.

Our experiments adopt the parameter values provided by the initial sample configuration for the DPDK forwarder. The authors found those reasonable parameter values during their experiments [23]. We list the chosen parameter values in Table 4.2. Specifically, it contains parameter values used in the memory pool, face creation, forwarder data plane, and core allocation in our experiments. Parameters in Name Dispatch Table (NDT) and FIB will not affect the single prefix data retrieval performance, and thus we do not include them in this dissertation. Readers are suggested to refer the sample initial configuration object for the meaning of each parameter and the rest section for detail values.

The NDN-DPDK forwarder performance could be affected by the number of name components, the data payload length, and Interest sending rate from the client as well. Since applications can decide those factors, the generated traffic allows us to benchmark the performance of the NDN-DPDK forwarder under those conditions. In our experiments, we collected performance results under various factor values, providing a baseline for later investigation.

In each experiment, FIB entries are registered in the NDN-DPDK forwarder to allow Interests to forward toward the producer. The producer would satisfy every Interest with a Data packet of the same name. We do not care about the caching effectiveness of the forwarder in those experiments so that each name is unique. We repeated the experiment 10 times and plotted the collected results

Table 4.2: NDN-DPDK Configuration

Memory Pool Parameters	IND	Capacity	2097151
		Cachesize	337
		Dataroomsize	N/A
	ETHRX	Capacity	1048575
		Cachesize	465
		Dataroomsize	2200
	Name linearize	Capacity	65535
		Cachesize	255
		Dataroomsize	N/A
	HDR	Capacity	65535
		Cachesize	255
		Dataroomsize	N/A
	INTG	Capacity	65535
		Cachesize	255
		Dataroomsize	N/A
	INT	Capacity	65535
		Cachesize	255
		Dataroomsize	N/A
	Data	Capacity	65535
		Cachesize	255
		Dataroomsize	8600 on server-side
Face Creation Parameters	ethmtu		1500
	ethrxqframes		8192
	ethtxqpkts		256
	ethtxqframes		8192
	socktxqpkts		256
	socktxqframes		1024
	chanrxgframes		4096
Forwarder Data Plane Parameters	fwdqueuecapacity		128
	latencysamplefreq		16
	pcctcapacity		131071
	cscapmd		32768
	cscapmi		32768
Number of Cores Allocated to Forwarder	RX		2
	TX		2
	CRYPTO		1
	FWD		5

in the next section. Moreover, we benchmark the performance under various conditions, including the various payloads and various name components.

4.3 Key Outcomes of Measuring NDN-DPDK

Interest and Data parameters can be manipulated by the traffic generator to produce traffic as needed. We collected the performance data using various combination of Interest and Data parameters. Those data give insights into NDN performance.

4.3.1 Effect of Factors on Performance

To start, we investigate the effect of the number of FIB entries on forwarder performance over the 40Gbps NICs. In the beginning, we registered one FIB entry to allow the traffic to go through the forwarder. We fixed the Interest name to use five name components, and each component contains five letters: /ABCDE/FGHIJ/KLMNO/PQRST/UVWXY. We only used zero data payload length to eliminate the effect of data payload lengths, and we can focus on the effect of name components. After ndnping-DPDK collected MSI with one FIB entry, we added more FIB entries. Then we repeated the experiments until the number of FIB entries reaches 375. In our experiments, each entry may contain up to 26 name components, and each component may have up to 16 letters to make each entry unique. Figure 4.3a shows that the number of FIB entries does not change the MSI. Therefore, for simplicity, we use one FIB entry in rest measurements.

A longer name, which contains multiple name components, requires more iterations during table lookups, potentially increasing Interest forwarding latency. Our second experiment is to investigate the effect of the name component number on forwarding performance. We inserted one FIB entry to ensure the basic reachability of the producer. Used Interests contain various number of name components (one, five, and 25). We use the 40Gbps NICs for this experiment, and the data payload length is set as zero. Our results (Figure 4.3b) show that longer names do introduce overhead. When the number of name components is one, MSIs are around 1.3 us. However, when the number of name components reaches 25, MSI stayed around 2.5 us. The increase in MSI

directly lowers the PPS and hurts the throughput for zero payload length. However, the conclusion is not valid for large payload lengths because the results with five name components. Figure 4.3c shows that the measured MSI is slower (around 3.5us) when the payload is 3000 Bytes.

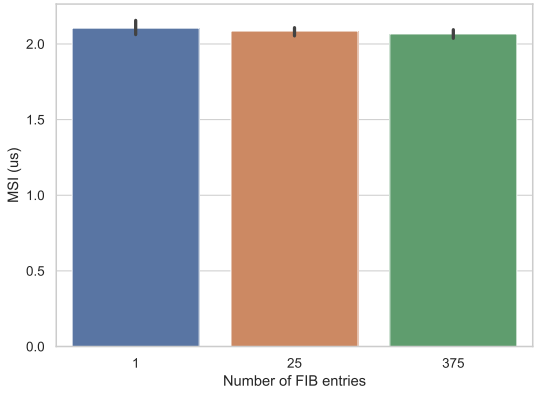
The data payload length could affect the performance of NDN-DPDK forwarder. A larger payload length leads to a large Data packet. The large Data packet increases demand for memory bandwidth, thus potentially increasing latencies. Additionally, transmitting such large Data packets over an interface whose MTU is 1500B will introduce fragmentation. Fragmentation could further delay packets.

We used five name components and one FIB entry in our experiments to check the effect of payload lengths. Figure 4.3c proves our theory. The larger payload length increases MSI, no matter which network card is used. The MSI results for both NICs are similar when the payload length is zero. The size of Interest packets is small, and it does not place much pressure on NICs. It is mainly the dedicated CPUs' job to process Interests. While a larger payload length is used, the corresponding MSI also increases. When the payload length reaches 8000, the MSI for 1Gbps NIC is as high as 75us, but the one for 40Gbps NIC stays around eight us. We believe this is because the 1Gbps NIC becomes the bottleneck and limits the Interest rate. Next subsection presents additional results to confirm this statement.

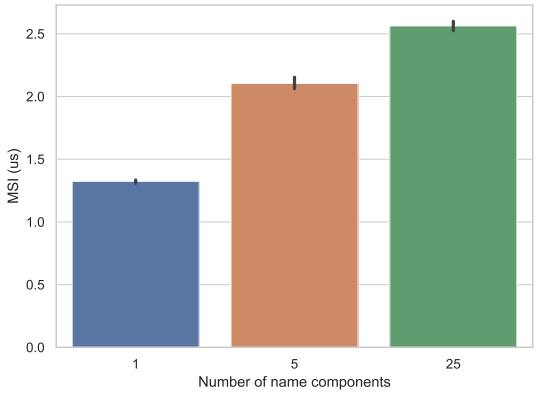
4.3.2 Estimated Forwarder Performance

MSI is simple but not straight-forward for users to understand the forwarder performance. Other performance metrics, such as the packet per second (PPS) and goodput, are more friendly to users. We can estimate those performance metrics based on MSI results. Figure 4.4 presents the estimated performance results of various payload size on the 40Gbps NIC from various aspects. The results give us a general idea about the range of performance for a namespace.

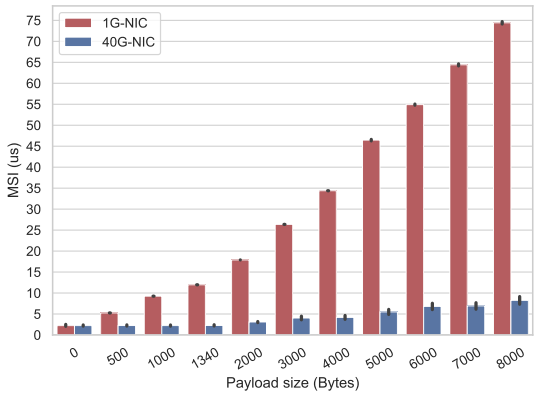
The results for the 1Gbps network card indicate that the measured performance is bounded by the network card, not the NDN-DPDK forwarder. Figure 4.4a presents the results in the thousand PPS for both types of NICs. One could find that the PPS results for both NICs are similar when



(a) Measured MSI with various FIB entries.

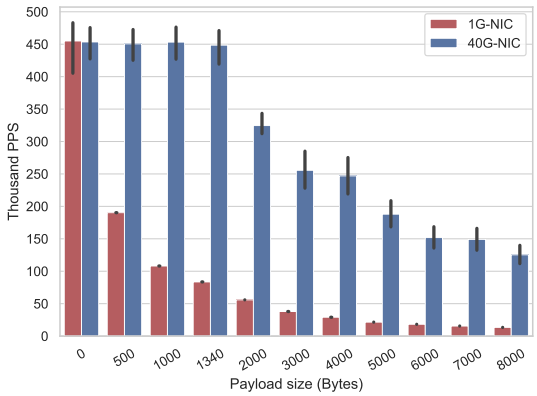


(b) Measured MSI with various name components

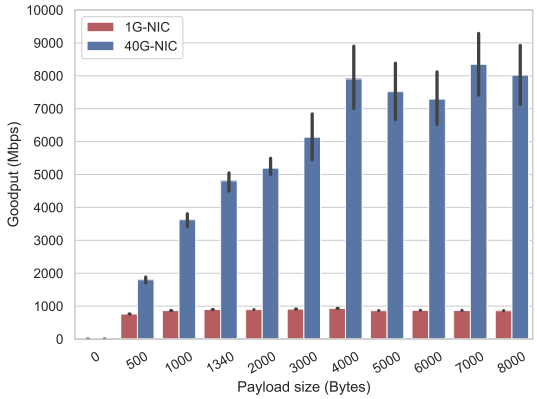


(c) Measured MSI with various payload size.

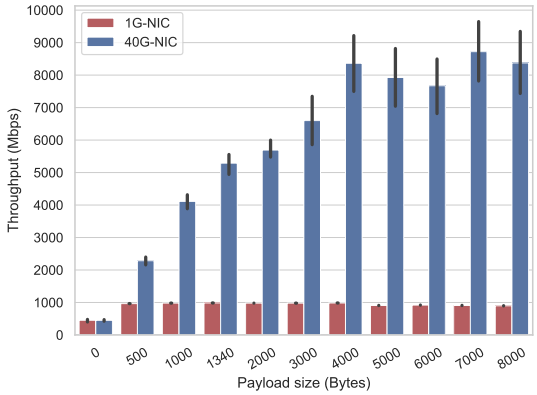
Figure 4.3: The effect of various factors. (a) Measured MSI with various FIB entries. (b) Measured MSI with various name components. (c) Measured MSI with various payload size.



(a) Estimated PPS with various payload size.



(b) Estimated goodput with various payload size.



(c) Estimated throughput with various payload size.

Figure 4.4: Estimation for various payload size. (a) Estimated PPS with various payload size. (b) Estimated goodput with various payload size. (c) Estimated throughput with various payload size.

payload length is zero. This is because the Interest and Data packet sizes are small (94 Bytes for Interest packets, and 124 Bytes for Data packets). As mentioned before, such a small packet does not place too much pressure on the network card. With the increase of payload lengths, the MSI for the 1Gbps network card drops quickly as the network card becomes the bottleneck. When the Data pipeline reaches the capacity of the network card, ndnping-dpdk decreases the interest sending rate to ensure the content delivery to avoid packet drops. Figure 4.4c demonstrates that the content delivery over the 1Gbps network card stays around 1Gbps for any payload length that is larger than 500 Bytes.

The 40Gbps NIC has a larger capacity so that measurement over the 40Gbps NIC could demonstrate the actual performance of NDN-DPDK. As shown in Figure 4.4c, the throughput never touched the 40Gbps NIC's capacity. One may notice that the standard deviation in Figure 4.4 is not small. This is because the MSI results of the NDN-DPDK forwarder vary from one experiment to another. Once the DPDK forwarder is running, ndnping-dpdk reports that there are little differences in MSI results between measurements. The dedicated CPU cores help the forwarder eliminate the overhead during the packet forwarding.

When 40Gbps NICs are used, the PPS drops with payload lengths (Figure 4.4a. An interesting finding is that payloads with the same number of fragments have similar PPS values. For instance, the estimated PPS for payload lengths ranges from 0 to 1340 Bytes stay around 450 thousand PPS. The results for payload 3000B and 4000B are similar as well. We claim the hop-by-hop fragmentation contributes this. The hop-by-hop fragmentation on NDN networks [117] requires the forwarder to assemble fragments on each NDN router, introducing additional delays. The link MTU is 1500B in our experiments. Transferring a large Data packet over those links would unavoidably trigger fragmentation. Figure 4.4a shows that fragmentation is critical to performance. Payload lengths 0B, 500B, 1000B, and 1340B generate same thousand PPS results as they all have only one fragment. Both payload lengths 3000B and payload 4000B have three fragments, and their PPS values stay around 250k.

Neither MSI nor PPS can represent the efficiency of NDN-DPDK forwarder. Small MSIs do not necessarily produce high goodput. Applications usually put bits in each Data packet as many as possible to save the overhead introduced by headers. Figure 4.4b shows the estimated goodput for using various payload lengths. When the payload length is zero, applications cannot get any meaningful goodput. Not too surprisingly, using 7000B as the payload length gives the best goodput, around 8Gbps. NDN header processing dominates content distribution. Large Data packets use less NDN headers Bytes to carry more payload, and thus it saves header processing time. Payload lengths that are larger than 3000B produce better goodput than smaller payload lengths. Besides, if the number of generated frames is the same, carrying more actual bits usually indicates better goodput. Both payload 7000B and payload 6000B produce five frames, but payload 7000B has better goodput than payload 6000B. The conclusion is correct to payload 3000B and payload 4000B; they both generate three frames. Payload 4000B has more bits in the last frame, and thus the goodput is larger. On the contrary, carrying more bits may not produce a better goodput if the payload is not carefully chosen. Each fragment introduces additional delay in Data pipeline. Payload 8000B generates a goodput 8Gbps, but payload 7000B gives a slightly better result. The last Ethernet frame of payload 7000B is 1358 Bytes, while the last frame of payload 8000 is 912 Bytes, which does not use the frame efficiently. This fact indicates the importance of efficiently utilizing the fragments.

4.4 Summary

After the initial benchmark and analysis of the NDN-DPDK forwarder on the local testbed, we find that both the name component length and Data payload length affect the forwarder performance. Longer name components require more iterations during table lookups and increase Interest forwarding latency, while larger payload lengths place pressures on the network card to transmit actual bits. In general, payload lengths dominate data retrieval performance. Moreover, we find that efficiently utilizing fragment is critical to get better goodput for applications. These findings instruct us to choose NDN packet parameters carefully.

As a straight-forward active measurement tool, ndnping-dpdk benchmark module is time-consuming. Ndnping-dpdk requires users to specify a range of Interest sending rate to start the measurement task, If the rough MSI is unknown, a user needs to use a larger range for each condition. As a result, the experiments took several minutes to finish. Additionally, ndnping-dpdk injects a large amount of data into the network to get a closer estimation of the capacity, which may introduce severe issues to the on-going traffic. In the next chapter, we investigate the feasibility of using the packet pair technology [118] to measure NDN networks.

Chapter 5

Packet Dispersion Techniques in NDN

Chapter 4 shows that the traffic generator tool, ndnping-dpdk, adopts an active measurement approach to measure the NDN networks. The tool can serve a traffic generator to benchmark a forwarder or a network. The goal of the benchmark module is to find the minimum sustained interval (MSI) such that the Interest satisfaction ratio stays near 100% within a period. The MSI can then be used for calculating other performance metrics, such as packet per second (PPS), throughput, and goodput.

The ndnping-dpdk tool has drawbacks like other straight-forward designed active measurement tools. It needs to inject a large amount of traffic into the network. NDN networks are stateful. The injected traffic could severely affect cross traffic as it may occupy both the bandwidth and cache resources, and changes the forwarding statistics on intermediate routers. The measurement process is also time-consuming. The ndnping-dpdk tool needs to try several Interest intervals that are within the specified interval range for measuring the MSI. IP networks have solutions, like packet dispersion techniques [82], to address the above issues.

In this chapter, we present the first work to investigate the feasibility of replicating packet dispersion techniques in NDN networks. Specifically, when content names are unique, and only a single producer generates those content, data retrieval in NDN networks is similar to IP networks where packets are exchanging between two end hosts. We demonstrate that the packet dispersion techniques can estimate the end-to-end capacity when 1Gbps NICs are used. We also show that the DPDK forwarder introduces variances in packet interarrivals when the forwarder uses 40Gbps NICs. Additionally, we identify the potential bottlenecks in the design of NDN-DPDK.

5.1 Experiment Settings

We use the same topology as the benchmark experiments shown in Chapter 4: a linear topology with three machines and MTU is 1500B. After setting up the topology in Figure 5.1, we let the

ndnping-dpdk client send out 64 Interests to the server. These Interests bring back Data packets with the same name. Each Interest carries a unique name so that no Data packet is from the in-network cache.

The bulk and burst access feature of Data Plane Development Kit (DPDK) [24] allows the 64 Interests to be sent out with only one costly atomic operation. The initial packet intervals are small so that the server-side could capture the capacity information by checking the interarrivals. As mentioned in Chapter 2, using the AF_Packet with DPDK lets the traffic go through the kernel. Since Tcpdump is capable to timestamp all the packets when they arrive the kernel, we use it to dump the trace on all the interfaces at nanoseconds. After collected the dumped trace, we investigate the packet interarrivals to study the performance for both Interest pipeline and Data pipeline.

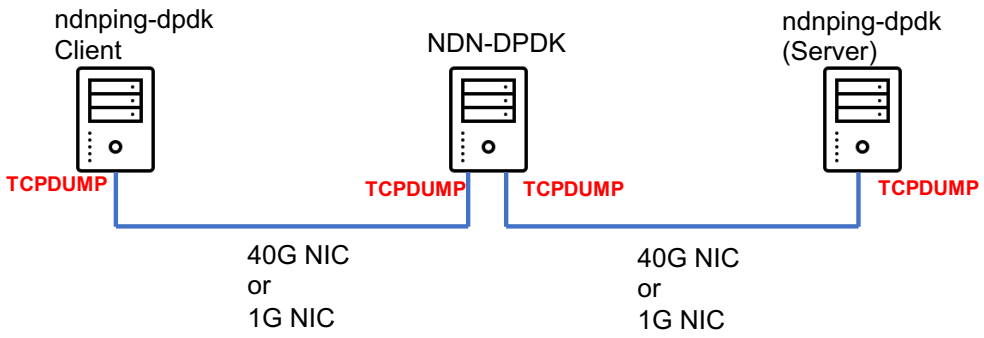


Figure 5.1: Experiment settings for measurements using packet pair technology.

On Interest pipeline, we fixed the Interest name to have five name components, and each component contains five letters: /ABCDE/FGHIJ/KLMNO/PQRST/UVWXY. The traffic generator, ndnping-dpdk, appends a unique sequence number after the content name. The unique content names ensure all Interests reach the server.

Chapter 4 pointed out that efficiently utilizing every Ethernet frame could generate better performance. To study the effects of Data packet sizes on measurements, we deliberately chose six payload lengths in our experiments. These lengths were 1340B, 2780B, 4220B, 5660B, 7120B, and 8550B. The above payload lengths ensure that the last Ethernet frame for each Data packet is

close to the MTU. Some NICs may use the number of packets as the parameter to trigger an interrupt [119]. An Ethernet frame that is much less than MTU will make the calculation under-estimate the capacity. However, the chosen payload lengths can give an approximate upper bound of the performance. In our experiments, we repeat the burst sending for 30 times, generating enough data for each payload length.

In our experiments, we measure the interarrivals of Data packets for capacity estimation. First, Chapter 4 shows that the data processing pipeline dominates the network performance. The hop-by-hop fragmentation [117] on NDN networks breaks the existing packet dispersion techniques. When a large NDN Data packet cannot be encapsulated into one Ethernet frame, the packet is fragmented into several Ethernet frames. After a router receives all the fragments, it reassembles them into the NDN Data packet. The hop-by-hop fragmentation happens on routers along the path from the server to the client. The interarrivals of Ethernet frames captured on a client are determined by the capacity of the last hop link, not the bottleneck between the two ends. Therefore, the Data packet intervals should be the metric for end-to-end capacity in NDN networks. Specifically, for large NDN Data packets, the interval should be the time elapsed between the last fragment of two Data packets.

5.2 Initial Results of NICs

It is necessary to identify the behavior of network cards as they may affect the estimation results. As mentioned in the last section, we use a burst of 64 Interests to pull Data packets from the server. To simplify the tests, we use 1340B payload lengths to ensure that no fragmentation happened during the measurement. Interest packets are usually small, and their lengths are around 94B. We plot the packet intervals for the Interest pipeline and Data pipeline separately in Figure 5.2.

Figure 5.2 confirms that the Data pipeline dominates the forwarding performance. As described in Chapter 2, on the arrival of an Interest with the unique name, an NDN router needs to check against at least three data structures, the Content Store (CS), the pending Interest table (PIT), and

the forwarding information base (FIB) to forward the Interest. Besides, forwarding strategies also need to apply the local policy to the Interest. Each structure will introduce some processing delays in the pipeline. However, Figure 5.2 shows that Interest intervals are smaller than Data intervals in our experiments, especially for 40G NICs.

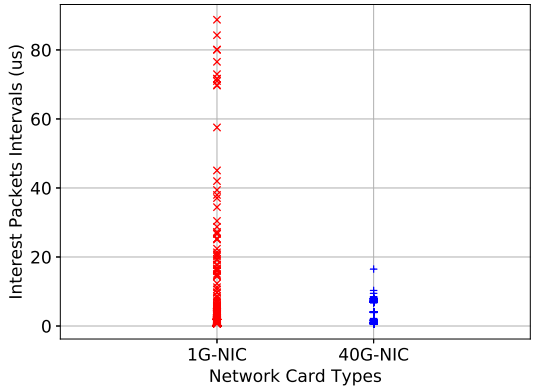
Interrupt coalescence (IC) is a technology to delay the generation of an interrupt for a few hundreds of microseconds [84]. Reducing the number of interrupts could decrease the time spent on calls and let more packets be received. NICs with IC need special approaches to do the estimation and checking if a NIC adopts IC is critical in packet pair measurements.

Figure 5.2b demonstrates that the 1G NICs adopt IC. The cumulative dispersion of a 64-packet train on the 1G NIC and the 40G NIC is shown in Figure 5.3. In the case of using 1G NICs, we see IC groups five Data packets together on the client-side. The idle time between the two groups is around 50 us. When 40G NICs are used, the packet intervals also divide those packets into several segments. However, as shown in the figure, there is a clear difference between them. The former is a deterministic event that affects all the packets of a packet train. The latter is more like a random event to break packets into segments.

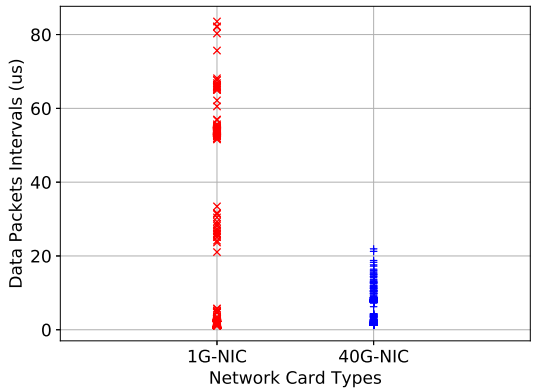
In the rest of this chapter, we present our work to study the feasibility of applying packet dispersion techniques on these two NICs separately.

5.3 Capacity Estimation on NIC with Interrupt Coalescence

IC is a deterministic event that affects all the packets of a packet train [84]. As shown in Figure 5.3, IC forms a pattern with a regular bust length and duration. The existing approach [84] proposed to estimate the path capacity based on the interarrivals between successive bursts in the presence of IC. Use B as the length of the first burst, L as the packet size, and t as the dispersion between the first packet of two consecutive bursts, then the capacity of a path is $C = BL/t$. The above approach still works in NDN when Data payload length is within an MTU. Since there is no fragmentation, the pattern is applied to all NDN Data packets.



(a) Interest packet intervals between 1G NIC and 40G NIC.



(b) Data packet time intervals between 1G NIC and 40G NIC.

Figure 5.2: Comparing packet time intervals between 1G NIC and 40G NIC for Data payload 1340B.

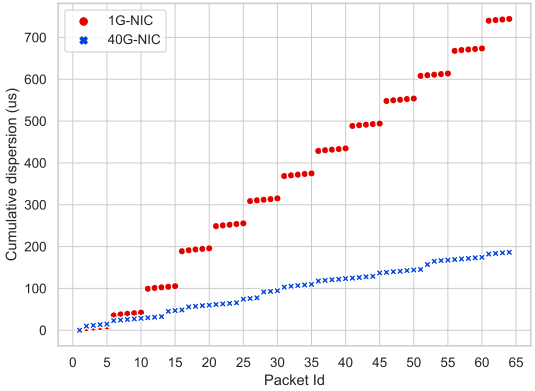


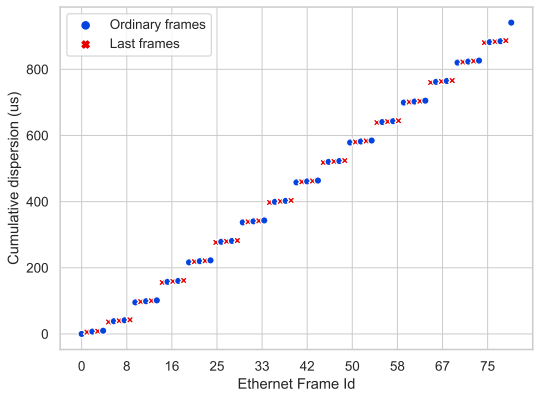
Figure 5.3: Data packet intervals with timeline for 1G NIC and 40G NIC for Data payload 1340B.

NDN Data packets can be large, and using interarrivals of Ethernet frames does not work. The underlying Ethernet network must fragment the large Data packet into smaller fragments that could fit in Ethernet frames. The hop-by-hop fragmentation makes the previous-hop the source for the received Ethernet frames on a client. The estimation using these Ethernet frames is the capacity of the last hop. If the bottleneck is several hops aways, the estimation overestimates the capacity. We argue that the better approach is to use interarrivals of Data packets. NDN data packets can use various payload lengths, and each payload length generates a different number of fragments. Using Data packet intervals requires to figure out the pattern in IC settings.

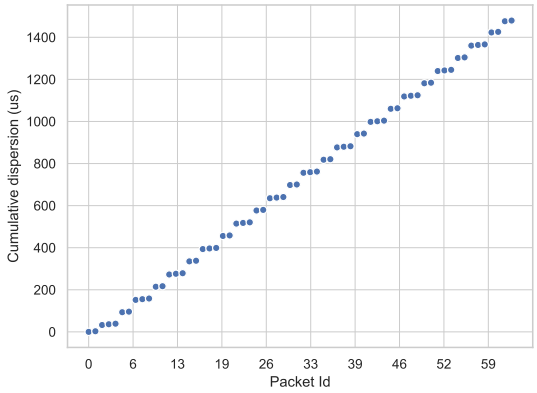
Since IC has the deterministic pattern, and each Data payload length generates a fixed number of fragments, we should also be able to find a pattern for Data packets. Figure 5.4 and Figure 5.5 show that the bursts produce new patterns when the Data payload length is 2780B and 4220B separately. The last frame of Data packets is marked with a red "x" in figures. We can see that the bursts generated by IC lead to a new pattern for Data packets. When the Data payload length is 2780B, the pattern has two Data packets in one burst and three Data packets in another (2-3). If the Data payload length is 4220B, the pattern changes to 1-2-2. In these complex patterns, the equation is still valid, but B should be changed to the length of the bursts in the pattern.

The dispersion must be long enough to capture the pattern. The minimum number of fragments to detect the pattern is predictable. Let Nf be the number of fragments that a Data packet could generate, and B be the length of the burst, then the required minimum number of fragments is $Nf * B$.

We also find that DPDK constantly introduces delays in leading packets when sending out a bursty of packets. We captured traces on both the client and the server. Interest packets are small, and the timestamp captures the time when a packet reaches the kernel. For Data packets, the timestamp is for the last Ethernet frame. Figure 5.6a show that DPDK introduces delays after the first Ethernet frame in both Interest pipeline and Data pipeline. For the rest Ethernet frames in the same burst, the delay is not distinct. When large payload lengths are used, we do not find such issues. This finding indicates that capacity estimation needs to exclude the leading packets

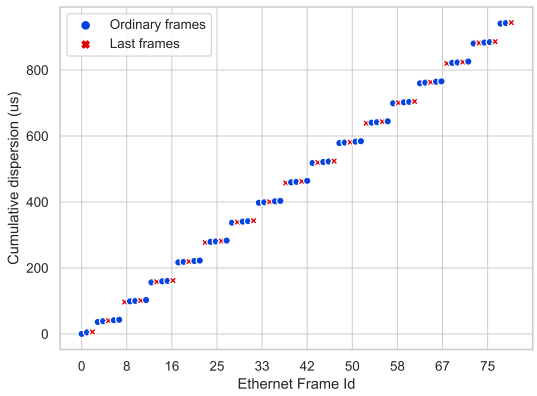


(a) Ethernet frame intervals with timeline.

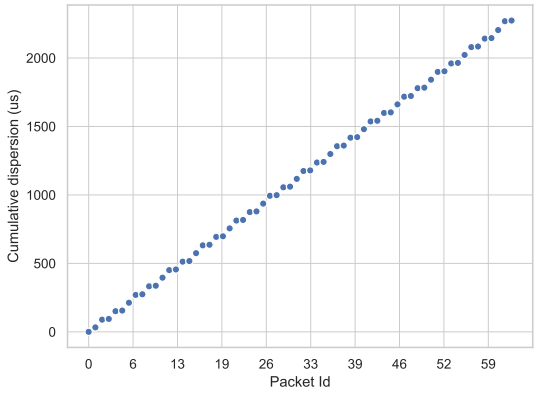


(b) Data packet intervals with timeline.

Figure 5.4: Observed packet time intervals for payload 2780B over 1G NIC.



(a) Ethernet frame intervals with timeline.



(b) Data packet intervals with timeline.

Figure 5.5: Observed packet time intervals for payload 4220B over 1G NIC.

when using DPDK to generate probe packets. In our experiments, a packet train of 64 packets is sufficient to detect the IC signature reliably.

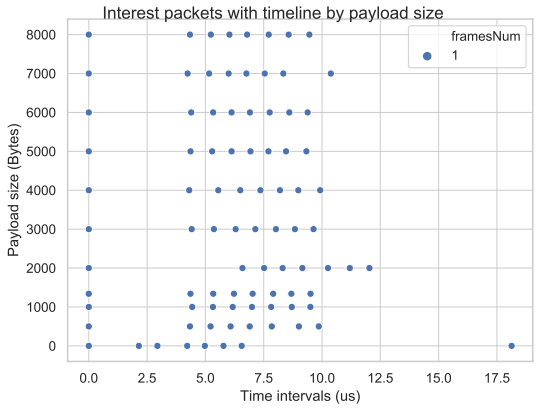
Finally, we use the formula in section 2.7.3 to estimate the end-to-end capacity. Let Δ be the dispersion between the first packet of two consecutive bursts. The capacity is $C = BL/\Delta$, where B is the length of the first burst, L is the Data packet size. Here, the burst is the sequence of Data packets that covers the pattern mentioned above. Our results show that the estimated end-to-end capacity is around 980Mbps, which is close to the 1Gbps capacity. The experiment results show that the packet pair techniques can estimate the end-to-end capacity on an NDN network over the NDN-DPDK forwarders with 1Gbps NIC. The rest of this chapter presents our investigation over the high-speed NICs.

5.4 Investigating the High Performance NIC

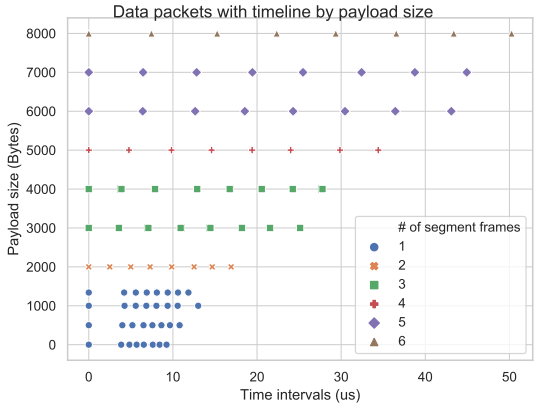
As shown in Chaper 4, the actual bandwidth of a name prefix over the DPDK forwarder is around 7Gbps. The experiments on 1Gbps NICs show that IP-based packet dispersion techniques are applicable when the network card is the bottleneck. This section demonstrates that the measurements from the end host detect variances in packet interarrival when 40Gbps NICs are used. The variances become the hurdle to apply the packet dispersion techniques.

Figure 5.3 gives the Data packet intervals with timeline on 40Gbps NICs. We find that the line contains several segments. To check if the 40Gbps NICs adopt IC, we compare the dispersion with timeline in two connection types. Again, Data payload length 1340B is used to ensure no fragmentation happens.

Figure 5.7a shows that the direct connection has a almost linear increase line with timeline. The DPDK stack delays the first packet to create a gap after the first packet. All the packets are transmitted within 83us. In contrast, forwarding the 64 packets over the DPDK forwarder uses about twice the time. The line is broken into several segments. The segment starts at random locations in the packet train, which contains a random number of packets. A segment has a large

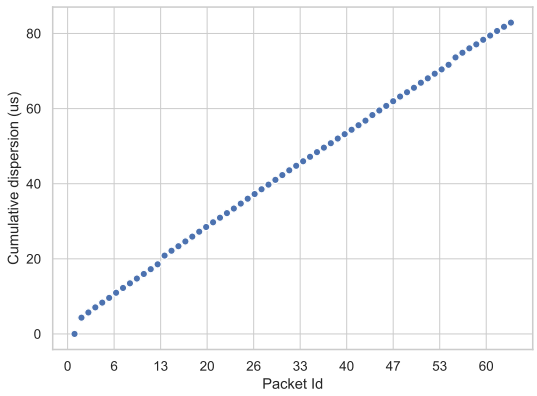


(a) Observed Interest packet time intervals over 40G NIC.

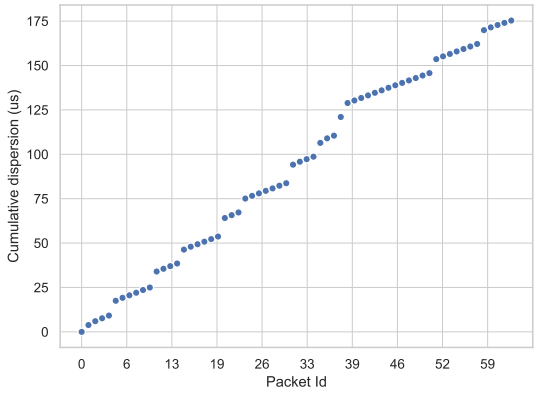


(b) Observed Data packet time intervals over 40G NIC.

Figure 5.6: Observed packet time intervals over 40G NIC.



(a) Direct Connection.



(b) Via DPDK forwarder.

Figure 5.7: Observed Data packet interarrivals for payload length 1340B over 40G NIC with various connections.

number of packets from packet id 38 to 50 (13 packets). Another burst only contains one packet, id 37. No distinct pattern exists in the packet train.

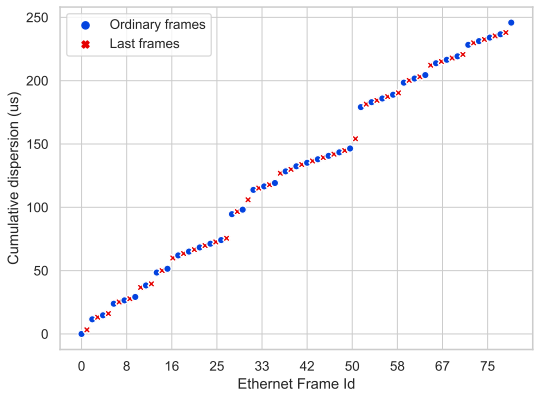
To verify if other payload lengths have the same issue, we plot the cumulative dispersion with the timeline in Figure 5.8. We mark the last frame of each Data packet with a red "x" in these figures, expecting to figure out potential patterns if any. Unfortunately, the results are similar to the payload length of 1340B. Bursts have random start positions, and the length of bursts has no pattern. Some red "x" symbols drop in the same burst, and packet intervals are small. Sometimes a symbol may stand out independently, as the frame id 63 in Figure 5.8c. With such randomness in the packet intervals, estimating the end-to-end capacity with packet dispersion techniques becomes difficult. Given that the testbed has no other traffic, we suspect the NDN-DPDK may have modules not well-implemented, or something else introduces variances.

5.5 Overhead of NDN-DPDK forwarder

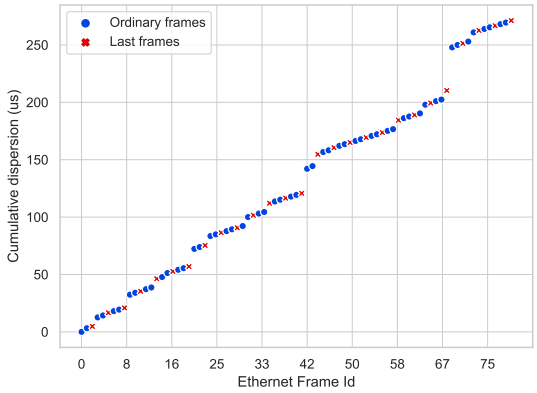
Previous results show that the NDN-DPDK forwarder introduces variances. To figure out the cause of such variances, we use two approaches. First, profilers can report the time spent on each module and help analyze the performance of DPDK forwarder. NDN packet processing is CPU intensive. Profiling the NDN-DPDK forwarder can help us identify bottlenecks in the pipeline. We can then analyze if these bottlenecks introduce variances. On the other hand, we can plot the packet intervals in figures. The results can disclose the relationships between the variances and other packet parameters.

5.5.1 Profiling forwarder

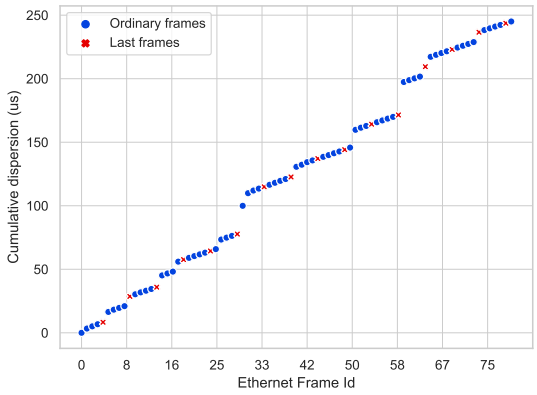
Intel VTune Amplifier performance profiler [120] is the perfect tool to profile and benchmark NDN-DPDK. The Intel processor provides performance counters to monitor events. After recompiling the DPDK with profiling flags enabled, VTune is attached to the DPDK forwarder. While the DPDK forwarder is running, VTune collects performance-related statistics through monitoring event counters.



(a) Payload Length = 2780B.



(b) Payload Length = 4220B.



(c) Payload Length = 7120B.

Figure 5.8: Observed Ethernet frame time intervals for various payload lengths over 40G NIC.

To start, we give the ndnping-DPDK benchmark module a broad interval range to keep the tool generating enough Interests. The generated traffic allows the VTune to collect sufficient samples to produce an accurate report. Based on the report, we plot the call graph figures for various Data payload lengths. Figure 5.5.1 is the call graph for payoad length 8000B.

In the call graph, each box is a function that is used during the forwarding. Most boxes are in blue, while boxes that consume more CPU time are in other colors, so the big consumers stand out in the graph. The arrow indicates the caller and callee. The number in the boxes is the cumulative percent, presenting CPU time for the functions. The cumulative percent in the parent box contains the cumulative percent of the child box. The number in the bracket is the absolute percent, which is the CPU time was spent in the function.

The data plane consists of two types of threads: input thread and forwarding thread. The Input Thread (FwInput) runs the main input loop (RxLoop_Run()), which reads and decodes packets from one or more network interfaces. The Forwarding Thread (FwFwd) runs the forwarding loop (TxLoop_Run()), which reads packets from its input queue and handles each packet separately. Figure 5.5.1 shows that forwarding (FwFwd_Run()) is CPU intensive, taking 73.46% CPU time. FwFwd_RxData uses 2.43% CPU time, and there is no Interest forwarding function (FwFwd_RxInterest()) shown in the figure. This is because Data packets are larger than Interest packets, and sending a Data packet is more expensive. Regarding the major structures, CS function (Cs_Insert()) is less than 1%, which is not critical. Compared with CS, PIT is more expensive, and it uses 1.67%.

The Userspace RCU library [121] helps the NDN-DPDK forwarder read packets. The URCU library claims to provide efficient data structures and lock-free algorithms so that read-side access scales linearly with the number of cores. As shown in Figure 5.5.1, the URCU library (urcu_qsbr_quiescent_state()) is one of the big consumer, more than 17% CPU time.

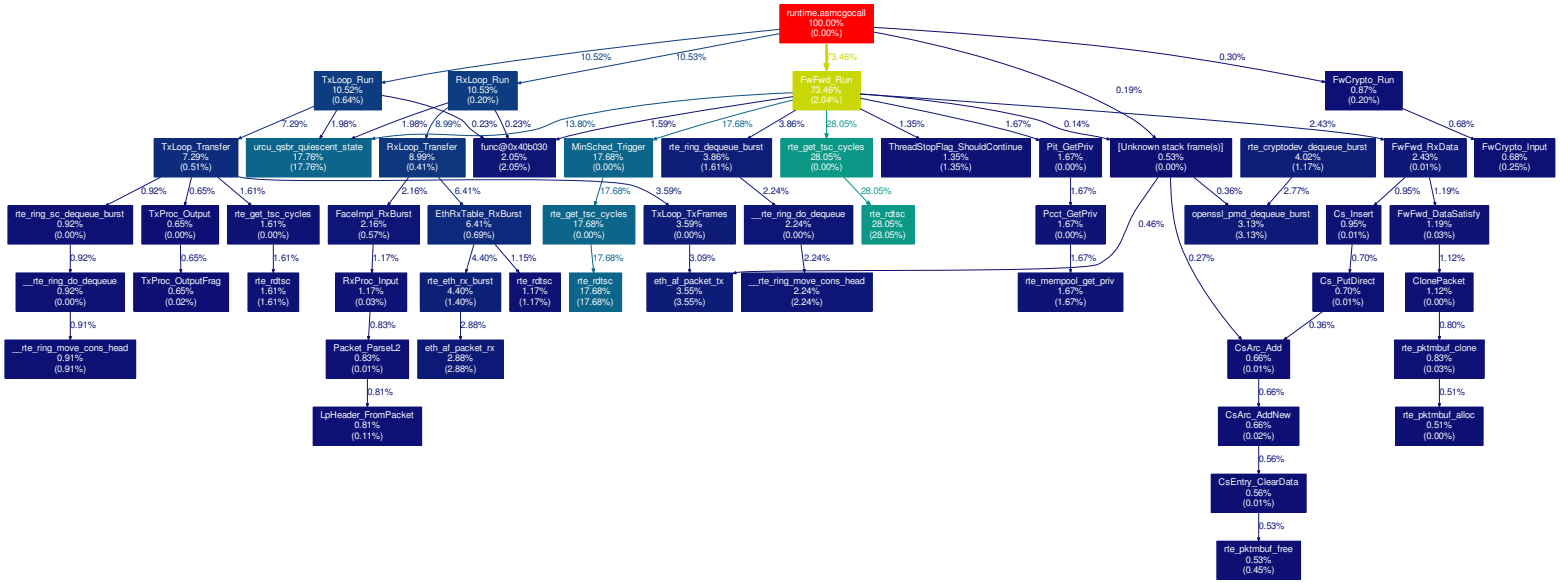


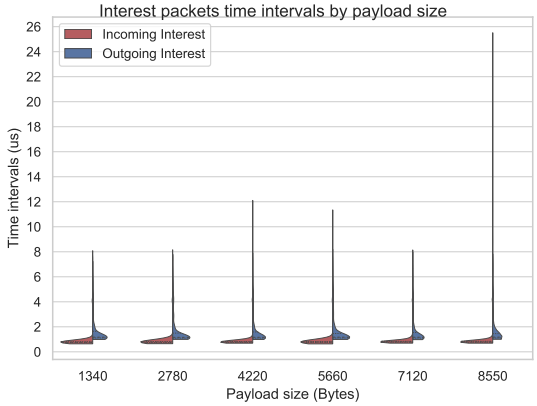
Figure 5.9: The call graph of NDN-DPDK for Data payload length 8000B.

The packet transmission loop and the receive loop uses around 10%, respectively. They are mainly for sending and receiving packets. The transmit loop (function TxLoop_Transfer()) takes around 7.29%, while the input loop (function RxLoop_Transfer()) is 8.99%. The input also spent 0.83% CPU time on the link-layer header parsing introduced by the hop-by-hop fragmentation.

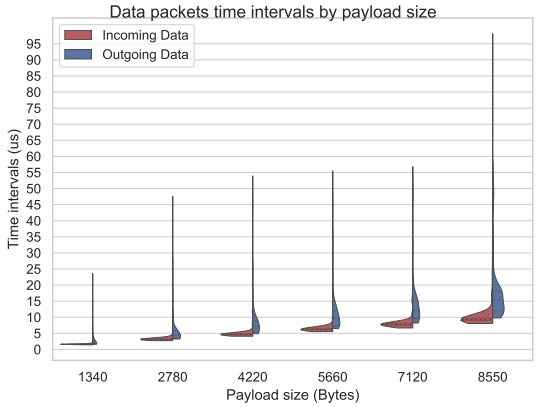
When a Data finds a PIT entry whose Interest carries the digest component, the forwarder needs to compute the Data's implicit digest in order to determine whether the Data satisfies the Interest. FwCrypto provides Data implicit digest computation in the forwarder's data plane. The function rte_cryptodev_dequeue_burst() box shows that it is one of the big consumers, around 4.02% CPU time. The dequeue operations seem to be more expensive. Specifically, dequeue functions like rte_ring_dequeue_burst() and openssl_pmd_dequeue_burst() are expensive in the experiment.

Another big consumer is MinSched_Trigger. It uses 17.68% CPU time, and it mainly spent in function rte_get_tsc_cycles(), which returns the number of TSC cycles for calculating elapsed time. The minute scheduler invokes MinSched_Trigger() periodically. The typical usage of the minute scheduler is to clean out inactive table entries, for example, when a PIT entry is expired. We also note that the function rte_get_tsc_cycles() is also called by other objects in the code, such as the NACK forwarding module, Ethernet face module, and Interest forwarding module. Heavily relying on the timestamps is an essential issue in the NDN forwarding. Figure 5.5.1 shows that the function rte_get_tsc_cycles() takes more than 28% CPU time. This result indicates using a better design to replace timer related operations may improve the forwarder performance.

The results about the other functions seem reasonable in the benchmark process. Time-related operations are the bottleneck in the NDN-DPDK design that could be improved in the future. However, the use of time-related functions should not introduce any variances. Something else is the cause of variances in packet interarrivals.



(a) Interest packet intervals.



(b) Data packet intervals.

Figure 5.10: Observed packet intervals on forwarder.

5.5.2 Effects on Packet Intervals

We use NDN-DPDK forwarder with AF_Packet to forward packets between the server and the client. In such settings, traffic needs to go through the kernel. The MTU is 1500B, and large Data packets must be fragmented before sending out.

Plotting the packet intervals with distributions before and after the DPDK forwarder can present the delay changes. Box plot is one option to give this information, but the Violin plot is more informative than a plain box plot. Violin plots are similar to box plots, except that they also show the probability density of the data at different values. After captured traces on both interfaces on the DPDK forwarder, we plot the violin plot for the Interest pipeline and the Data pipeline in Figure 5.10.

Figure 5.10a shows that the DPDK forwarder does introduce delays in the Interest pipeline. When the payload length is 1340B, the minimum Interest intervals are around 0.8us for Incoming Interests. While after the Interests go though the DPDK forwarder, the minimum Interest intervals are increased to 1us. This is reasonable, as the PIT table operations and FIB search introduce delays, about 0.2us in the results. The width of the violin plots indicates the probability density of the data at different values. Comparing the width of incoming Interests and outgoing Interests, we find that the intervals of outgoing Interests are larger than the incoming ones. Besides, we observe that most of the Interest packet intervals do not change with the payload lengths.

Figure 5.10b is more informative. Similar to the Interest pipeline, the minimal Data intervals are increased after packets go through the forwarder for all the payload lengths. That means that the DPDK forwarder introduces delays in the Data pipeline. An interesting finding is that the larger Data payload lengths have larger maximum data intervals. The hop-by-hop fragmentation may be the major contributor to this, as the forwarder must receive all the fragments before forwarding out the Data packet. Comparing the incoming and outgoing Data packets intervals, we can find that the width of the outgoing ones is smaller than the incoming ones. That indicates that the intervals in the outgoing Data packets are larger, which in some sense confirms the findings in Figure 5.8 that DPDK introduces bursts. The OS schedules may be one cause of bursts. Besides, the forwarder is

hard-coded with a forwarding burst size 64 in modules to send out packets. A large Data packet is fragmented into several Ethernet frames. These frames cannot be put in one burst, and thus the forwarder produces multiple bursts to send out the Data packets.

5.6 Summary

We present the work to investigate the feasibility of replicating the IP-based packet dispersion techniques in NDN networks. We devise the techniques to use NDN Data packet intervals for the estimation of end-to-end capacity. The interrupt coalescence (IC) groups the incoming packets into bursts with a fixed length. We propose to redefine the "burst" as the sequence of packets to cover a Data packet pattern. We show that using a packet train with tens of NDN packets should be able to detect the pattern. Our results show that the packet dispersion techniques can correctly estimate the capacity between the client and the server when intermediate NICs are the bottlenecks.

The measurements from end hosts indicate that the NDN-DPDK forwarder introduces variances in packet intervals. We study the potential causes by profiling and NDN-DPDK forwarder and plotting the intervals for the incoming and outgoing packets. We find that the NDN-DPDK design contains a potential bottleneck to excessively use time-related operations. That could help improve forwarder performance in future work. Based on the observation of packet intervals changes on the forwarder, we argue that the operating system scheduling and the fragmentation may be the causes of variances. Using dedicated NICs is a way to exclude scheduling overhead and avoid fragmentation. We plan to study the replication work on NDN-DPDK forwarder with dedicated NICs in the future.

Chapter 6

Detecting Caching Decision Mechanisms

Caching is critical to an NDN network's performance. In NDN networks, routers may cache pass-by Data packets in the cache storage. Any requests carrying the same content name can be satisfied by routers on the path.

The caching mechanism is beneficial to both end-users and servers. Caching improves the data retrieval performance and reduces the load on servers. Researchers proposed various caching mechanisms to improve caching efficiency, but none of them are best for all scenarios [57]. We believe that various caching mechanisms may exist in the same NDN network. These mechanisms may interact poorly, and misconfiguration may introduce performance degradation. Both operators and Internet users need the ability to detect caching mechanisms.

In this chapter, we present a novel caching detection approach to identify caching decisions. We first review the NDN caching process to clarify that caching mechanisms contain caching decisions and cache replacement. We use the distribution of data chunks that observed by the consumer as the profile of a caching decision. The results over ndnSIM [110] show that our method can successfully identify various caching decisions. The method does not need any help of in-network nodes and only uses a few probe packets. Any end-users can utilize our method to figure out the deployed caching decisions.

6.1 Introduction

Caching is one of the most critical components in NDN [1]. Caching mechanisms leverage the local cache storage in routers to eliminate the need to ask a producer for the same data. Specifically, since every NDN data packet contains a unique name and signature, NDN routers can cache pass-by Data packets in its local cache storage, the Content Store (CS). A router can then respond with the local copy to an incoming Interest that contains the same name, instead of forwarding the Interest to the original producer(s). Caching can decrease consumers' access latency. When

a mobile user is moving from one access point to another, he/she is able to fetch data from the in-network cache, without the need to ask the server for data [1]. Caching can also reduce loads on the server-side. When duplicate requests are satisfied on intermediate routers, the server can save the precious computational resources for other uses.

NDN caching is fundamentally different from the caching on IP networks. The present Internet has used caching in various scenarios. Web caching [122], P2P network caching [29, 30], and content delivery networks (CDN) [27, 28] are examples to utilize caching to reduce bandwidth consumption. However, all (or lots of) routers in NDN networks are equipped with CS. Unlike IP networks where identical objects lack unique identification, NDN networks adopt hierarchical names to identify each piece of data uniquely. These in-network storages form complicated total cache systems to provide pervasive caching. Consequently, ubiquity and fine-granularity have made traditional caching theories, models, and optimization techniques developed for hierarchical Web caching and CDN caching systems unable to be directly and seamlessly ported to NDN caches.

To improve the cache hit ratio and optimize the utilization of in-network cache, researchers are continuously exploring new caching mechanisms. For example, Li et al. proposed a cooperative caching mechanism designed for large video streams (e.g., live TV services) [62]. LCC [123] is a hash-based caching mechanism for inter-domain cooperation, while HSS [124] is designed for heterogeneous scenarios to allocate content routers different cache sizes based on certain criteria. The performance of these in-network caching systems depends on many factors, such as the popularity of contents, routing of content request packets, cache decision, and cache replacement policy. Each application scenario has its special requirements, and no caching mechanism can fit all the scenarios.

Therefore, it is likely that multiple caching mechanisms could be deployed in NDN networks to help data distribution. Moreover, such co-existence of caching mechanisms could help provide better service quality. For instance, users are more likely to notice the improvement of downloading latency of HTML pages rather than that of the embedded objects. Assigning higher caching

probability to HTML pages could help NDN networks efficiently use in-network storage without hurting users' service quality. In an NDN testbed, researchers could use their specific caching mechanisms to conduct experiments. Moreover, ISPs may have different contracts with different content providers to ensure some name prefixes have a better service for a negotiated price [125].

However, the co-existence of caching mechanisms may introduce management issues on NDN networks. Like other management, caching mechanisms management will be error-prone. Mistakenly assigning a high caching probability to embedded objects other than the HTML pages will waste precious cache resources and hurt user-perceived performance. Content providers may not gain advantages from higher caching probability due to an ISP's misconfiguration.

In this chapter, we present an approach to allow end-users to detect the deployed caching mechanism for a name prefix. We produce a unique profile for each caching mechanism in the simulation. We also show that the approach works well for a real topology.

To capture the profile for each caching mechanism, we let the client send out probe packets several rounds. From one round to another, the cached data chunks give us the change of cached chunk distribution. We use Violin Plot to capture the distribution of hops of cached data chunks as the profile.

To evaluate our method, we leverage ndnSIM on a real RocketFuel topology. The results show that simply using RTT information could identify some caching mechanisms. For other mechanisms where RTT is not enough, we apply the k-means clustering algorithm [126] to group samples with similar RTT. The generated results are enough to help identify caching mechanisms.

Our first contribution is to introduce a method to capture the profile of caching mechanisms. The method does not require help from any third-party. Users just need to send a small number of probe packets under a name prefix. Since NDN routers are stateful, the small volume of probe packets is essential to the success of measurement. Our method can also estimate the used probability when static probabilistic caching mechanisms are used. We use ndnSIM to evaluate the method on a real topology and point out the potential issues of the approach.

6.2 Detecting Caching Decisions

Unlike other work that develops NDN management protocols or platforms, we design our method without help from any third-party. Our method identifies caching decisions based on the insight that cached Data chunks move differently according to the caching rules. The difference can be perceived by the consumers when they sends out duplicate Interests.

6.2.1 Caching Strategies

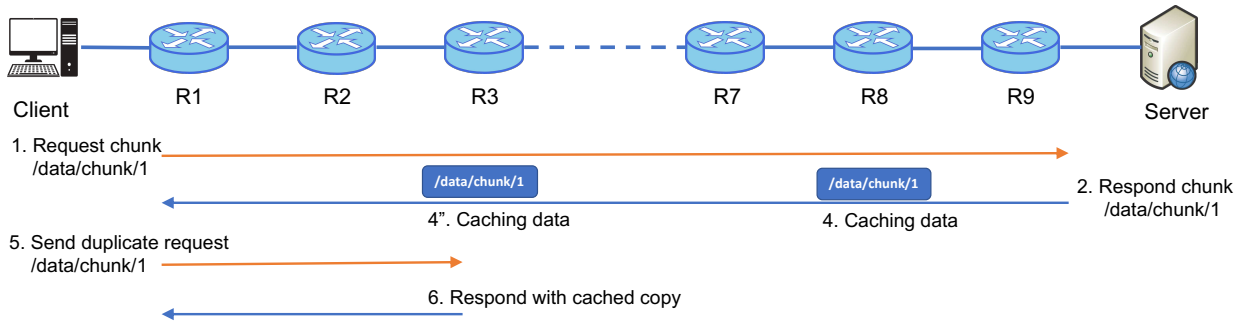


Figure 6.1: An overview of NDN caching mechanism.

As introduced in Section 2, caching is one of the most critical components in NDN architecture. Since each NDN Data packet carries a unique name and a signature, NDN routers can cache received Data packets in their Content Store (CS) and use the CS to satisfy future requests. Unlike IP networks where IP routers cannot reuse a packet after forwarding it to its destination, NDN routers treat storage and network channels identically in terms of data retrieval.

Figure 6.1 demonstrates the overview of NDN caching. When consumers (a.k.a clients) want to fetch a file, named "/data," they send out an Interest that carries the data name. If the file size is too large to put into one Data packet, the producer (a.k.a server) separates the actual bits and puts them into a sequence of Data packets. The Data packets that contain the sequence number are also called chunks. To get a chunk, "/data/chunk/1", the Interest must carry its name. Upon the arrival of the Interest, the producer responds with a chunk that contains the same name. Intermediate NDN routers then forward the chunk along the reverse path to the requester. Those routers are

configured with a specific caching mechanism, which will decide if the router should cache the pass-by Data packets or not. In the future, whenever a router receives an Interest that can be satisfied by the cached data, it immediately responds without forwarding the Interest to the next hop. NDN caching can benefit both the static file and dynamic content until the cached packets are stale or evicted. For static files, NDN achieves almost optimal data delivery. Even dynamic content can benefit from caching in the case of multicast (e.g., realtime teleconferencing) or retransmission after a packet loss.

Each caching strategy contains two components: the caching decision and the cache replacement decision. The caching decision is to decide whether or not an incoming content chunk is to be stored at the local CS. The cache replacement decision is to decide when and which cached Data chunk needs to be evicted. Cache replacement typically happens when a new Data chunk needs to be placed in a cache that has reached its capacity.

6.2.2 Objectives

In this work, we aim to detect the deployed caching decision for a prefix at the end-side for several reasons. First, caching decision happens more often than cache replacement. Caching decision happens whenever a Data chunk arrives at the NDN router, while cache replacement will be triggered only when a cache is full. Second, end-users have no tools or methods to detect caching decision misconfiguration. The misconfiguration will affect the performance to retrieve data. Specifically, the misconfiguration may happen when content providers require differentiated caching services. Mistakenly assigning a lower probability to a content provider who contracted with an ISP for higher probability violates the contract and lets other competitors gain advantages. Besides, one caching decision may not work well in some scenarios. For example, content providers prefer to cache chunks at the server side at the beginning, in order to utilize network storages to help serve a burst of requests. They would prefer the ProbCache-inv mechanism [53], instead of the built-in Cache Everything Everywhere (CEE) caching mechanism. Furthermore, to figure out the caching mechanisms, ISPs could access CS entries on each router, but it needs spe-

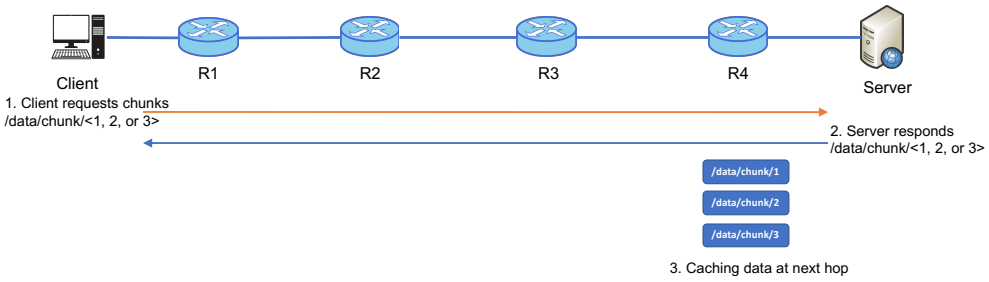
cial privileges or additional management protocols. Our goal, however, is to let end-users detect deployed caching mechanisms without any additional requirements.

To ensure that our measurement behaves correctly, we must minimize the probe packets. As pointed out in Section 2, NDN routers are stateful. It can save pass-by content chunks into the local CS to help future requests. Injecting too many probe packets into the NDN network will place pressure on the network state or even change the network state. Besides that, caching decision mechanisms may utilize any information to make decisions, such as local random numbers, topology information, data labels, or even traffic information. Our method should allow users to specify as much information as possible and observe the network's behavior under those conditions.

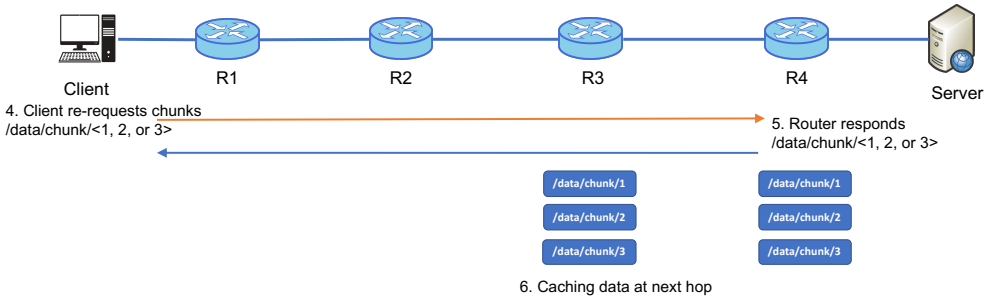
6.2.3 Methodology

We derive our method from the Leave Copy Down (LCD) caching mechanism [57]. As shown in Figure 6.2, the client sends out duplicate Interests in several rounds. At time t1, the client sends out Interests for data chunks 1, 2, and 3 for Content "/data." Those Interests are forwarded towards the server. Upon the arrival of those Interests, the server responds with corresponding Data chunks. Since NDN routers have LCD caching decisions deployed, the Data chunks carry a field to indicate the hop count away from the storage. According to the hop count field, the first-hop router will save all chunks locally. Other routers forward Data chunks downstream to the client. At time t2, the client sends out those duplicate Interests again. The first-hop router will respond with local copies. For the same reason, caching happens at the next router. Similar caching behavior happens at time t3 as well.

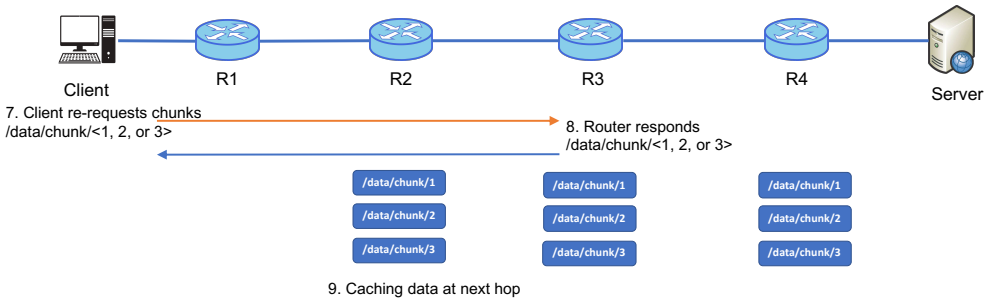
The state change in the whole process forms a unique profile for caching decisions. In each round, the client could perceive the hop changes of each chunk after it fetches them. The distribution of cached Data chunks demonstrates the feature of retrieval in one round. The LCD caching mechanism puts all chunks that belong to the same round in the same hop. From one round to another, LCD gradually move Data chunks from the upper router to lower routers. In the client's view, it can notice Data chunks in the second round have decremented hop counts. Therefore, the



(a) Caching state at time t1.



(b) Caching state at time t2.



(c) Caching state at time t3.

Figure 6.2: Caching state changes for LCD caching mechanism.

cached chunk distribution and its change between rounds form the profile for a caching decision mechanism. If the Data chunks have the same hop count in one round and the subsequent rounds have one hop less, then users can point out the deployed caching decision is LCD.

To facilitate the detection, we use Violin Plots to demonstrate both the cached chunk distribution and its change between rounds for several reasons. First, chunks are cached along the path, and they may appear on any routers that subject to the assigned probability. The formula is difficult to represent the chunk distribution. Second, other commonly-used plots cannot capture both the feature. For example, Box Plot is limited in its display of the data. Their visual simplicity tends to hide important details about how values in the data are distributed. However, a Violin Plot can show the distribution shape of the data. A Violin Plot is a combination of a Box Plot and a Density Plot that is rotated and placed on each side, to Similar to Box Plots, a Violin Plot has the median value, and the thick black bar in the center represents the interquartile range. The thin black line extended from it represents the upper (max) and lower (min) adjacent values in the data. Aside from showing the abovementioned statistics, a Violin Plot also shows the entire distribution of the data using the violin shape, which is the advantage of the Violin Plot over the Box Plot.

To gather information mentioned above, we let the client send out Interests under a name prefix, and the server respond with Data. We assume that users can access both the client node and the server node so that they can generate the Data packets with various parameters. Users can specificy name prefix, Data payload length, and other parameters to observe the caching behavior. We also assume that NDN routers use best-effort forwarding strategy, and no cache replacement happens. To capture the unique distribution of chunks, the client needs to send out tens of unique Interests. The small number ensures that no too much overhead are introduced. The client must repeats sending duplicate Interests for several rounds so that it can detect the state changes. To help detect cached chunks, we leverage the Interest parameters. Each Interest contains a unique parameter value, and the corresponding Data chunk encapsulates the value as payload. When the client receives the Data chunk, it needs to check if the parameter value matches. The mismatch

between the parameter value in Interest and Data indicates a cache hit. After the client collects all the Data chunks, we plot the Violin Plots for caching mechanism detection.

6.3 Empirical results

We use ndnSIM-2.7 to validate our caching decision detection method. We first use a simple linear topology (Figure 6.1) to capture the profile for some common caching decisions. The topology has enough hops to plot chunk distributions for those caching decisions. Then we investigate the effect of cross traffic and the feasibility to estimate the assigned probability for static probabilistic caching decisions. To study the effect of real topologies, we use a RocketFuel topology. The results show that our method works well for some caching decisions using just delay information. For those who do not work well, we group samples with the k-means clustering algorithm [126]. The k-means algorithm helps map the RTTs to hop counts, which then makes our method able to detect caching mechanisms correctly.

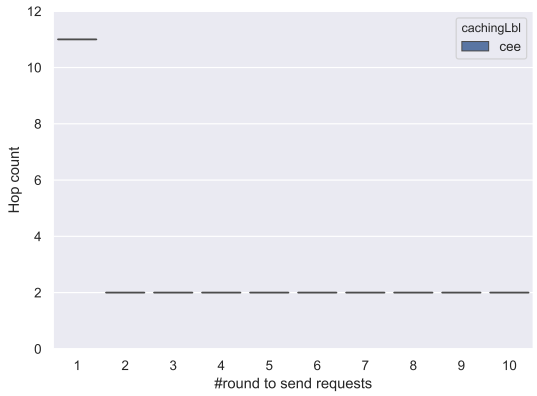
Our simulation assumes the content store is installed on all the routers except the first one, as that is usually the consumer's localhost. To simplify the scenario, we assume that there are only one consumer and one producer. The Best Route Strategy is configured on all the NDN nodes, as other forwarding strategies may introduce randomness. For each round, the client sends out 50 Interests as probe packets. The client repeats sending duplicate probing messages ten times to ensure that the state changes could be detected. Typically, all data chunks in the first round must be from the server-side. Since the second round, we expect data chunks to show the caching effect. After collecting all the data, we use Violin Plots to depict the state changes.

6.3.1 Caching Decision Profiles

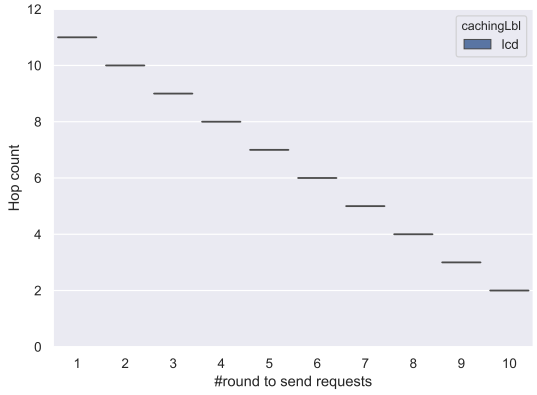
Since ndnSIM provides hop count information for each Data chunk, we use the Violin Plots to show the profiles. Figure 6.3 gives the profiles for CEE, LCD, and label-caching mechanisms [62]. When the LCD mechanism is deployed, content is always cached only at the next hop from the node where the cache hit occurred. Label-caching adopts the implicit coordination, and routers

follow a priori rules that govern what content they can cache. Specifically, each node is assigned a fixed label $l < k$ at setup and only caches content chunks whose IDs modulo k are equal to l. The purpose is to ensure that cached content is automatically stratified into equal subsets and evenly distributed across the network without the overhead of explicit coordination between nodes. To distinguish a caching decision from others, we need to check the chunk distribution in every round. Apparently, both CEE and LCD deterministically save Data chunks at a specific hop. Throughout the rounds, CEE caches chunks at the second hop, and cached copies never move after that. When LCD is used, chunks first stay at larger hop routers and then move to the next hop from one round to the next. In contrast, the label-caching mechanism cache chunks on multiple routers. In terms of state changes, the label-caching mechanism consistently keep chunks at specific hops. Duplicate Interests do not change the caching state. Figure 6.3c shows that all the rounds produce the same Violin Plot.

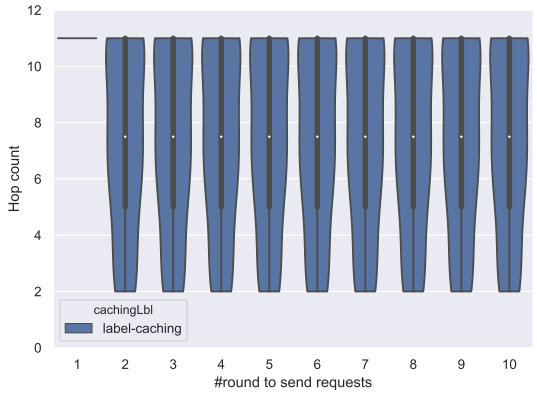
Static probabilistic caching is the easiest way to achieve higher cache diversity without increasing the complexity. It is important to investigate the profile for static probabilistic caching with various probability values. Figure 6.4 shows the profile of static probabilistic caching decisions with probability 20, 50, and 80, respectively. Comparing with CEE and LCD in Figure 6.3, a major difference is that cached chunks have various hop count when static probabilistic caching decisions are used. Those data disperse along the path with some probability and form a violin shape in our plots. With more and more duplicate Interests being sent out, the violin shape changes from one round to another. The first round always has a line, which indicates all chunks are from the server. The second round has the tallest violin shape in a specific probabilistic caching. With time goes on, the bottom of the violin shapes becomes wider and wider. That is because duplicate Interests let chunks go through the downstream routers, and they have a higher chance of being cached. The violin shape change between rounds also represents the static probability. When probability 80 is used, data chunks are quickly cached at the closest hop. Probability 50 makes the process much slower. Even when most chunks are cached one hop away, it still takes six rounds to cache all chunks at the closest hop. Probability 20 is the worst one in terms of the time to cache all



(a) CEE



(b) LCD

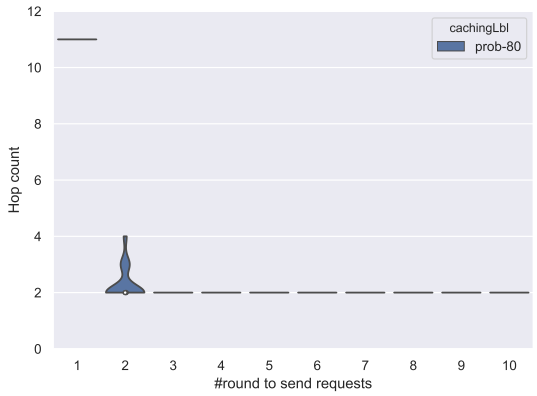


(c) Label-caching

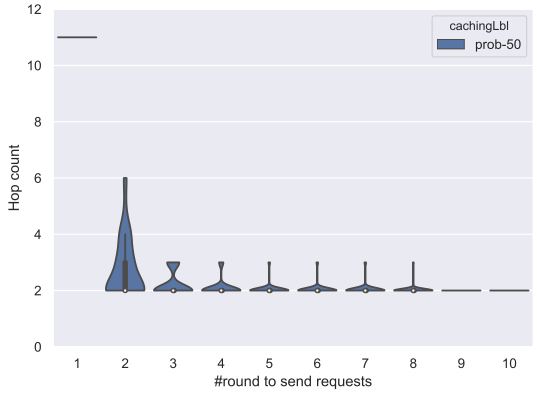
Figure 6.3: Profiles for various caching decisions.

chunks at the closest router. After ten rounds, there are still some chunks that are not cached at the hop-two router. We also note that the height of the interquartile range for round two indicates the pre-defined static probability value. We present the approach to estimating the probability value in the next subsection.

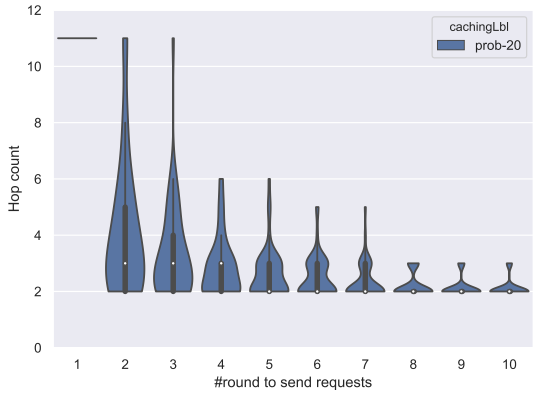
Instead of defining a priori caching probability that is the same for every caching decision at every node, researchers also proposed techniques that dynamically computes a caching probability for each individual node or even for each content chunk. The calculation is based on available information to adapt the caching behavior to the state of the network. ProbCache [53] is one of such caching mechanisms introduced in Chapter 2. It computes the caching probability of a given content chunk based on the total number of hops between its producer and the consumer that requested it, as well as the number of hops remaining on the path to the consumer. In other words, the caching probability is larger when a chunk gets closer to the consumer in each round. Figure 6.5a shows that the height of the interquartile range for ProbCache is similar to static probability 80 when the client sends duplicate Interests for the first time. That is because the ProbCache caching mechanism assigns a larger probability to routers that is closer to the client when the path is long. Apart from static probabilistic caching mechanisms, ProbCache recalculates the probability value for each round. We can find that caching all chunks at the closest hops, which is hop 2, takes another four rounds. The calculation rule makes the probability value drop when most chunks are cached around. If a chunk is cached at the 3rd hop in our topology, the probability to be cached at the 2nd hop is 2/3, which is less than static probability 80. ProbCache-inv is identical to ProbCache in every way except that the final caching probability is inverted $(1 - CacheWeight)$. Comparing with ProbCache, ProbCache-inv has a taller violin shape. ProbCache-inv tries to chunks closer to the producer, so the probability drops when chunks get closer to the client. From the interquartile range at the 2nd round, ProbCache-inv is like a static probability between 50 and 20. However, ProbCache-inv assigns a higher probability when the router is close to the Data sink (cache or producer). Therefore, ProbCache-inv takes fewer rounds to cache all chunks at hop two.



(a) Prob-80

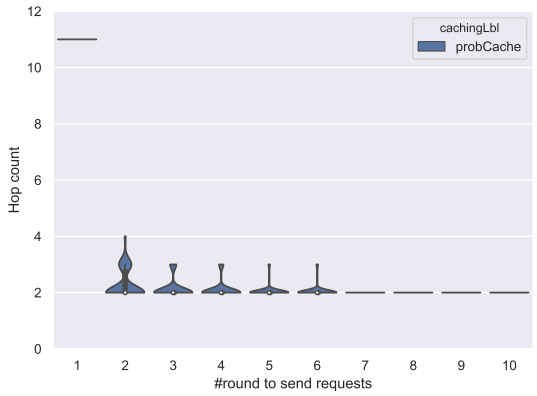


(b) Prob-50

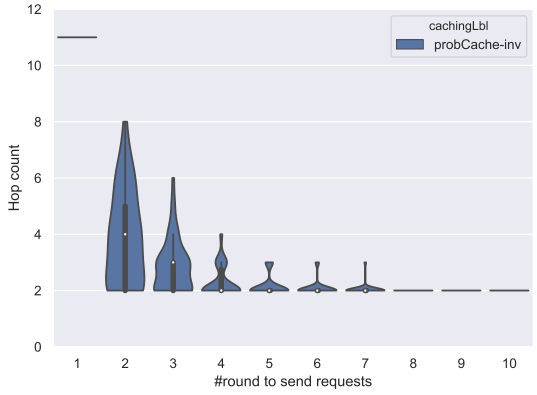


(c) Prob-20

Figure 6.4: Profiles for various static probabilistic caching decisions.



(a) ProbCache



(b) ProbCache-inv

Figure 6.5: Profiles for various caching decisions.

6.3.2 Estimation of static probability

Static probabilistic caching is widely used due to its simplicity to improve cache diversity across the network. It treats all chunks equally with the pre-assigned probability. To each Data chunk, the router generates a random number to decide if it should cache the chunk. An important implication is that more "popular" content — i.e., content that is requested and sent more frequently - has a higher chance of being stored at more points in the network.

Estimating the pre-assigned probability is of importance to both consumers and producers. Static probabilistic caching may accept any probability value between 0 and 1. The misconfiguration of probability value may dramatically change the behavior of caching behavior. Having a precise probability number could help end-users predict application performance or debug applications. Specifically, if end-users know the pre-assigned probability p, they could easily estimate the probability of a chunk to be saved is $1 - (1 - p)^n$ when a router receives the same one n times.

To estimate the static probability, we could leverage the Violin Plot as well. Let n be the number of chunks, p be the assigned probability throughout the network, i be the hop number that starts from the client toward the server. Then, we could calculate the percentage of chunks on i-th router as Formular 6.1.

$$p_i = (1 - \sum_{m=1}^{i-1})p \tag{6.1}$$

After getting all the p_i, we can estimate the number of chunks that are supposed to be cached on each router as np_i. Then, it is straightforward to plot a Violin Plot using the numbers of cached chunks on a router.

Using the above method, we plot the profile for the three static probabilistic caching in Figure 6.6. The plotted profile is a Violin Plot that shows the ideal case of distribution of cached chunks for round two. To make it clear, the number of cached chunks at a hop is also annotated aside the violin shape. Figure 6.6 shows that the median for probability 20 is around four, while probability 50 and 80 are zero. The interquartile range in those three probabilistic caching decisions is different too. Probability 20 has the longest interquartile range, while the interquartile

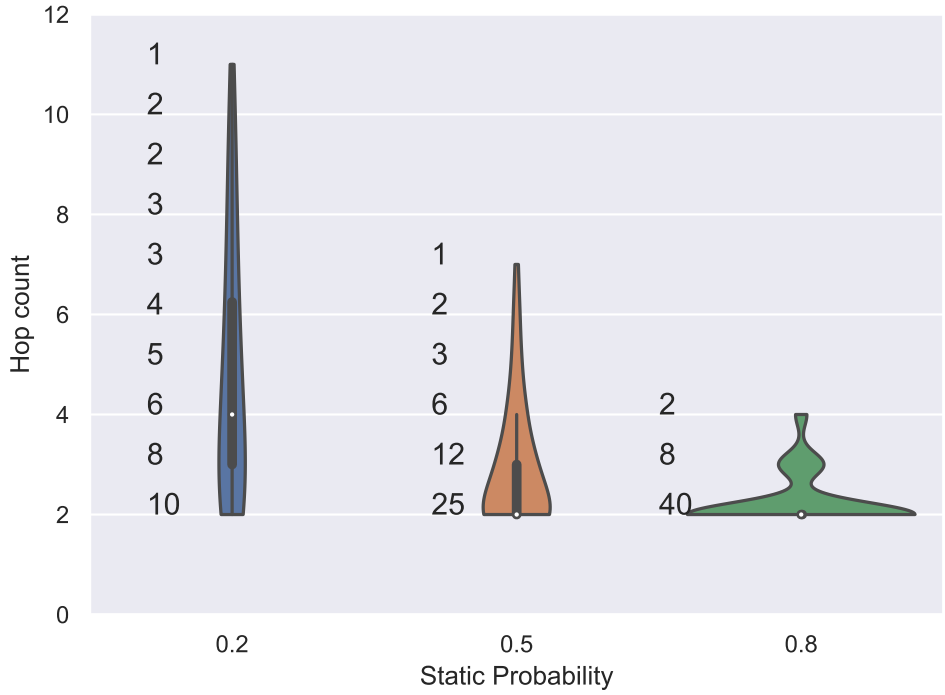


Figure 6.6: Estimated profile for three staic probabilistic caching.

range for probability 50 is much shorter Probability 80 has most chunks cached at the closest router, and thus the range height is zero. Comparing Figure 6.6 with Figure 6.4, we can find that there are slight differences between the simulation results and the ideal case. For example, the median for probability 20 in Figure fig:caching:mechanism-profiles-part-2 is three. The bottom of probability 20's interquartile range is two in our simulations, while the ideal case is three. However, the median, the interquartile range, and together with the violin shape form a unique fingerprint for caching mechanisms. The feature helps us identify static probabilistic caching mechanisms with pre-assigned values. Comparing to Figure 6.6, we can easily identify the probability 20, 50, and 80 in Figure 6.4.

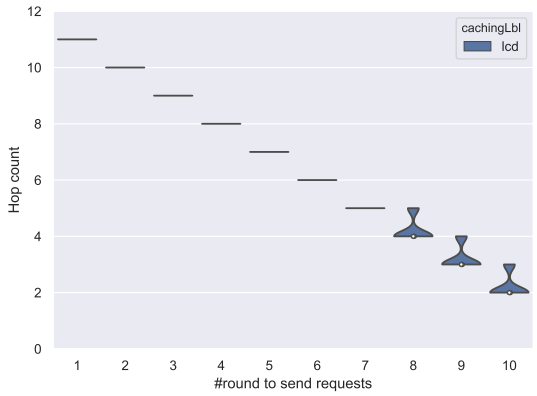
6.3.3 The effect of cross traffic

Traffic sent by other applications may lead to competition on shared network resources (bandwidth, content store, and others). Competition on the bandwidth will trigger more packet drops. A large volume of data in the same direction (between client and server) may use out of the content

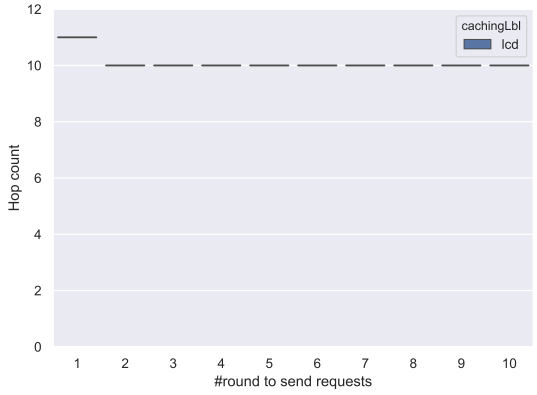
store and trigger cache replacement events. Competition in the same direction is the worst case, as our method needs cached chunks to plot reasonable good plots. In this chapter, we assume that such worst-case does not happen.

We are more interested in the correctness of our method when encountering cross traffic, as it is more common in the network. To introduce cross traffic, we attach the traffic generator at router eight and three in Figure 6.1. The traffic generator produces enough traffic to overload the router. Cross-traffic could happen at any router, but we believe putting traffic on both the server-side and the client-side could help us understand its effect separately. In our simulation, we only turn on one cross traffic at a time. To let cache replacement happen more often, we set the CS size to be 100.

The results show that LCD is the caching decision that could be affected by cross traffic. Figure 6.7 demonstrates that LCD cannot move forward to lower hops after hop nine (client node is the first hop) when the cross traffic happens at router eight. LCD attaches tags that initialized to one when a cached chunk is sending out. If the Data chunk is evicted, LCD cannot attach such information, and the next-hop cannot receive a chunk that contains a cache signal. When the cross traffic happens at router three, the competition is at hop four. We can see that samples are not stuck at hop four, but round 8, 9, and 10 contain samples with deviations. In this case, duplicate Interests arrive before the cache replacement happens for most data chunks. The competition happens close to the client. The delay between the client and the router is small, and thus duplicate Interests get a chance to reach the router before eviction happens. In contrast, the delay between the client and the router eight is much larger. When Interests arrive, cross traffic has evicted all the chunks in the CS. However, the Violin Plot can still detect LCD mechanisms. Other mechanisms are not so sensitive to cross traffic. Some mechanisms leave more data chunks at the server-side, and thus they have taller interquartile range, but that does not affect the detection.



(a) Cross traffic at client side



(b) Cross traffic at server side

Figure 6.7: Profiles for LCD mechanism when cross traffic happens.

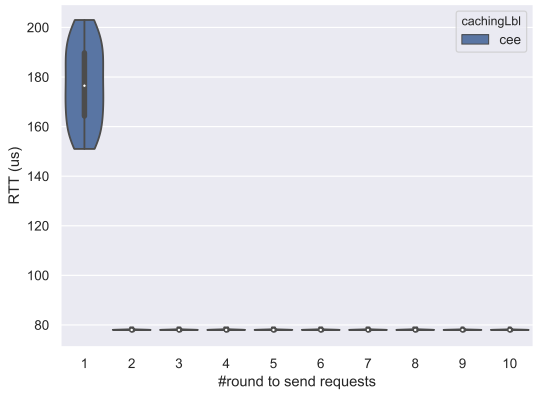
6.3.4 Detecting caching decisions on real topologies

This subsection presents the method to detect caching decisions on real topologies. The previous subsection shows that using hop information with Violin Plot could profile a caching decision. The issue is that although ndnSIM provides hop counts, the NDN stack does not explicitly expose the hop information to applications.

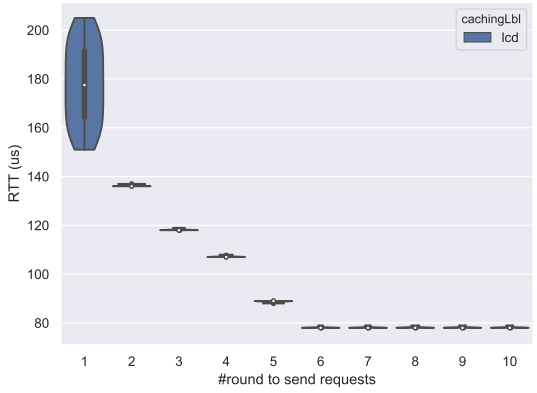
NDN application client could use HopLimit to figure out how many hops it needs to fetch a specific chunk, but it may slow down the measurement and introduce a lot of overhead. NDN Interests have an field called HopLimit to limit the number of hops the Interest is allowed to be forwarded. In order to figure out hop counts of a chunk, the client needs to send out an Interest with HopLimit one. If no data is received, it increments HopLimit to two and sends out the same Interest again until there is a data packet comes back. This approach has two issues. First, the client must send Interests to cover all hops until an Interest reaches the router that contains the data packet. Most Interests are wasted without any Data chunks returned. Those Interests introduce overhead not only into the bandwidth but also the PIT. Second, the measurement with HopLimit will be time-consuming. Interest has to wait until it times out when no Data comes back. There is no way to speed up the measurement process. To this end, we use RTT as the indicator of hops in our simulation. When RTT does not work, we apply k-means [126] to estimate hop counts for chunks.

We simulate the measurement process using ndnSIM on Rocketfuel topology 7018 [127]. We do not introduce other traffic in the network, as the point of this simulation is not for figuring out the effect of cross traffic. The simulation contains just one client and one server to exchange messages. They are randomly assigned to two nodes in the topology. Similar to the method mentioned before, the client sends out Interests to fetch Data chunks. When data chunks arrive, the client calculates the RTT. After that, we use the RTT information to plot the Violin Plot.

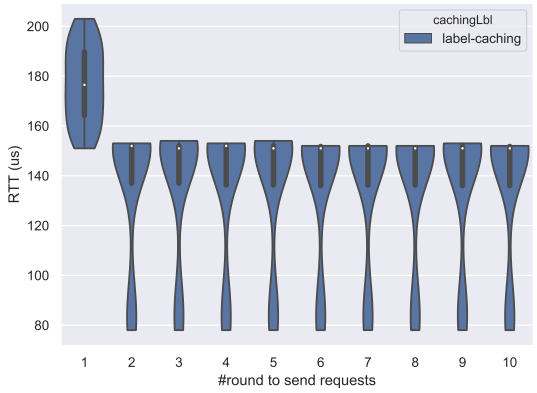
Figure 6.8 shows that simply using RTT in ViolinPlot could be good enough to identify some caching decisions. The delays for chunks from the producer varies, but they do not affect us to distinguish caching mechanisms. Figure 6.8a clearly points out that CEE keeps all chunks at



(a) CEE



(b) LCD



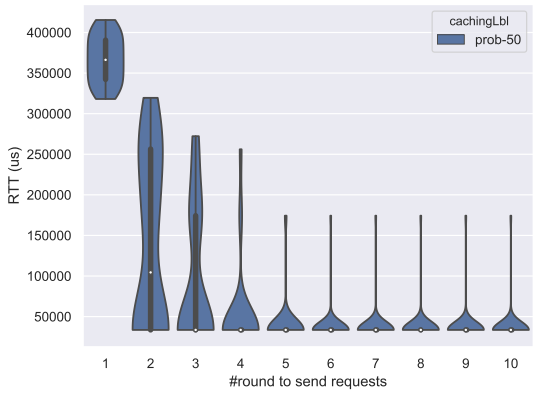
(c) Label-caching

Figure 6.8: Profiles for various caching decisions using RTT.

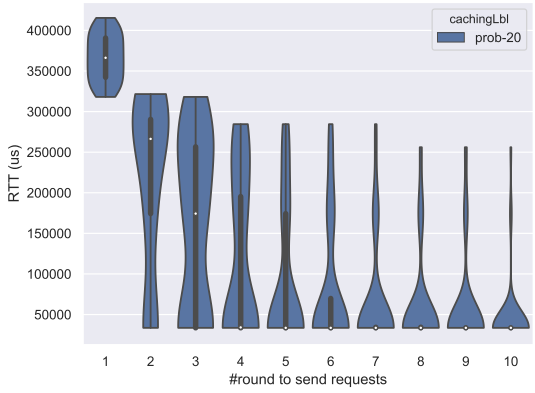
the closest hop since round two. The caching state never changes after that. Just like using hop counts, LCD moves data chunks hop by hop towards the client when using RTT (Figure 6.8b). Some samples have slightly different deviations, but they do not change the plot shapes too much. Finally, Figure 6.8c demonstrates a similar violin shape for all rounds, which indicates the caching decisions do not change when encountering duplicate Interests. Among all the caching decisions, we know the label-caching mechanism is the one who has that profile.

In some other cases, RTT may not be good enough to identify the deployed caching decisions. Figure 6.9 shows that the generated ViolinPlots for static probabilistic caching are misleading. The reason is that some links have small delays, while others have large delays. RTT values group some samples visually, but they cannot represent the hop counts correctly. We argue that the samples in the same group are from the same router. The rationale behind this is that the chunks from the same router go through the same links, and the Interests to pull these chunks use the same path. After grouping all samples, we can rank groups by their RTT values, and then each group can represent a hop. To this end, we apply the k-means clustering algorithm [126] to group samples. The k-means algorithm takes the collected RTT data and the target cluster number k as input, splitting the data into a fixed number (k) of clusters. The algorithm yields a cluster id associated with each sample in the data. We can then sort the clusters by the median RTTs. Each cluster is assigned a hop number, starting from hop one. Specifically, in our experiments, we specify six as the k value, and the generated plots present the reasonably good estimated hop counts. Figure 6.10 shows that the generated plots are similar to the ones in ideal cases (Figure 6.4).

In summary, we claim that using RTT with the k-means algorithm in our method is enough to identify caching decisions on the real topology. We are aware that the k-means clustering algorithm has the difficulty of deciding perfect k-value. The plot shapes generated by incorrect k-value is misleading in caching policy detection. For example, when the probability value 80 is used, the static probabilistic caching decision saves chunks on the first four hops from the client. In this case, we cannot produce the correct shapes with k-value seven. Fortunately, the collected data contains RTT information for chunks. As long as we know the link delays as prior, we can estimate the



(a) Prob-50



(b) Prob-20

Figure 6.9: Profiles for various static probabilistic caching decisions.

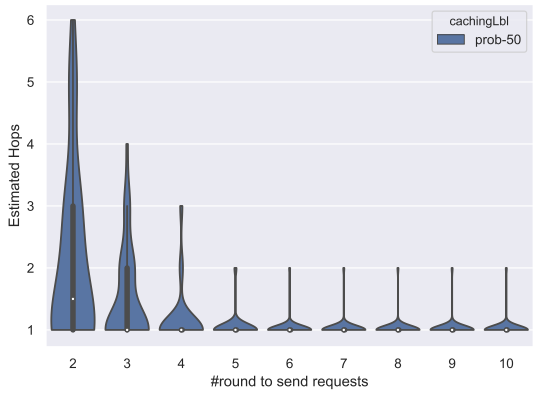
range of hop counts as the k-value. NDN-trace [19] is a perfect candidate tool to figure out the hop count and the link delays between the client and the server.

Many other clustering algorithms are available to group samples with similar delays in the same clusters. These clustering algorithms may be based on the partition, the hierarchy, the density, and other methods [128]. The partition-based clustering algorithms organize the data into non-hierarchical groups. K-means [126] and K-medoids [129] are the most famous ones. Hierarchical clustering constructs the hierarchical relationship among data [130]. Users can then cut the hierarchical-structure at the right level to get a number of groups. CURE [131] and BIRCH [132] are examples of this type of algorithms. Density-based clustering algorithms, including Mean-shift [133] and DBSCAN [134], adopt the method to group the data that is in the region with a high density of the data space. This type of algorithm has difficulty in handling data with varying densities and high dimensions. Depending on the scenarios, some algorithms may be better than others [128]. For example, both k-means and CURE can be used for large-scale data, but k-means is not good at handling high dimensional data. Comparing with the two algorithms, Mean-shift is not so sensitive to the sequence of inputting data.

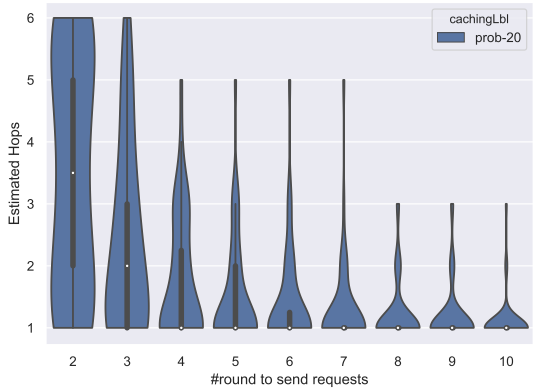
We can use any available algorithms for estimating the hop counts. Some clustering algorithms are designed to improve scalability. Some other algorithms focus on large-scale data processing or high dimensional data processing. However, our experiments do not require high scalability. The dataset size in our experiments is small, typically hundreds of samples. Delay is the only dimension that the algorithms need to support.

6.4 Summary

The new networking paradigm introduced by NDN, in particular, its stateful data plane with caching and name-based forwarding, require a solution to detect caching mechanisms. In this chapter, we present a method that leverages ViolinPlot to detect the working caching decision. Our method introduces small overhead to capture the profile of caching decisions. Specifically, the client sends out a small number of Interests to request Data packets under the target name prefix.



(a) Prob-50



(b) Prob-20

Figure 6.10: Profiles for various static probabilistic caching decisions with estimated hops.

After repeating the steps several rounds, the client can collect delay information for chunks to produce the representative ViolinPlot. The ViolinPlot contains information such as the hop counts and rounds to uniquely identify a caching decision. We show that our method can estimate the probability value for static probabilistic caching mechanisms, and it is robust to cross traffic. We also study our method on the real topology. Our results demonstrate that the method can map delays to hop counts to identify caching decisions.

While we initially evaluated the approach through simulation, we intend to apply it to NDN testbed for further study. We also notice that NDN's networking paradigm is much more complicated than our assumption. We have investigated the Best-route forwarding strategy in our simulation, but our method may not work well when other strategies (e.g., Multicast, Load-balance, etc.) are used. Besides, researchers have proposed many various caching decision mechanisms. This section does not cover all of them. Specifically, we do not look into explicit cooperative caching mechanisms. They usually exchange information about their cache contents and/or even forward content chunks to one another for caching. Our approach is working at both the end-host side so that it cannot identify the explicit coordination. Some caching decisions use other information, such as betweenness centrality of the caching node and content popularity, to help improve caching effectiveness. We plan to use our method to investigate those caching decisions in the future.

Chapter 7

Future Work and Conclusions

In this chapter, we present possible directions for future work and conclude this dissertation.

7.1 Future Work

NDN measurement research is at its early stage. Several directions could expand, enrich, and strengthen our studies. We discuss the possible future directions related to our studies in this section.

In Chapter 3, we propose the first conceptual measurement framework for NDN networks. We think NDN as a superset of IP in terms of functionality, and the framework is the first trial to establish connections between NDN and IP to help identify similarities and differences. However, the framework is an initial work, which only provides the general picture for NDN measurement. There are several directions for future work that can enrich this framework. First, we can extend the framework to include IP measurements. Since NDN is a superset of IP, we envision our framework can include the well-established IP measurement. The framework should be able to point out the differences between the two architecture. Such a framework can better help researchers identify the similarities and differences in measurements. On the other hand, we can enrich the study by including other existing work. Some researchers [135, 136] presents a comparative measurement study between NDN and IP-based protocols. We expect the framework to include them. Other works [137–139] utilize in-network measurements to help fight against DDOS attacks should also be included.

In Chapter 4, we present the first study for data retrieval over the NDN-DPDK forwarder on a dedicated testbed. This work focuses on the study of performance for a name prefix and the effects of NDN packet parameters on the performance. No cross traffic exists in the testbed in our experiments. We use AF_Packet in our experiments, so all packets go through the kernel. There are several directions for a better understanding of NDN-DPDK performance. First of all, it is

worthy of verifying our findings with dedicated NICs. Using dedicated NICs allows the forwarder to use the jumbo frame, an Ethernet frame with more than 1500 bytes of payload. Since DPDK can directly talk to dedicated NICs, using dedicated NICs eliminates the overhead of scheduling on operating systems. Additionally, we can extend our study by introducing cross traffic. We can study the performance reported by the ndnping-dpdk tool in the presence of cross traffic. Quantifying the overhead can help us show the efficiency of using well-designed measurement tools.

In Chapter 5, we present the first effort to investigate the feasibility of replicating the IP-based packet dispersion techniques in NDN networks. The results show that packet dispersion techniques could estimate the capacity between two ends when the 1Gbps NICs are used. When the high-speed NIC is used, the scheduling overhead and the hop-by-hop fragmentation introduce delay variance between packets. In the future, we plan to use dedicated NICs in our experiments. Using dedicated NICs allows the forwarder to talk to NICs without suffering from the scheduling overhead. In this case, the forwarder can avoid the hop-by-hop fragmentation by using jumbo Ethernet frames. We need to study how packet dispersion techniques work in such settings. The second direction is to propose approaches to estimate the path capacity in a network built over NDN-DPDK. Users may deploy the NDN-DPDK forwarder on various hardware, and the scheduling may introduce uncertainty. We need a capacity estimation approach to take this overhead into account. Finally, we need to revisit the design of NDN-DPDK to remove the potential bottlenecks. Specifically, we profile the NDN-DPDK forwarder to identify that time-related functions are the bottlenecks. Such a forwarder may not work well in the backbone networks. Revisiting its design to fit the high-performance requirements is necessary.

In Chapter 6, we propose a brand new approach to detect the active caching mechanisms under a name prefix. Our simulation results show that the approach can detect various caching decision mechanisms in several rounds. There are several future work directions to extend our work. First, we plan to detect deployed caching decision mechanisms on the real testbed. Simulation environments lack real-world information, such as layer interactions and others. Detecting caching

decisions on a testbed allows us to use real-world devices to validate our method. Second, one tool may not be able to generate meaningful information about the insight of networks. We plan to integrate our tool with the NDN measurement framework [115], which is proposed by the National Institute of Standards and Technology (NIST). Our tool can identify the active caching decisions for target name prefixes, providing the framework with the ability to detect caching decisions. In that direction, the framework can generate reports about the NDN network in various aspects, helping users identify and troubleshoot network issues. Third, we can enrich our work by extracting profiles for other caching decision mechanisms. In this dissertation, we produce fingerprints for some common caching decisions. Other mechanisms may be deployed in NDN networks. Providing more fingerprints can help detect cache decisions in the future. Moreover, various caching decisions may be mistakenly configured on different NDN routers for the same name prefix. The mixed-use of caching decisions may result in conflicts in saving chunks. Comparing measurements with the ideal fingerprints has the potential to identify misconfigured policies. We envision that our method could help troubleshoot the misconfiguration in NDN networks. Finally, we assume that one producer, one consumer, and the best-route forwarding strategy are used. In NDN, however, multiple producers, and various forwarding strategies (e.g., multicast, anycast, etc.) may be deployed. We have to evaluate our method in these complex scenarios.

Besides the immediate future work closely related to our studies, this dissertation suggests new directions for NDN measurement research. In our work, we choose IP-based packet dispersion techniques as an example to demonstrate the feasibility of replicating IP-based measurement approaches in NDN networks (Chapter 5). IP networks have a set of measurement tools and approaches, and each may fit specific measurement requirements in NDN networks. We can explore the ways to replicate other IP-based tools in NDN in the future. Similarly, we prefer to detect active caching configurations from end hosts in NDN networks (Chapter 6). Network state could be topologies, routes, and configurations. In NDN networks, not only caching mechanisms but also forwarding strategies maintain state on routers. We expect future research work can detect these network states.

7.2 Conclusions

Named Data Networking (NDN) is a future Internet architecture that addresses the content directly rather than addressing servers. NDN has shown promising results in a range of applications, including large-data applications [3], building automation systems [4], vehicular networks [5], and IoT applications [6]. However, deploying NDN in the real-world is not easy. One challenge is network measurement. Unlike IP that has a set of measurement approaches and tools, NDN only has a few achievements. Lacking sufficient measurement tools becomes a hurdle to deploy NDN networks.

To start, we proposed the first conceptual measurement framework for NDN networks by comparing IP and NDN in Chapter 3. Using tables, we showed that NDN is a superset of IP in terms of functionality. Based on this finding, we pointed out NDN and IP differ in many ways, but they share similarities at the same time. The framework inspired us to list the available tools for measurement targets in both IP and NDN networks. The comparison identified research gaps in NDN measurement from end hosts and motivated our thesis statement. We then presented our work to support the statement in two directions.

This dissertation hypothesized that *we can capture a substantial amount of useful and actionable measurements of NDN networks from end hosts*. To demonstrate the thesis statement, we presented a work to replicate IP-based approaches on NDN to estimate the end-to-end capacity (Chapter 5), and a new method to detect active caching decision mechanisms (Chapter 6). In both works, end hosts only rely on the probe packets to collect useful information about internal states in NDN networks.

The similarity between IP and NDN motivated our first work, which is to replicate IP-based measurements in NDN networks. We chose an IP measurement, packet dispersion technique, and studied it in NDN environments. This IP measurement allows end hosts to estimate path capacity using a small number of probe packets. In Chapter 4, we studied the data retrieval performance on a dedicated testbed, which is built using the NDN-DPDK forwarder, with various NDN packet parameters. The results confirm prior work that longer name components decrease forwarding per-

formance. Our experiments also show that hop-by-hop fragmentation introduces additional delays. Applications must carefully choose the Data payload size to gain better data retrieval performance. The results in Chapter 4 help understand data retrieval performance over NDN-DPDK, and it also serves as a baseline for validating the correctness of packet dispersion techniques in NDN networks. In Chapter 5, we present our work to replicate packet dispersion techniques. We devised the packet dispersion techniques to use the interarrivals of NDN Data packets for capacity estimation. Our measurements are conducted on two types of NICs, 1Gbps and 40Gbps, respectively. The results demonstrated that packet dispersion techniques work well on 1Gbps NICs. The statistics on end hosts showed that the NDN-DPDK forwarder introduces variances in interarrivals when the NIC is high-speed (40Gbps). The major contributors may be the operating system scheduling and the hop-by-hop fragmentation. Besides that, we also profiled the forwarder to show that time-related functions are the performance bottlenecks. This finding indicates the need to revisit the NDN forwarder design for performance improvement.

The differences between IP and NDN motivated our second work. The work is to detect the active caching decision mechanisms in NDN networks. In Chapter 6, we presented the first work to detect the caching decision from end hosts. We proposed a new method to extract the fingerprint for caching decision mechanisms, such as Cache Everything Everywhere (CEE), Leave the Copy Down (LCD) and others. The method uses two metrics for fingerprint generation. The first metric is Data packets' hop count distribution in measurement, and the second metric is the changes in the distribution across multiple measurements. To verify the robustness of the proposed method, we simulated the measurement on a small-scale topology to demonstrate that our method can detect caching decisions in the presence of cross traffic. The results also showed that our method could estimate the pre-assigned probability value for static probabilistic caching decisions. Since the NDN stack does not expose hop count information to applications, we proposed to use clustering algorithms to map delays to hop counts. We then evaluated the proposed method on a real topology. Simulations demonstrated that we could approximately identify the active caching decisions.

To conclude, in this dissertation, we have shown we can capture a substantial amount of useful and actionable measurements of NDN networks from end hosts. Any user can use the proposed methods to measure NDN networks from an end host, without the need for a special access privilege or the help from a third party. Our methods can integrate with any NDN measurement framework to provide insight into the target network, to help manage NDN networks, and to address network issues. Moreover, we minimize the overhead by sending a small number of probe packets during the measurement.

Bibliography

[1] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[2] Forwarding strategies. Available at https://ndnsim.net/2.1/fw.html.

[3] Chengyu Fan, Susmit Shannigrahi, Steve DiBenedetto, Catherine Olschanowsky, Christos Papadopoulos, and Harvey Newman. Managing scientific data with named data networking. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, page 1. ACM, 2015.

[4] Wentao Shang, Qiuhan Ding, Alessandro Marianantoni, Jeff Burke, and Lixia Zhang. Securing building management systems using named data networking. *IEEE Network*, 28(3):50–56, 2014.

[5] Lucas Wang, Ryuji Wakikawa, Romain Kuntz, Rama Vuyyuru, and Lixia Zhang. Data naming in vehicle-to-vehicle communications. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 328–333. IEEE, 2012.

[6] Wentao Shang, Alex Afanasyev, and Lixia Zhang. The design and implementation of the ndn protocol stack for riot-os. In *Globecom Workshops (GC Wkshps), 2016 IEEE*, pages 1–6. IEEE, 2016.

[7] NDN Testbed. Available at https://named-data.net/ndn-testbed/.

[8] ICN2020 Deliverables. Available at http://www.icn2020.org/.

[9] Piotr Zuraniewski, Niels van Adrichem, Daan Ravesteijn, Wieger IJntema, Christos Papadopoulos, and Chengyu Fan. Facilitating icn deployment with an extended openflow protocol. In *The 4th ACM Conference on Information-Centric Networking–ICN'17*, 2017.

[10] Walter Willinger. Measurements. Available at http://conferences.sigcomm.org/sigcomm/2018/files/tp/sigcomm18-tp-10-measurements.pdf.

[11] Ramon Caceres, Nick Duffield, Anja Feldmann, John D Friedmann, Albert Greenberg, Rick Greer, Theodore Johnson, Charles R Kalmanek, Balachander Krishnamurthy, Dianne Lavelle, et al. Measurement and analysis of ip network usage and behavior. *IEEE Communications Magazine*, 38(5):144–151, 2000.

[12] Stefan Savage et al. Sting: A tcp-based network measurement tool. In *USENIX Symposium on Internet Technologies and Systems*, volume 2, pages 7–7, 1999.

[13] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of http video streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 485–492. IEEE, 2011.

[14] Sebastian Egger, Tobias Hossfeld, Raimund Schatz, and Markus Fiedler. Waiting times in quality of experience for web based services. In *2012 Fourth International Workshop on Quality of Multimedia Experience*, pages 86–96. IEEE, 2012.

[15] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. Measurement of quality of experience of video-on-demand services: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):401–418, 2015.

[16] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer, 2002.

[17] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 2, pages 905–914. IEEE, 2001.

[18] ndn-tools. Available at https://github.com/named-data/ndn-tools.

[19] Siham Khoussi, Davide Pesavento, Lotfi Benmohamed, and Abdella Battou. Ndn-trace: a path tracing utility for named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 116–122. ACM, 2017.

[20] X. Shao H. Asaeda. Ccninfo: Discovering content and network information in content-centric networks. Technical report, 2018.

[21] Xavier Marchal, Thibault Cholez, and Olivier Festor. Server-side performance evaluation of ndn. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 148–153. ACM, 2016.

[22] Viewing network performance through the eyes of end hosts. Available at https://blog.apnic.net/2018/05/17/viewing-network-performance-through-the-eyes-of-end-hosts/.

[23] Siham Khoussi, Ayoub Nouri, Junxiao Shi, James Filliben, Lotfi Benmohamed, Abdella Battou, and Saddek Bensalem. Performance evaluation of a ndn forwarder using statistical model checking. *arXiv preprint arXiv:1905.01607*, 2019.

[24] Data plane development kit. Available at https://www.dpdk.org/.

[25] 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*, pages 1–12. ACM, 2009.

[26] A. Rahman, D. Trossen, D. Kutscher, R. Ravindran. Deployment Considerations for Information-Centric Networking (ICN). Available at https://tools.ietf.org/html/draft-irtf-icnrg-deployment-guidelines-03.

[27] M Zubair Shafiq, Amir R Khakpour, and Alex X Liu. Characterizing caching workload of a large commercial content delivery network. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[28] Jiaming Zhao, Pingjia Liang, Weiming Liufu, and Zhengping Fan. Recent developments in content delivery network: A survey. In *International Symposium on Parallel Architectures, Algorithms and Programming*, pages 98–106. Springer, 2019.

[29] Adam Wierzbicki, Nathaniel Leibowitz, Matei Ripeanu, and Rafał Woźniak. Cache replacement policies for p2p file sharing protocols. *European Transactions on Telecommunications*, 15(6):559–569, 2004.

[30] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 177–190. ACM, 2002.

[31] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. *IETF*, 3:1–67, 2011.

[32] Tatu Ylonen and Chris Lonvick. Rfc 4253: The secure shell (ssh) transport layer protocol. *The Internet Society*, 2006.

[33] NDN Essential Tools. Available at https://github.com/named-data/ndn-tools.

[34] ndn-atmos, https://github.com/named-data/ndn-atmos.

[35] repo-ng: Next generation of NDN repository. Available at https://github.com/named-data/repo-ng.

[36] Peter Gusev and Jeff Burke. Ndn-rtc: Real-time videoconferencing over named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, pages 117–126. ACM, 2015.

[37] NFD - Named Data Networking Forwarding Daemon. Available at https://github.com/named-data/NFD.git.

[38] National science foundation (NSF) future of internet architecture (FIA) program. Available at http://www.nets-fia.net/.

[39] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. Schematizing trust in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, pages 177–186. ACM, 2015.

[40] Jon Postel. Rfc 791. *The Internet Protocol*, 1981.

[41] Steve Deering and Robert Hinden. Rfc2460: Internet protocol, version 6 (ipv6) specification, 1998.

[42] NDN Memo. Packet fragmentation in ndn: Why ndn uses hop-by-hop fragmentation.

[43] Shehnaaz Yusuf. Survey of publish subscribe communication system. *Department of Computer Science, Kent State University*, 2004.

[44] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content based routing with elvin4. In *Proceedings AUUG2k*, 2000.

[45] Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gün Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *NSDI*, volume 6, pages 2–2, 2006.

[46] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 14–25, 2007.

[47] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *Proceedings*

of the 2007 inaugural international conference on Distributed event-based systems, pages 2–13, 2007.

[48] Clint Heyer. The å publish/subscribe framework. In *International Conference on Ubiquitous Intelligence and Computing*, pages 99–110. Springer, 2009.

[49] Vasilis Sourlas, Georgios S Paschos, Paris Flegkas, and Leandros Tassiulas. Caching in content-based publish/subscribe systems. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2009.

[50] Vasilis Sourlas, Georgios S Paschos, Petteri Mannersalo, Paris Flegkas, and Leandros Tassiulas. Modeling the dynamics of caching in content-based publish/subscribe systems. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 478–485, 2011.

[51] Luca Muscariello, Giovanna Carofiglio, and Massimo Gallo. Bandwidth and storage sharing performance in information centric networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 26–31, 2011.

[52] Giovanna Carofiglio, Vinicius Gehlen, and Diego Perino. Experimental evaluation of memory management in content-centric networking. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2011.

[53] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60, 2012.

[54] Ioannis Psaras, Richard G Clegg, Raul Landa, Wei Koong Chai, and George Pavlou. Modelling and evaluation of ccn-caching trees. In *International Conference on Research in Networking*, pages 78–91. Springer, 2011.

[55] Tiankui Zhang, Xiaogeng Xu, Le Zhou, Xinwei Jiang, and Jonathan Loo. Cache space efficient caching scheme for content-centric mobile ad hoc networks. *IEEE Systems Journal*, 13(1):530–541, 2018.

[56] Xiaoyan Hu, Jian Gong, Guang Cheng, and Chengyu Fan. Enhancing in-network caching by coupling cache placement, replacement and location. In *2015 IEEE International Conference on Communications (ICC)*, pages 5672–5678. IEEE, 2015.

[57] Jakob Pfender, Alvin Valera, and Winston KG Seah. Content delivery latency of caching strategies for information-centric iot. *arXiv preprint arXiv:1905.01011*, 2019.

[58] Jakob Pfender, Alvin Valera, and Winston KG Seah. Performance comparison of caching strategies for information-centric iot. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pages 43–53, 2018.

[59] Yanyong Zhang, Dipankar Raychadhuri, Luigi Alfredo Grieco, Emmanuel Baccelli, Jeff Burke, Ravishankar Ravindran, Guoqiang Wang, Anders Lindgren, Bengt Ahlgren, and Olov Schelén. Requirements and challenges for iot over icn. 2015.

[60] Mohamed Ahmed Hail, Marica Amadeo, Antonella Molinaro, and Stefan Fischer. Caching in named data networking for the wireless internet of things. In *2015 international conference on recent advances in internet of things (RIoT)*, pages 1–6. IEEE, 2015.

[61] Saran Tarnoi, Kalika Suksomboon, Wuttipong Kumwilaisak, and Yusheng Ji. Performance of probabilistic caching and cache replacement policies for content-centric networks. In *39th Annual IEEE Conference on Local Computer Networks*, pages 99–106. IEEE, 2014.

[62] Zhe Li and Gwendal Simon. Time-shifted tv in content centric networks: The case for cooperative in-network caching. In *2011 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2011.

[63] Yinlong Liu, Dali Zhu, and Wei Ma. A novel cooperative caching scheme for content centric mobile ad hoc networks. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 824–829. IEEE, 2016.

[64] Serdar Vural, Ning Wang, Pirabakaran Navaratnam, and Rahim Tafazolli. Caching transient data in internet content routers. *IEEE/ACM Transactions on Networking*, 25(2):1048–1061, 2016.

[65] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. Cache "less for more" in information-centric networks (extended version). *Computer Communications*, 36(7):758–770, 2013.

[66] Hitoshi Asaeda, Kazuhisa Matsuzono, and Thierry Turletti. Contrace: a tool for measuring and tracing content-centric networks. *IEEE Communications Magazine*, 53(3):182–188, 2015.

[67] Hitoshi Asaeda and X Shao. Ccninfo: Discovering content and network information in content-centric networks. Technical report, IRTF Internet Draft (work in progress), 2018.

[68] Cc* integration: Sandie: Sdn-assisted ndn for data intensive experiments. Available at http://grantome.com/grant/NSF/ACI-1659403.

[69] Claude Chaudet, Eric Fleury, Isabelle Guérin Lassous, Hervé Rivano, and Marie-Emilie Voge. Optimal positioning of active and passive monitoring devices. In *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 71–82. ACM, 2005.

[70] Mike Freedman. Network measurement. Available at http://www.cs.princeton.edu/courses/archive/spr14/cos461/docs/rec10-measurement.pdf.

[71] Nevil Brownlee, Chris Loosley, et al. Fundamentals of internet measurement: A tutorial _. 2001.

[72] Venkat Mohan, YR Janardhan Reddy, and K Kalpana. Active and passive network measurements: a survey. *International Journal of Computer Science and Information Technologies*, 2(4):1372–1385, 2011.

[73] Richard A Steenbergen. A practical guide to (correctly) troubleshooting with traceroute. *North American Network Operators Group*, pages 1–49, 2009.

[74] Ratul Mahajan, Ming Zhang, Lindsey Poole, and Vivek S Pai. Uncovering performance differences among backbone isps with netdiff. In *NSDI*, pages 205–218, 2008.

[75] Ethan Katz-Bassett, Harsha V Madhyastha, John P John, Arvind Krishnamurthy, David Wetherall, and Thomas E Anderson. Studying black holes in the internet with hubble. In *NSDI*, volume 8, pages 247–262, 2008.

[76] Guy Almes, Sunil Kalidindi, and Matthew Zekauskas. A round-trip delay metric for ippm. Technical report, 1999.

[77] Ravi Prasad, Constantinos Dovrolis, Margaret Murray, and KC Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE network*, 17(6):27–35, 2003.

[78] Stanislav Shalunov, Benjamin Teitelbaum, Anatoly Karp, Jeff Boote, and Matthew Zekauskas. A one-way active measurement protocol (owamp). Technical report, 2006.

[79] Van Jacobson. Pathchar: A tool to infer characteristics of internet paths, 1997.

[80] Benoit Claise. Cisco systems netflow services export version 9. Technical report, 2004.

[81] Jeffrey D Case, Mark Fedor, Martin L Schoffstall, and James Davin. Simple network management protocol (snmp). Technical report, 1990.

[82] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions On Networking*, 12(6):963–977, 2004.

[83] Attila Pásztor and Darryl Veitch. The packet size dependence of packet pair like methods. In *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No. 02EX564)*, pages 204–213. IEEE, 2002.

[84] Ravi Prasad, Manish Jain, and Constantinos Dovrolis. Effects of interrupt coalescence on network measurements. In *International Workshop on Passive and Active Network Measurement*, pages 247–256. Springer, 2004.

[85] pathrate. Available at https://www.cc.gatech.edu/~dovrolis/bw-est/pathrate.html.

[86] sting. Available at http://cseweb.ucsd.edu/~savage/sting/.

[87] fping. Available at https://fping.org/.

[88] echoping. Available at https://framagit.org/bortzmeyer/echoping.

[89] Owamp. Available at http://software.internet2.edu/owamp/.

[90] ndnping-dpdk. Available at https://github.com/usnistgov/ndn-dpdk/tree/master/cmd/ndnping-dpdk.

[91] iperf3. Available at https://iperf.fr.

[92] netperf. Available at https://hewlettpackard.github.io/netperf/.

[93] bing. Available at http://fgouget.free.fr/bing/index-en.shtml.

[94] clink. Available at http://allendowney.com/research/clink/.

[95] pchar. Available at http://www.kitchenlab.org/www/bmah/Software/pchar/.

[96] sprobe. Available at http://sprobe.cs.washington.edu/.

[97] ndn-traffic-generator: Traffic generator for ndn. Available at https://github.com/named-data/ndn-traffic-generator.

[98] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[99] Mtr. Available at http://www.bitwizard.nl/mtr/.

[100] Bgpmon. Available at https://www.bgpmon.io/.

[101] Tcpdump & lib-cap. Available at http://www.tcpdump.org/.

[102] Iptraf. Available at http://iptraf.seul.org/.

[103] tcpdpri. Available at http://fly.isti.cnr.it/software/tcpdpriv/.

[104] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[105] Argus. Available at http://argus.tcp4me.com/.

[106] Cisco ios netflow. Available at https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[107] Dummynet. Available at https://cs.baylor.edu/~donahoo/tools/dummy/.

[108] ns-3. Available at https://www.nsnam.org/.

[109] Opnet modeler. Available at http://opnetprojects.com/opnet-modeler/.

[110] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim 2.0: A new version of the ndn simulator for ns-3. *NDN, Technical Report NDN-0028*, 2015.

[111] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Icarus: a caching simulator for information centric networking (icn). In *SimuTools*, volume 7, pages 66–75. ICST, 2014.

[112] Andreas Hanemann, Jeff W Boote, Eric L Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Łapacz, D Martin Swany, Szymon Trocha, and Jason Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *International conference on service-oriented computing*, pages 241–254. Springer, 2005.

[113] Ripe atlas. Available at https://atlas.ripe.net/.

[114] Same knows. Available at https://samknows.com/.

[115] Davide Pesavento, Omar Ilias El Mimouni, Eric Newberry, Lotfi Benmohamed, and Abdella Battou. A network measurement framework for named data networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 200–201, 2017.

[116] AF_PACKET Poll Mode Driver. Available at https://doc.dpdk.org/guides/nics/af_packet.html.

[117] Alexander Afanasyev, Junxiao Shi, Lan Wang, Beichuan Zhang, and Lixia Zhang. Packet fragmentation in ndn: why ndn uses hop-by-hop fragmentation. *NDN Technical Report NDN-0032*, 2015.

[118] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *USITS*, volume 1, pages 11–11, 2001.

[119] Interrupt coalescence at the nic layer. Available at https://www.ibm.com/support/knowledgecenter/en/SSQPD3_2.6.0/com.ibm.wllm.doc/batchingnic.html.

[120] Intel Developer Zone. Intel vtune amplifier, 2017. *Documentation at the URL: https://software. intel. com/en-us/intel-vtune-amplifier-xe-support/documentation*.

[121] Userspace rcu. Available at http://liburcu.org/.

[122] Michael Rabinovich and Oliver Spatscheck. *Web caching and replication*, volume 67. Addison-Wesley Boston, USA, 2002.

[123] Sumanta Saha, Andrey Lukyanenko, and Antti Ylä-Jääski. Cooperative caching through routing control in information-centric networks. In *2013 Proceedings IEEE INFOCOM*, pages 100–104. IEEE, 2013.

[124] Dario Rossi and Giuseppe Rossini. On sizing ccn content stores by exploiting topological information. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 280–285. IEEE, 2012.

[125] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16):3128–3141, 2013.

[126] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[127] Rocketfuel: An isp topology mapping engine. Available at https://research.cs.washington.edu/networking/rocketfuel/.

[128] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[129] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[130] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[131] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998.

[132] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM Sigmod Record*, 25(2):103–114, 1996.

[133] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[134] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[135] Cenk Gündoğran, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C Schmidt, and Matthias Wählisch. Ndn, coap, and mqtt: a comparative measurement study in the

iot. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pages 159–171, 2018.

[136] Haowei Yuan and Patrick Crowley. Performance measurement of name-centric content distribution methods. In *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pages 223–224. IEEE, 2011.

[137] Huichen Dai, Yi Wang, Jindou Fan, and Bin Liu. Mitigate ddos attacks in ndn by interest traceback. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 381–386. IEEE, 2013.

[138] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. Dos and ddos in named data networking. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7. IEEE, 2013.

[139] Alberto Compagno, Mauro Conti, Paolo Gasti, and Gene Tsudik. Poseidon: Mitigating interest flooding ddos attacks in named data networking. In *38th annual IEEE conference on local computer networks*, pages 630–638. IEEE, 2013.