UNIVERSITY OF CALIFORNIA
Los Angeles

# Peer-to-Peer Data Sharing in Named Data Networking

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

by

Spyridon Mastorakis

2019

ABSTRACT OF THE DISSERTATION

Peer-to-Peer Data Sharing
in Named Data Networking

by

Spyridon Mastorakis
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2019
Professor Lixia Zhang, Chair

Peer-to-peer file sharing applications envision a world, where peers will communicate in terms of the data that they are looking for. In this world, peers will be able to retrieve the desired data from *any other* peer that can provide it, without the need of specifying the location that this data can be found. Some peer-to-peer applications, such as BitTorrent, also provide data-centric security primitives by verifying the integrity of the downloaded data through cryptographic hashes. However, the current point-to-point TCP/IP network architecture poses a number of challenges to the design and implementation of peer-to-peer systems both in infrastructure-based and mobile ad-hoc networks. Specifically, in infrastructure-based networks, peers have to select others (identified by an IP address) to download data from, estimate the quality of each connection, and constantly try to find peers that can provide higher bandwidth. In mobile ad-hoc networks, peers typically rely on a routing protocol to establish an end-to-end path to the IP addresses of other peers. The established paths may constantly break, since the position of mobile peers changes, therefore, new paths have to be frequently established.

In this dissertation, we present peer-to-peer file sharing application designs and implementations that run on top of the Named Data Networking (NDN) architecture. NDN provides a data-centric communication model directly at the network layer, as well as data-centric security primitives, making security a property of the data that stays with it at rest

and in transit across the network. Our experimental results show that NDN's named and secured data, adaptive forwarding, and caching provide a solid foundation for peer-to-peer applications both in infrastructure-based and mobile ad-hoc networks, achieving lower data dissemination delays and overheads compared to TCP/IP-based solutions.

At the end of this dissertation, we present the design of the most popular NDN software platform, ndnSIM, the NDN simulator based on the ns-3 network simulator. As a network architecture that fundamentally departs from TCP/IP, NDN requires extensive evaluation and experimentation. This need along with the modular and scalable design of ndnSIM, featuring integration with the real-world NDN prototypes, have contributed to the massive ndnSIM adoption by the broader research community.

The dissertation of Spyridon Mastorakis is approved.

Beichuan Zhang

Jingsheng Jason Cong

Songwu Lu

Lixia Zhang, Committee Chair

University of California, Los Angeles

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Lixia Zhang, for her support throughout my PhD studies, as well as my PhD committee members, Prof. Songwu Lu, Prof. Jason Cong, and Prof. Beichuan Zhang. I would like to thank my collaborators from the Internet Research Laboratory at UCLA, as well as my family and friends for their support.

Material from Chapters 2, 3, and 6 has been published in the paper S. Mastorakis, A. Afanasyev, Y. Yu, and L. Zhang, "nTorrent: Peer-to-Peer File Sharing in Named Data Networking", in Proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN), July 2017. L. Zhang proposed the original idea and helped with the design of nTorrent, as well as the publication submission. A. Afanasyev and Y. Yu helped with the nTorrent design and the publication submission.

Material from Chapters 5 and 6 has been published in the paper S. Mastorakis, A. Afanasyev, and L. Zhang, "On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation," ACM SIGCOMM Computer Communication Review (CCR), July 2017. A. Afanasyev has originally developed ndnSIM in 2012. Since 2014, A. Afanasyev and I have been co-leading the ndnSIM design and development. L. Zhang has been overseeing the process. A. Afanasyev and L. Zhang also provided feedback on the publication manuscript and helped during revisions of manuscript drafts.

# VITA

| | |
|---|---|
| 2013–2014 | Undergraduate Research Assistant, Network Management & Optimal Design Laboratory, National Technical University of Athens, Athens, Greece. |
| 2014 | B.S. (Electrical and Computer Engineering), National Technical University of Athens, Athens, Greece. |
| 2014-2019 | Graduate Student Researcher, UCLA, Los Angeles, California. |
| 2015-2019 | Teaching Assistant, UCLA, Los Angeles, California. |
| 2016 | Software Engineering Intern, Cisco Systems, Cambridge, Massachusetts. |
| 2017 | M.S. (Computer Science), UCLA, Los Angeles, California.. |
| 2017 | Software Engineering Intern, Intel Corporation, Santa Clara, California. |

# PUBLICATIONS

Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, L. Zhang, "An Overview of Security Support in Named Data Networking," IEEE Communications Magazine, November 2018.

Y. Zhang, Z. Xia, S. Mastorakis, L. Zhang, "KITE: Producer Mobility Support in Named Data Networking," in Proceedings of the 5th ACM Conference on Information-Centric Networking, September 2018.

S. Mastorakis, P. Gusev, A. Afanasyev, and L. Zhang, "Real-Time Data Retrieval in Named

Data Networking," in Proceedings of the 1st IEEE International Conference on Hot Topics in Information-Centric Networking (IEEE HotICN), August 2018.

Z. Zhang, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, "Security Support in Named Data Networking," Technical Report NDN-0057, March 2018.

S. Mastorakis, T. Ahmed, and J. Pisharath, "ISA-Based Trusted Network Functions and Server Applications in the Untrusted Cloud," arXiv, February 2018.

S. Mastorakis, K. Chan, B. Ko, A. Afanasyev, and L. Zhang, "Experimentation with Fuzzy Interest Forwarding in Named Data Networking," arXiv, January 2018.

K. Chan, B. Ko, S. Mastorakis, A. Afanasyev, and L. Zhang, "Fuzzy Interest Forwarding," in Proceedings of the 13th Asian Internet Engineering Conference (AINTEC), November 2017.

S. Mastorakis, A. Afanasyev, and L. Zhang, "On the Evolution of ndnSIM: An Open-Source Simulator for NDN Experimentation," ACM SIGCOMM Computer Communication Review (CCR), July 2017.

S. Mastorakis, A. Afanasyev, Y. Yu, and L. Zhang, "nTorrent: Peer-to-Peer File Sharing in Named Data Networking," in Proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN), July 2017.

S. Mastorakis, J. Gibson, I. Moiseenko, R. Droms, and D. Oran "ICN Traceroute Protocol Specification," IRTF ICN Research Group (ICNRG), September 2016.

S. Mastorakis, J. Gibson, I. Moiseenko, R. Droms, and D. Oran "ICN Ping Protocol Specification," IRTF ICN Research Group (ICNRG), September 2016.

# CHAPTER 1

# Introduction

Peer-to-peer file sharing applications envision a data-centric world for the dissemination of data, where applications exclusively focus on securely sharing the desired data with low delays and minimal overhead. Existing peer-to-peer applications implement this data-centric view at the application layer; such applications run on top of the point-to-point TCP/IP network architecture, which introduces a number of challenges to their design and implementation.

Peer-to-peer applications in infrastructure-based networks, such as BitTorrent [Coh03], have to maintain an overlay of peers. They first need to discover the IP address of peers in the swarm and then establish TCP connections to all or a subset of these peers. In addition to that, they have to constantly measure the performance of each connection and select the "best" peers. Given that such applications have no knowledge of the network topology or the routing policies, they can only estimate who the "best" peers are based on the download bandwidth that others can provide.

In Mobile Ad-hoc NETworking (MANET) communication scenarios, peers are mobile with intermittent connectivity to each other and the network topology dynamic. Existing peer-to-peer applications typically rely on a MANET routing protocol to establish an end-to-end path to others. After a path is established, the data dissemination process can begin, while security in the routing and data dissemination process is not considered [YLY04]. Moreover, IP address configuration becomes a challenge; a number of existing solutions have been proposed [NP02, Mis01], which share the goal of assigning to each entity an IP address that does not collide with others. That is, in the context of MANET communication, IP addresses are merely unique node identifiers, since the node location may change constantly.

As a result, the established paths to peers may break, therefore, the routing protocol has to frequently re-establish new paths to peers.

Named Data Networking (NDN) [Zha14] is a proposed Internet architecture that features a receiver-driven data-centric communication model. NDN makes named and secured data packets the centerpiece of the network architecture. This enables applications to focus exclusively on what data to fetch, leaving to the network to decide exactly from where to fetch the data.

This dissertation focuses on investigating the impact and trade-offs of providing the data-centric logic at the network layer instead of the application layer for peer-to-peer applications both for infrastructure-based and MANET communication scenarios. For infrastructure-based networks, we propose nTorrent [MAY17b], a peer-to-peer file sharing system that fully utilizes the inherent features of NDN; stateful and adaptive forwarding, data-centric security and integrity on per network layer packet basis, and data caching and retention provided by the network. We evaluate nTorrent under a wide range of different scenarios and we demonstrate that, by leveraging these inherent NDN features, it can effectively deal with with heavy peer churn and resist again flash crowd scenarios. As a result, nTorrent can achieve 30% lower data downloading delays, while generating more than 50% less network traffic compared to BitTorrent when multiple peers share common files at the same time.

For MANET communication scenarios, we propose DAPES, a peer-to-peer file sharing solution built on top of the NDN abstraction for named and secured data. DAPES provides a set of abstractions to name collections of files and mechanisms to discover neighboring peers and what data they can offer, advertise what data each peer has in a compressed fashion, facilitate the dissemination of rare data, as well as mitigate collisions in case of simultaneous transmissions. DAPES can also achieve communication among peers over multiple wireless hops without the need of "traditional" MANET routing protocol. As a result, DAPES achieves 2x-3.5x lower overheads and 25% shorter data download times compared to solutions that rely on IP-based mobile ad-hoc routing.

A large portion of the experiments for nTorrent and DAPES were conducted on ndnSIM [MAM15,

MAM16, MAZ17] , the de-facto NDN simulator based on ns-3. ndnSIM was originally designed and developed in 2012 to provide an NDN evaluation and experimentation platform common to the broader research community. In 2014, ndnSIM was heavily re-designed and re-factored to offer integration with the real-world NDN prototype software (the NFD software forwarder [AS15] and the ndn-cxx software library). Seven years after its original release, ndnSIM is the most popular NDN software platforms, featuring a mailing list of more than 700 users. ndnSIM has been used by thousands of students and researchers all over the world and has been cited hundreds of times in published research work over the last few years.

The rest of this dissertation is organized as follows. In Chapter 2, we present some brief background on NDN and BitTorrent. We also describe the challenges that BitTorrent has faced by running at the application layer of the point-to-point TCP/IP architecture, as well as investigate the similarities and differences between NDN and BitTorrent. In Chapter 3, we present the nTorrent design and our experimental evaluation. In Chapter 4, we present the design of DAPES and the related evaluation experiments and results. In Chapter 5, we present the design of ndnSIM, we discuss the integration process with the NDN prototypes and the challenges we faced, we describe the ndnSIM software development model, and present data about its adoption by the community. In Chapter 6, we present related work on peer-to-peer file sharing in wired infrastructure networks and MANET scenarios, as well as some existing network evaluation frameworks, including testbed deployments, emulation, and simulation frameworks. Finally, Chapter 7 concludes this dissertation and discusses future work.

# CHAPTER 2

# Background

## 2.1 NDN Overview

The NDN communication model is consumer-driven. Data consumers express Interest packets (requests for data) containing a name that uniquely identify the desired data (Figure 2.2). An NDN router (Figure 2.1) forwards requests towards data producer(s) (upstream direction) based on the requests' name and information on its name-based Forwarding Information Table (FIB). When an Interest reaches a node (router or end host) that has the requested data, this node returns the data packet to "satisfy" the Interest.

NDN names are hierarchical, for example "`/ucla.edu/cs/spyros/papers/ntorrent.pdf /segment1`". Therefore, an Interest can be satisfied by a data packet based on name prefix matching; the data packet can have a longer name than the corresponding Interest.

Every router along the Interest forwarding path keeps the state for each unsatisfied Interest in a Pending Interest Table (PIT), where simultaneous Interests for the same data are aggregated, so that a single request is forwarded upstream. A data packet uses the state set up by its corresponding Interest at each-hop router to follow the reverse way (downstream direction) back to all the requesting consumers (inherent support for data multicast). The corresponding pending Interest entry in each router's PIT is satisfied and a closed feedback loop is created that enables routers to measure data plane performance. A router can also cache data packets in its Content Store (CS) to satisfy future requests for the same data.

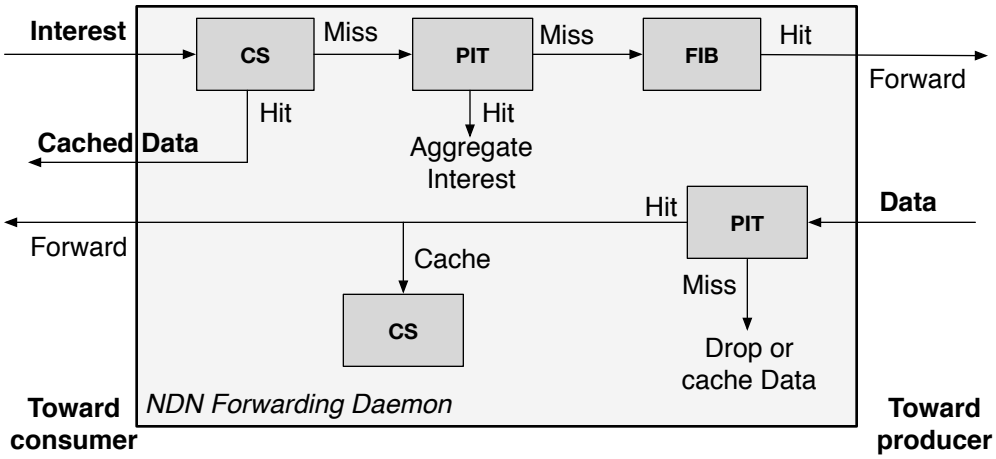



Figure 2.1: Packet processing and forwarding by an NDN router

### 2.1.1 NDN Security & Data Integrity Verification

NDN secures data directly at the network layer, so that applications do not have to care where the data comes from. Each NDN data packet has a digital signature generated by its producer (Figure 2.2). This signature binds the data to its name, so that a data consumer can authenticate the data directly using the producer's public key, rather than relying on a secure channel.

To enable consumer applications to verify data integrity without the need of checking the signature of each individual data packet, consumers can retrieve data packets using a *full name*; a concatenation of the data name and the hash of the entire data packet (called implicit digest [Net11, NDN14]).

### 2.1.2 NDN Routing and Forwarding

The NDN routing protocols (e.g., NLSR [HAA13]) help routers to build their FIB. With the routing protocol, data producers can announce the name prefixes of their data to the network, while routers can propagate the reachability of name prefixes across the network.

The decision of whether, when and through which face(s), an Interest will be forwarded is made by a network-layer module running at each router, called *forwarding strategy*, that accepts input from the FIB. A forwarding strategy achieves *adaptive forwarding*, *traffic*

*localization* and *maximization of the data retrieval speed* through the following mechanisms [YAM13]:

- **Prioritizes next-hops toward local (within the same network or AS) over remote data.** To achieve that, it leverages the local routing policies.
- **Discovers the best data location(s) (producer applications or in-network caches) to fetch data from in terms of data plane performance.** To determine the performance of each next-hop, the strategy performs a next-hop (path) exploration toward the potential locations at the beginning of data retrieval; it tries all the available faces of a FIB entry (multi-path Interest forwarding) and measures their performance in terms of Round Trip Time (RTT). Throughout the fetching process, the strategy sends probe Interests through faces that either have not been explored yet or could not retrieve data in the past to dynamically discover potentially better locations.
- **Resolves data retrieval errors locally by trying alternative next-hops or occasionally repeating the path exploration phase.** During data retrieval, a router might not be able further forward an Interest (there is no entry for this prefix in its FIB) or fetch data for it (an Interest reached a location that does not have the requested data). In this case, the router returns a Negative ACKnowledgement [YAM13] back to its downstream router.
- **Fully utilizes the next-hop toward the location with the best data plane performance first, and then, expands Interest forwarding to subsequent next-hops, fetching data in parallel by multiple locations.** To achieve congestion control on a hop-by-hop basis, downstream routers can estimate the bandwidth utilization of each next-hop and upstream routers can send a NACK downstream to control the sending rate of a downstream router.

### 2.1.3 Scaling NDN Forwarding

Broadly speaking, Interests carry "what" semantics (i.e., the name of the data they are looking for). The network helps the Interests bring data back by being aware of "where" semantics (i.e., where the requested data is located). The network becomes aware of the "where" semantics through routing announcements. However, since there is an unbounded number of application names, there may be concerns on how to keep the FIB size under control across the global Internet when routing announcements take place directly on application names. To maintain the scalability of NDN forwarding, Afanasyev et al. [AYW15] proposed the SNAMP scheme, which is briefly discussed below.

SNAMP [AYW15] proposes a mapping of application names to a set of globally routable name prefixes used for data retrieval across the Internet. A producer creates an association between a data prefix and a number of globally routable name prefixes. The association of an *application level data name* to a globally routable name is called *a forwarding hint*. A consumer application attaches one or more forwarding hints to the expressed Interests, which will act as guidance for the Interest forwarding across the Internet. Once the Interests reach a network, where their name is on the FIB table of routers, Interests will be forwarded directly based on their name.

The problem of scaling the forwarding/routing plane is not new. The same problem applies to IP-based networks as well and the SNAMP approach is based on the map and encap approach to scale IP routing [Dee96]. This approach has not been widely deployed, however, a few more protocols, like LISP [FLM13] and ILNP [AB12], are based on this approach. As we mentioned above, in NDN, an Interest can carry both the "what" and "where" semantics, which is directly visible to the network. The "where" semantics corresponds to the forwarding hint, which helps routers during the forwarding process. In IP, packets still carry both semantics just in different forms, where only the "where" semantics (IP addresses, which are analogous to a forwarding hint) is visible to the network. The "what" semantics (application data) is encapsulated into a TCP/UDP datagram, which is encapsulated into an IP packet, thus it is invisible to the network and can only be accessed by applications.

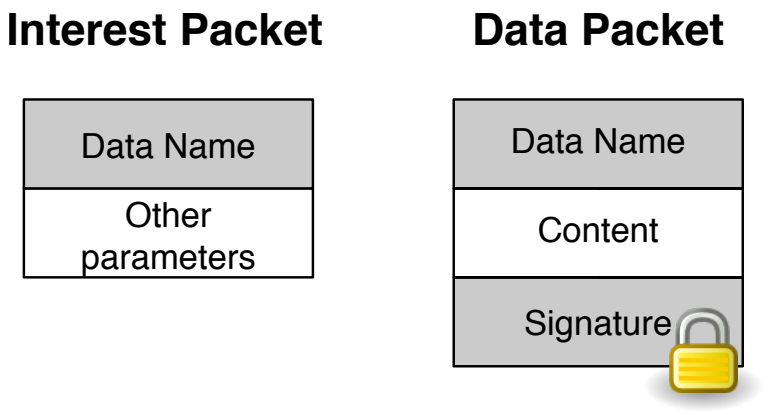



Figure 2.2: Interest & data packets

## 2.2 BitTorrent Design Challenges

In this section we briefly remind the reader with internals of the BitTorrent protocol and then discuss the challenges that BitTorrent faces due to implementing the data-centric logic on top of the point-to-point TCP/IP network.

### 2.2.1 BitTorrent Background

BitTorrent achieves fast peer-to-peer file downloading and distribution. A host running BitTorrent, called *peer*, participates in the downloading and sharing process with others of a collection of data, called *torrent*. Torrents are divided into equal sized data chunks, called *pieces*, each further split into packets [Tar10]. Peers that have complete torrent dataset are called *seeders* and peers with incomplete torrent are *leecher*. Leechers and seeders interested in downloading a common torrent form a sharing group, called *swarm*.

To start downloading a torrent, a leecher needs to discover some other seeders and/or leachers and join the swarm. In traditional BitTorrent, this is achieved by contacting a rendezvous point (*tracker*), which monitors IP addresses of seeders and leechers in the swarm. The information about the tracker is included in the *.torrent file*; obtained by to-be-leachers out-of-band (e.g., from a website, via email, etc.). Each piece of the torrent is directly secured through inclusion of its SHA1 hash in the .torrent file. This way, peers do not need to care from whom pieces are downloaded, as (accidentally or maliciously) corrupted data

will be simply ignored.

### 2.2.2 Peer Discovery

Because BiTorrent is built on top of TCP/IP, it requires a knowledge of IP addresses of peers in order to start downloading torrent pieces.

Traditionally, BitTorrent uses a tracker or a set of trackers to discover other peers in the swarm. More recent protocol extensions introduced tracker-less torrents, where peers can discover others in a decentralized way using a Distributed Hash Table (DHT) [Loe08] and a Peer Exchange (PEX) [WDH10] protocol. However, peers still need to bootstrap knowledge about peers in DHT either through the .torrent file that includes URLs of the "mainline" DHT nodes or using hardcoded information about the bootstrap nodes in the application. A peer first contacts a mainline DHT bootstrap node to get linked into the DHT, informs the system of its own existence and discovers more peers.

### 2.2.3 Peer Selection

After discovering others, a peer has to select the best subset of peers to fetch the torrent pieces from. Given that BitTorrent does not possess network topology or routing policy information, it can only guess which peers are the best based on download rate estimation. For that, it has to establish to pick a number of peers, establish TCP connection to each, and try to request torrent pieces. Given none of the initially selected peers can give the best performance, BitTorrent constantly picks news sets and re-evaluates from whom to download.

### 2.2.4 Rarest Piece Prioritization

Peers come and go in an unpredictable manner and if they go away before uploading their pieces to others, those pieces might become unavailable. Therefore, a peer has to decide which pieces to fetch first to benefit data replication and the overall downloading process for the entire swarm.

Since the TCP/IP network layer is not designed to support data retention, BitTorrent has to maximize the distribution of each piece at the application level, forcing peers to prioritize connections that serve rare pieces. To achieve function, BitTorrent uses the "Rarest Piece First" strategy [Coh], to ensure that each piece of the torrent is stored on as many available peers as possible.

### 2.2.5 Piece Integrity Verification

The TCP/IP network layer does not verify the integrity of individual packets, forcing BitTorrent implement this function at the application layer. In other words, when a peer downloads a piece, it verifies its integrity to ensure that the fetched data is not bogus using cryptographic hashes from the .torrent-file.

### 2.2.6 Data Sharing Incentives

To ensure that peers participate in data sharing, BitTorrent has to incentivize sharing using a game-theory based tit-fo-tat mechanism. Specifically, peers have to "choke" connections to "pure downloaders" (i.e., stop sending data to a leecher if it is not willing to upload data to others) [Coh], and "unchoke" only if the peer shares data. The choking state can be temporary, as optimistic unchoking [Coh] ensures that a leecher will be unchoked when it increases its upload rate.

### 2.2.7 Traffic Localization

The objective of BitTorrent peers is to minimize download time, while Internet Service Providers (ISPs) would like to minimize the volume of generated inter-AS traffic. However, BitTorrent has no knowledge of the underlying connectivity and make peer selection that goes against both peer and ISP goals. To address this problem, a lot of research has been conducted on improving the BitTorrent traffic locality [BCC06, LLD11, CB08]. These approaches either leverage some external knowledge of the network topology by the tracker or peers, or let peers perform path monitoring and latency measurements at the application

layer. Perkins [Per08] introduced a BitTorrent extension that uses DNS service discovery to enable peers within a local network discover each other and exchange data through this fast single-hop network. Some ISPs adopted a BitTorrent extension [HSH08] proposing the deployment of a "local" tracker within a domain to avoid inter-domain traffic.

The approaches mentioned above aim to make BitTorrent location-aware by using system components *external* to the protocol itself. The last approach also forces ISPs to update their DNS configuration to capture DNS queries about trackers from peers in their domain and return the address of the "local" tracker.

## 2.3 Similarities/Differences Between BitTorrent And NDN

Because of the common data-centric design model, NDN and BitTorrent share a number of common design elements. Given that they perform the data-centric communication model at different layers, they also have qualitatively different implementations of these elements, as summarized in Table 2.1.

Table 2.1: Comparison of common NDN and BitTorrent objectives

| | **NDN** | **BitTorrent** |
|---|---|---|
| **Data-centric security** | Cryptographic signature and full name per *data packet*, can be verified by both application and network | SHA1 hash *per piece*, can be verified by application only |
| **Efficient data retrieval** | At network layer, seamless to application through forwarding strategy | At application layer through peer selection |

### 2.3.1 Data-Centric Security

Both NDN and BitTorrent secure data directly. BitTorrent's security and integrity model is based on the .torrent file, while each piece can be verified only by peers. In NDN, each data packet carries a digital signature and can have a full name, so that it can be verified both by applications and routers.

### 2.3.2 Efficient Data Retrieval

NDN and BitTorrent share the goal of maximizing the data retrieval efficiency.

BitTorrent (without extensions that require infrastructure support) has no network-level knowledge, therefore it tries to maximize efficiency by increasing the downloading bandwidth; peers measure end-to-end download rates and choke and unchoke individual connections to find the best peers through trial-and-errors. The lack of network layer multicast also forces them to send the same data multiple times to simultaneous downloaders, reducing the useful network capacity.

Leveraging directly network layer information, NDN maximizes efficiency by retrieving the data that is "the most available" to consumers (prioritization of local copies with the best data plane performance, multi-path forwarding, parallel fetching). Peers can also retrieve recently fetched data from caches instead of other peers to reduce the retrieval delay and network load.

## 2.4 Communication Challenges of Peer-to-Peer File Sharing in TCP/IP-Based Mobile Ad-hoc Networks

In this section, we give an overview of the communication challenges for peer-to-peer file sharing in TCP/IP-Based Mobile Ad-hoc NETworks (MANETs).

***Establishing a connection between nodes through a MANET routing protocol:*** In IP-based MANETs the IP address has essentially lost its meaning, since it is merely used

for an identifier of the communicating nodes. To share data, a node needs to first establish a connection to another node (i.e., a forwarding path between a pair of IP addresses) and then start sharing data. Establishing a connection is typically achieved through an IP-based MANET routing protocol (e.g., DSDV [He02], AODV [PBD03]). However, nodes are mobile, therefore their position changes over time. To this end, the routing protocol needs to continuously re-establish the connection among a pair of nodes, which results in significant overhead for both reactive and proactive routing protocols.

***Node configuration:*** In IP-based MANETs, each node needs to be assigned an IP address. To achieve that, a number of solutions have been proposed over the years [Per00, CAG05, NP02, McA00, Mis01]. All the above solutions share the common goal of finding each node an IP address that does not collide with others. As mentioned above, in MANETs, IP addresses are merely a unique node identifier, since the node location may constantly change.

***Security:*** In such a dynamic environment, security is an especially important aspect, since nodes constantly move around, thus they are not aware of whom they encounter and share data with. As a result, it is trivial for malicious nodes to inject bogus packets in a MANET. In most of the MANET solutions proposed so far, security is not considered either during the routing or the data sharing process.

# CHAPTER 3

# nTorrent: Peer-to-Peer File Sharing in Wired Infrastructure Networks

## 3.1 NDN-Native Peer-to-Peer File Sharing

Based on our analysis in Chapters 2 and 2.3 and the challenges that BitTorrent faces in TCP/IP-based networks, we describe here what would be the benefits of designing and implementing a peer-to-peer file sharing application directly on top of a data-centric network architecture like NDN.

In NDN, peers do not need to explicitly discover other peers and/or manage individual connections to others (e.g., by measuring the stability and performance of these connections over time). As a result, an NDN-native BitTorrent application can get away from needing a tracker; peers simply express a request for data and the NDN network will make sure to find this data and return it back to them. In addition to that, an NDN-native BitTorrent can directly take advantage of network layer information to maximize the data downloading efficiency by retrieving data that is "the most available" to consumers (prioritization of local copies with the best data plane performance, multi-path forwarding, parallel fetching). Running over NDN helps a peer-to-peer file sharing application eliminate redundant fetching of the same data across the same path; NDN's native support for multicast data delivery and in-network caching reduce both network load and data retrieval delay. The stateful nature of the NDN network can offer adaptive forwarding of Interests to peers that are available at any given moment, as well as deal with peer disconnections and failures by forwarding Interests toward alternative directions/locations. In terms of security, BitTorrent provides mechanisms to verify the integrity of torrent pieces, which consist of multiple network-layer

packets, through a cryptographic hash for each piece. NDN can further enhance that by shifting the granularity of the integrity verification to individual network layer data packets; each NDN data packet is cryptographically signed by its producer, thus the packet can be authenticated once it is received, as well as peers can decide whether they trust the peer that generated the packet.

A peer in NDN acts as a consumer and a producer at the same time. As a consumer, it sends Interests to download data from others. As a producer, it uploads data to others (responds to received Interests from other peers), which requires the peer to announce the name prefixes of the data (i.e., the peer is willing to upload) to the routing system (as explained in section 2.1.2).

## 3.2 nTorrent Design

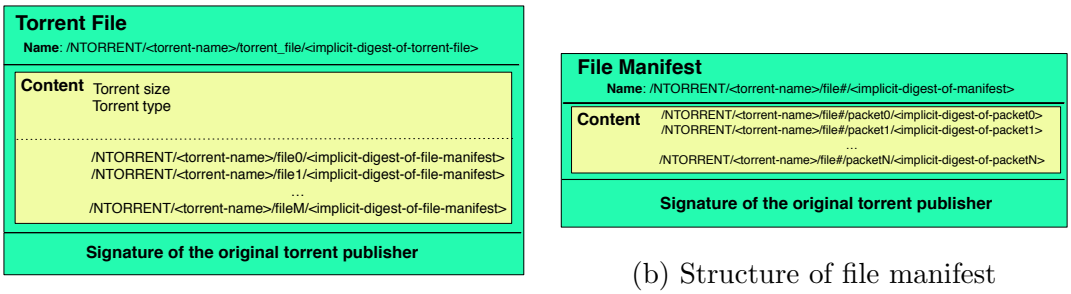
In this section, we present the nTorrent design [MAY17b, MAY17a]; we first discuss each individual design concept and then present a brief application scenario as an example of putting all the design concepts together.

### 3.2.1 Torrent File: the Truth of the File Set

To download a torrent (consisting of one or more files), a peer must first acquire the corresponding .torrent file; a piece of data uniquely identified by an NDN full name and signed by the original torrent producer, so that it can be securely retrieved and verified.

As illustrated in Figure 4.3, the name of a .torrent file consists of 4 components; the first one refers to the application, the second is the name of the torrent, the third refers to the .torrent file and the last one is the implicit digest of the .torrent file. Peers learn this name in a similar way as in BitTorrent.

With the .torrent file, peers discover the namespace structure (names of the file set), since it contains two pieces of information: 1) a description of the file set (e.g., torrent size and type), 2) the full names of a number of name catalogs (similar to [Moi14, BDN12]), called




(a) Structure of .torrent file

(b) Structure of file manifest

Figure 3.1: .Torrent File & File Manifest

*file manifests* (Figure 3.1b). Each of those catalogs contains the full names of the packets in a specific file in the torrent. The name of a file manifest consists of 4 components; the first refers to the application, the second is the name of the torrent, the third identifies a specific file in the torrent and the last one is the implicit digest of the file manifest.

In addition to having a full name, a manifest is signed by the original torrent publisher, therefore, it can be securely retrieved by peers. By downloading a manifest, a peer is able to securely retrieve the data packets of a file in the torrent.

The overall process of securely downloading the entire torrent is hierarchical and starts with the .torrent file, as illustrated in Figure 3.2; peers then download the file manifests and eventually the individual packets of each file in the torrent.

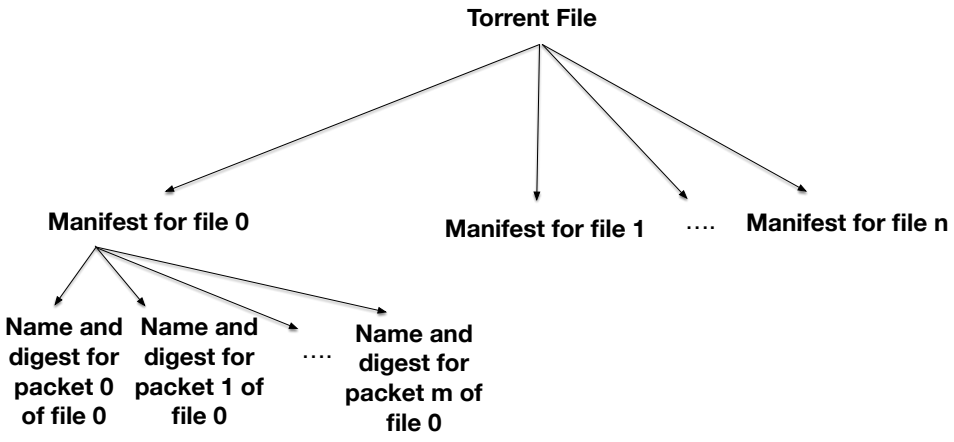



Figure 3.2: Hierarchical process of secure torrent retrieval

For large torrents, the original torrent publisher, based on the concept of Application

Layer Framing [CT90], segments each file in the torrent at proper boundaries. In such cases, .torrent files and file manifests may consist of one or more data packets (i.e., segments). If one data packet is not big enough to hold all the names of file manifests or packets in a file, it will carry some of them (let us assume the first k names), the second packet will carry the next k, and so on so forth. A peer interested in specific files contained in a torrent needs to fetch only the manifests of these files.

### 3.2.2 Namespace Design

Files in torrents can be fairly large (e.g., a few GBs each), therefore, they are split into multiple network layer data packets. Our namespace design needs to account for this fact, thus facilitate the easy identification of a selected torrent, a selected file in a torrent and each individual data packet. Moreover, our namespace design needs to provide a "cheap" way for users to verify the integrity of each data packet without the need to verify the signature of each single data packet.

To this end, we use a hierarchical naming scheme to reflect the relation among torrent name, set of files in a torrent, and individual data packets in a file as illustrated in Figure 3.3:

- The first component identifies the nTorrent application.
- The second refers to the name of the torrent.
- The next two components specify the offset of the desired file and packet respectively.
- The last component is the implicit digest of the packet.


Figure 3.3: nTorrent Interest format

### 3.2.3 Sequential Data Retrieval

Data packets are cached in NDN routers, making data replication inherent across the network and eliminating the need of a "Rarest Piece First" strategy.

nTorrent should fetch data in a way that maximizes the utilization of in-network caching. Intuitively, fetching data sequentially is likely to contribute to the utilization of caches during simultaneous downloads. A peer requests data, starting from the first packet in the first file in the torrent, one by one until the last packet in the last file. Therefore, a sequence of packets can be cached across the network and be retrieved by all the peers that download the torrent at the same time.

### 3.2.4 nTorrent Forwarding Strategy

The NDN network supports nTorrent through the deployment of a forwarding strategy at routers, which has the objectives mentioned in Section 2.1.2. Specifically, this strategy:

- Prioritizes next-hops toward local (within the same network or AS) over remote torrent data by following local routing policies.
- Discovers the best data location(s) (peers or in-network caches) to fetch data from in terms of data plane performance. The strategy tries all the available faces of a FIB entry (multi-path Interest forwarding) and measures their performance in terms of Round Trip Time (RTT). Throughout the fetching process, the strategy sends probe Interests through faces that either have not been explored yet or could not retrieve data in the past to dynamically discover potentially better locations (e.g., new peers or in-network caches).
- Resolves data retrieval errors (e.g., peer departures/disconnections) by trying alternative next-hops or occasionally repeating the path exploration phase. During torrent data retrieval, a router might not be able further forward an Interest (there is no entry for this prefix in its FIB) or fetch data for it (an Interest reach a peer that is disconnected or does not have the requested data). In this case, the router returns a Negative ACKnowledgement [YAM13] back to its downstream router.

- Fully utilizes the next-hop toward the location with the best data plane performance first, and then, expands Interest forwarding to subsequent next-hops, fetching data in parallel by multiple peers.

### 3.2.5 Routing Announcement Trade-Offs

To share data with others, a peer has to announce the data's name prefix(es) to the routing system, so that the network can forward other peers' Interests towards it. There are 2 majors decisions to be made about a peer's routing announcements: 1) what is the granularity of the announced prefixes, and 2) when the peer makes the announcement.

About the first one, there are three potential cases: a peer can announce the name prefix of 1) the entire torrent ("`/NTORRENT/<torrent-name>`"), 2) each file in a torrent ("`/NTORRENT/<torrent-name>/file#`") or 3) each individual packet ("`/NTORRENT/<torrent-name>/file#/packet#`"); an option that cannot scale, since it results in extremely large FIBs. The relation among the prefixes is hierarchical (hierarchical NDN namespace), therefore, this decision involves a trade-off between the accuracy of the routing and forwarding plane and the number of announced prefixes; coarse-grained announcements reduce the size of FIBs but, depending on when the announcements are made (as explained below), they might result in extended path exploration to find the data or reduced amounts of time that a peer uploads its data.

About the second one, a peer can make a routing announcement: 1) before it downloads the data for the announced prefix (approximate), 2) after it downloads the data for the announced prefix (precise), 3) when it has downloaded a certain amount of data for the announced prefix. Since a peer can announce a prefix without having all its data (e.g., the prefix of a file in the torrent, without having all the packets in the file), this decision involves a trade-off among the required amount of path exploration to find the data, the amount of time that a peer uploads its data and the peer agility in the sense of enabling peers to download data fast from multiple sources.

### 3.2.6 Baseline Application Scenario

As illustrated in Figure 3.4, let us assume that a peer would like to download a Linux distribution torrent, which consists of 10 files and each file of 10 data packets.

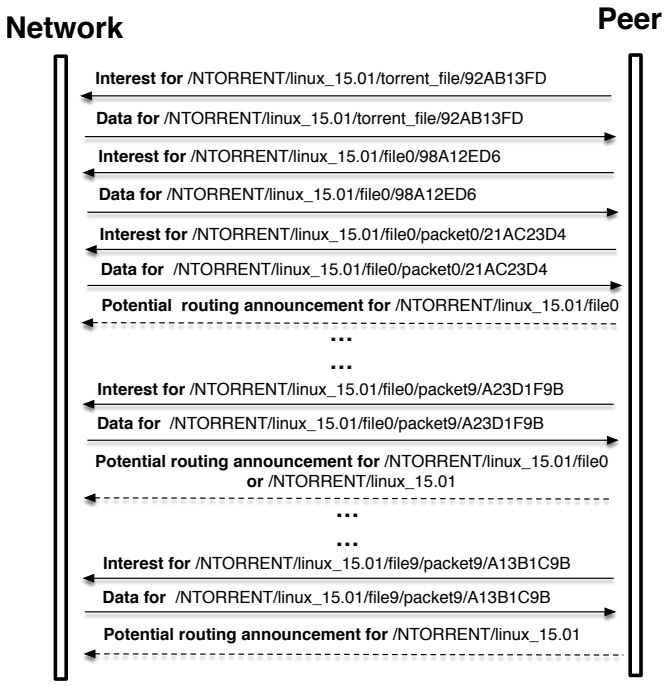



Figure 3.4: Example of downloading a linux distribution torrent

The peer acquires the .torrent file first, which is a data packet with a name like: "`/NTORRENT/linux_15.01/torrent_file/92AB13FD`" and contains the full name of the file manifests. The peer simply expresses an Interest for the .torrent file using its full name, while the network verifies the data packet integrity using this full name.

After receiving the .torrent file, the peer expresses Interests to acquire the desired file manifests that contain the full name of the data packets in each file (for example, the name of the manifest of the first file can be "`/NTORRENT/linux_15.01/file0/98A12ED6`"). When the peer has acquired the manifest of a specific file, it can start retrieving each individual data packet in this file by expressing an Interest for it using its full name (for example, the name of the first packet in the first file can be "`/NTORRENT/linux_15.01/file0/packet0/21AC23D4`").

Eventually, the peer will announce either the name prefix of a file in the torrent, or the prefix of the entire torrent. This announcement can be done when the retrieval of a file (or

the torrent respectively) is complete or when the first packet in a file (or the first file in the torrent) is retrieved.

## 3.3 Scaling nTorrent With Forwarding Hints

As mentioned in Section 2.1.3, broadly speaking, Interests carry "what" semantics (i.e., the name of the data they are looking for). The network helps the Interests bring data back by being aware of "where" semantics (i.e., where the requested data is located). The network becomes aware of the "where" semantics through routing announcements.

Routing directly on torrent names raises concerns about the system scalability when millions of torrents are available. To build a system that can scale up to millions of torrents, we leverage the notion of forwarding hint (Section 2.1.3). In the rest of this section, we first present a baseline scenario that we will use to elaborate on the design of scaling nTorrent with forwarding hints and then the specifics of our design.

In the rest of this section, we first present an overview of our design, then a baseline scenario, and, finally, we discuss each of the design components in more detail.

### 3.3.1 Design Overview

We assume that each peer is aware of the forwarding hint (i.e., routable prefix) of the network it resides on. We also assume the existence of a service (e.g., similar to the torrent services that exist today) with a routable prefix, which stores the forwarding hints of peers and helps them discover hints when necessary. The torrent producer initially generates the torrent and uploads its routable prefix to the service. Each peer that joins the swarm and is interested in the torrent will request the routable prefixes of other peers that have data for this torrent through the service. Peers can try different forwarding hints or attach multiple hints to the expressed Interests (Figure 3.6) to let the NDN network discover and retrieve the data from the closest location.

### 3.3.2 Baseline Scenario

We present our baseline scenario in Figure 3.5. There are 4 peers in total, one peer is within the "`/NIST`" network, one within the "`/FIU`" network, and 2 of them within the "`/UCLA`" network. There is also an nTorrent service instance with a routable prefix "`/nTorrent-service`". The name of a torrent is in the FIB of routers (i.e., routable) within each of these networks, while the name of each network is in the FIB of routers across the Internet.

Let us assume that peer 2 under "`/UCLA`" produced a torrent with a name "`/funny-video`" and peers 1, 3, and 4 would like to download it. After producing the torrent, peer 2 notifies the nTorrent service that torrent "`/funny-video`" can be found under the network "`/UCLA`". Peers 1, 3, and 4 perform an nTorrent service lookup to learn that in order to download the torrent "`/funny-video`", they need to use "`/UCLA`" as the forwarding hint of their Interests. As result, Interests for torrent data is forwarded based on the forwarding hint ("`/UCLA`") across the Internet and based on its name ("`funny-video`") within the UCLA network. Note that peer 3 is aware that its own forwarding hint is "`/UCLA`". As a result, peer 3 can identify that it is in the same network as peer 2, thus it does not need to attach the "`/UCLA`" forwarding hint to its Interests, since Interests within the UCLA network can reach peer 2 by being forwarded directly based on their names.

Let us assume that peer 1 under the "`/FIU`" network downloads the torrent from peer 2 after a while. At this point, peer 2 can notify the nTorrent service that torrent "`/funny-video`" can be found under the network "`/FIU`" as well. As a result, future service lookups will return both the "`/FIU`" and the "`/UCLA`" forwarding hints to peers interested in this torrent. The information about which torrent can be found under which network is soft-state on the service end. To this end, both peers 2 and 3 need to periodically refresh their forwarding hint at the service to ensure that it will not expire and that the service will continue to return their hint in response to future lookups for torrent "`/funny-video`". Note that a forwarding hint is fundamentally different than explicitly managing point-to-point connections as imposed by TCP/IP to BitTorrent, since under a single network, multiple peers and torrents may be available (e.g., 2 peers are available under the "`/UCLA`" network, while, in addition to torrent

"`/funny-video`", these peers may have data for other torrents too).

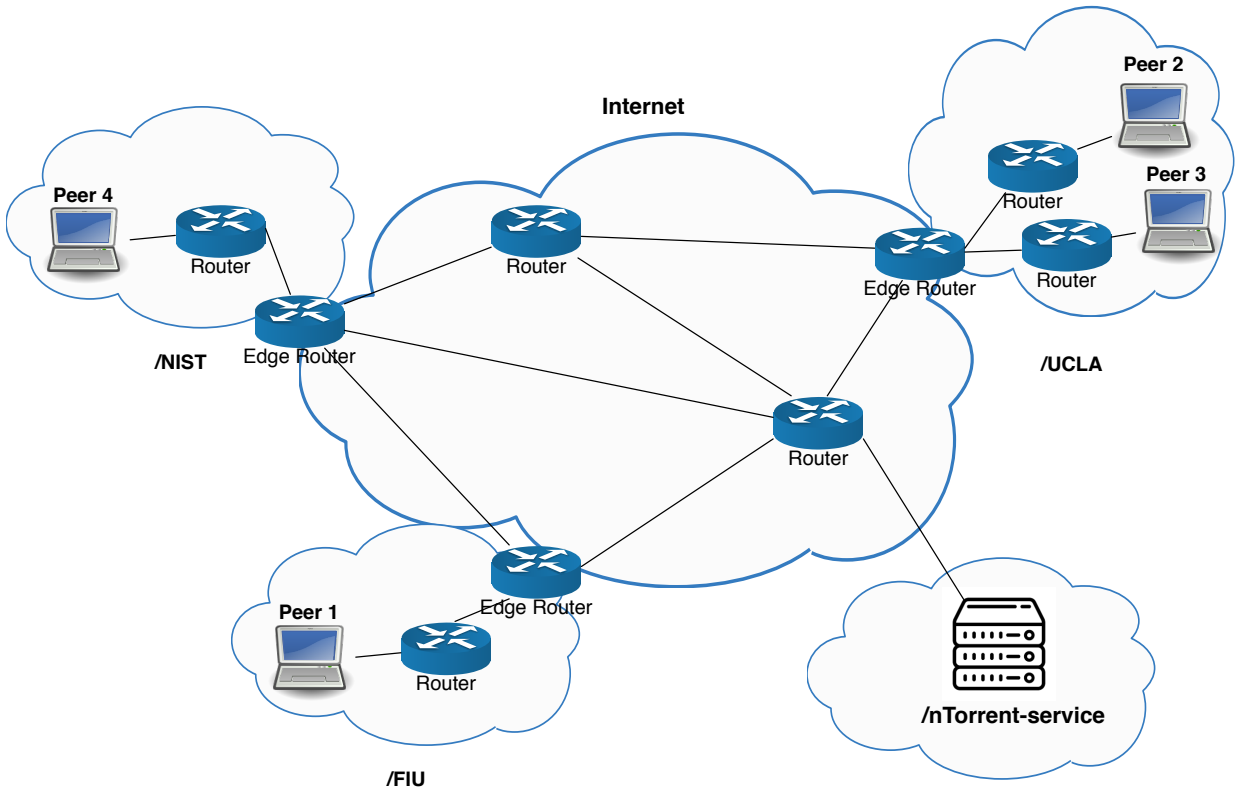



Figure 3.5: Scaling nTorrent with forwarding hints baseline example

### 3.3.3 nTorrent Service

The nTorrent service is responsible for maintaining information on which torrent is available under which network (in other words which torrent can be retrieved by using which forwarding hint). This information is soft-state; peers are responsible for refreshing their forwarding hint at the service, while if a forwarding hint is not refreshed after the soft-state expires, the service deletes it. For operational purposes, we assume that the service has a well-known and globally routable prefix.

The nTorrent service can be offered by stakeholders that would like to make their files available to others for sharing. Note that this service is different than the tracker service of

BitTorrent. The nTorrent service provides some general directions (in the form of forwarding hints) toward torrent data; multiple peers can be available toward each direction (e.g., 2 peers are available under the "`/UCLA`" network in the example of Figure 3.5). On the other hand, a BitTorrent tracker keeps track of specific peers (a set of IP addresses) that can provide data for a torrent.

### 3.3.4 Discovery of Routable Prefixes

The producer of a torrent initially registers its own routable prefix (e.g., "`/UCLA`" for peer 1 of Figure 3.5) to the service to indicate that a specific torrent can be found through a specific forwarding hint. When a new peer joins the swarm and would like to download a torrent, it sends a "lookup" request to the service for this torrent. The service receives this request and sends back the forwarding hint(s) under which data for a specific torrent can be found.

Peers can insert their own forwarding hint to the service (or refresh its soft-state if already inserted to the service) through either implicit or explicit notifications to the service.

- **Implicit notifications:** peers perform a lookup for forwarding hints, they report their own forwarding hint to the service even if they do not have all the data for a torrent.
- **Explicit notifications:** peers send their forwarding hint to the service only when they have downloaded all the data for a torrent.

### 3.3.5 Torrent Data Fetching

After performing a service lookup and receiving one or more forwarding hints from the service, peers can proceed to downloading data for the desired torrent. To achieve traffic localization, peers first try to find local peers (i.e., peers residing within the same network as the requester, thus having the same forwarding hint as the requester). Therefore, they express a request for torrent data without specifying a forwarding hint. If local peers are available, data fetching through them is prioritized. If no local peers are available, peers attach to their Interests a forwarding hint that contains one or more routable prefixes acquired through the

service (as shown in Figure 3.6), so that their Interests can reach peers across the Internet.

### 3.3.6 Forwarding Hints & Individual Peer Disconnections

The use of forwarding hints can partially hide disconnections/departures of individual peers along two different directions: (i) routers within a specific network (e.g., "`/UCLA`") based on the nTorrent forwarding strategy (Section 3.2.4) can adaptively forward Interests to different peers within their network by trying one or more alternative next hops when a selected next-hop cannot bring data back (e.g., due to a peer disconnection), (ii) requesting peers can attach more than one forwarding hints to the expressed Interests, which will be initially forwarded toward the direction of the closest attached hint to the requester. If the data is not available toward the first forwarded direction (e.g., because all the peers in this network have been disconnected), Interest can be forwarded based on a different attached forwarding hint toward an alternative direction[1].

For example, in Figure 3.6, let us assume that an Interest is forwarded toward the "`/UCLA`" direction and within the "`/UCLA`" network this Interest is initially forwarded to peer 3. However, peer 3 has been disconnected, therefore, the router attached to peer 3 detects that no data is coming back for this Interest. In this case, based on the nTorrent forwarding strategy, the router will send a NACK back to the edge router, which will then forward the Interest to the router toward peer 2. Moreover, let us assume that an Interest that carries multiple hints ("`/UCLA`" and "`/FIU`") is initially forwarded toward the "`/UCLA`" direction. The Interest enters the "`/UCLA`" network and is forwarded to peer 3 that has been disconnected and then to peer 2, who we assume that has also been disconnected. The "`/UCLA`" edge router becomes aware that no more peers are available in this network by receiving a NACK from both its next-hops (a NACK from the router toward peer 3, where the Interest was forwarded first and a NACK from the router toward peer 2, where the Interest was forwarded after peer 3). As a result, the "`/UCLA`" edge router will forward the Interest based on the second attached

[1]This approach requires a slight modification of the way that forwarding hints are handled once Interests enter the so called "producer region". Instead of being stripped off by the boarder router of this region, the forwarding hints should remain intact to facilitate cases, where Interest forwarding based on a different hint will be needed.

forwarding hint ("/FIU"). Eventually, this Interest based on the "/FIU" forwarding hint will reach the "/FIU" network and retrieve data from peer 1.

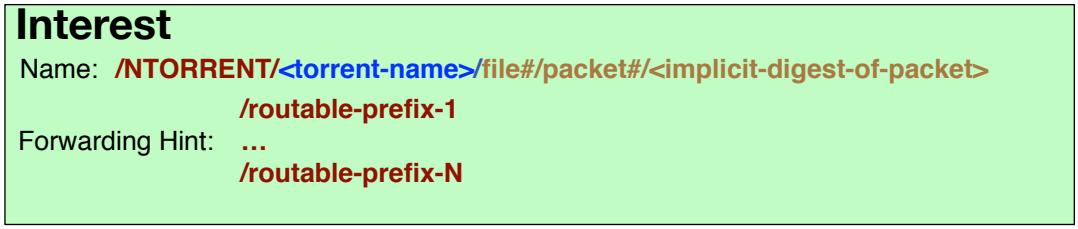



Figure 3.6: nTorrent Interest format

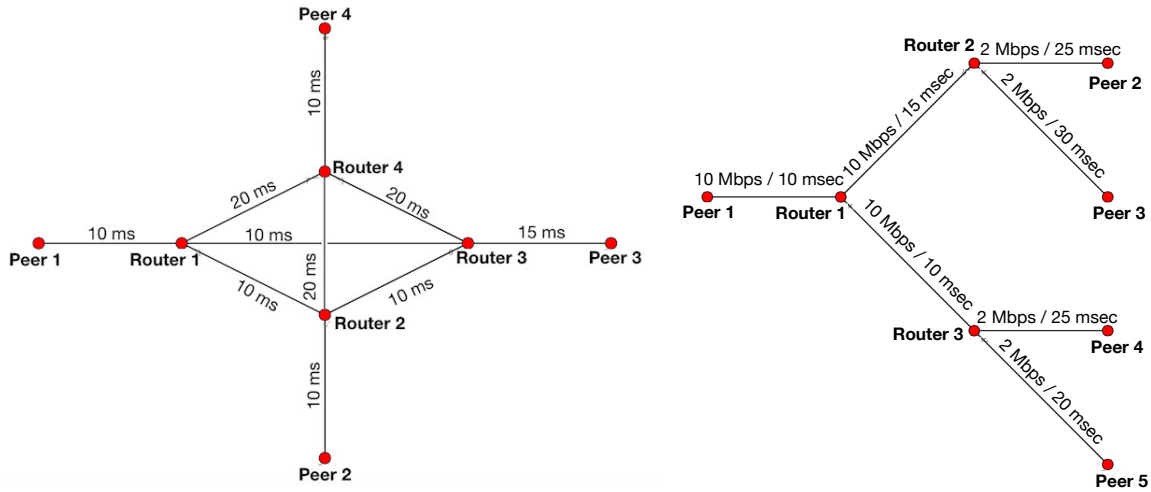
## 3.4 Experimental Evaluation

In this section, we perform a simulation study of nTorrent. Our goal is to examine the tradeoffs of its design and compare its performance with BitTorrent. Our study focuses on 3 aspects: 1) impact of the routing announcements on the application performance to explore the tradeoffs mentioned in section 3.2.5, 2) utilization of the NDN forwarding plane by nTorrent to achieve multi-path forwarding and download speed maximization, 3) swarm performance in flash crowd scenarios to study the impact of in-network caching in the case of simultaneous downloads.

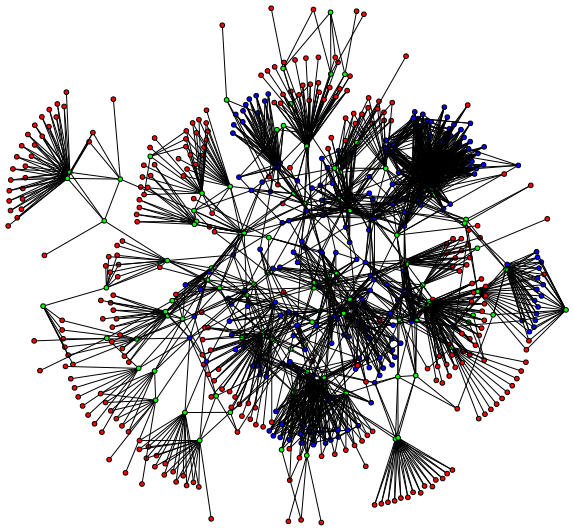
The simulation scenarios were implemented in ndnSIM 2 [MAZ17, MAM16, MAM15], an NDN simulator based on NS-3 [ns3], featuring the forwarding pipelines of the NDN Forwarding Daemon (NFD) [AS15]. We also implemented the plain BitTorrent functionality with a rarest-piece-first selection strategy as a separate NS-3 module. We should note that nTorrent has been deployed and can be used on the NDN testbed (Section 3.4.6)

### 3.4.1 Simulation Setup

We use the topologies shown in Figure 3.7. The first one (Figure 3.7a) offers multiple paths with different costs (in terms of delays) to peers (the bandwidth of the links is the same). The second one (Figure 3.7b) has bottleneck links to the majority of peers. These topologies were used to study the specifics of the nTorrent design and the benefits of NDN forwarding.




(a) Topology with router node degree equal to 4 (b) Topology with router node degree equal to 3



(c) Rocketfuel AT&T topology

Figure 3.7: Simulation topologies

Figure 3.7c illustrates the Rocketfuel AT&T topology, consisting of 625 nodes and 2,101 links. This topology represents a large Autonomous System (AS) and was used to study our flash crowd scenario and the approach of scaling nTorrent with forwarding hints.

We assume that the .torrent file is known by all the peers, and the torrent to be downloaded contains 100MB data and consists of 100 files. For the nTorrent simulations, each file consists of 1000 data packets, and for the BitTorrent simulations, of 1000 pieces. Each piece is retrieved as a single packet (in both nTorrent and BitTorrent, the size of each packet

is 1KB). Unless otherwise stated, background traffic is generated (30% of the total traffic with Poisson distribution acting as link capacity nuisance and cache pollution) to study the system behavior under semi-realistic network conditions.

We experimented with the following two cases of routing announcements by peers:

- **Precise announcements on a per file basis**: peers register the prefix of a file in the torrent for routing announcements when they download all the packets in the file.
- **Approximate announcements on a per torrent basis**: peers register the prefix of a torrent for routing announcements when they have a complete file in the torrent.

Routers' FIB is initially populated by a link-state routing protocol that uses the Dijkstra's algorithm. Routers also use a forwarding strategy, which probes faces to discover potential better next-hops for data retrieval. The implemented probing algorithm sets one timer per router face. When the timer of a face $F$ expires, the next Interest that reaches the router, and there is a FIB entry with $F$ as an outgoing face, is forwarded through $F$. When data does not come back from a probed face (or a NACK comes back), an exponential back-off timer specifies when the router will probe this face again. The initial value of the timer is 1 second and the maximum 32 seconds.

In the BitTorrent experiments, the peers follow the rarest-piece-first strategy, while for nTorrent, they request packets sequentially as explained in section 3.2.3.

### 3.4.2 Impact Of Routing Announcements

We first study the impact of routing announcements on nTorrent. We measure the amount of NACKs generated and the time needed for data retrieval. We use the topology illustrated in Figure 3.7a, where a seeder initially uploads a torrent and some leechers download it. We have disabled in-network caching to examine the worst-case scenario of the nTorrent performance.

Peer 4 acts as the seeder and announces prefixes across the network. The announcements propagate among the routers due to the routing protocol. Peer 1 starts downloading data

first. When it downloads 40% of the torrent, peer 3 starts downloading data. When peer 3 downloads 40% of the torrent, peer 2 starts its downloading; both peers 1 and 3 still act as leechers allowing us to study the effect of NDN forwarding.

For precise routing announcements, no local error recovery is required (Table 3.1); all the data can be retrieved through each outgoing FIB entry face.

For approximate routing announcements, local error recovery is low (Table 3.1), since the forwarding plane avoids constantly probing faces that cannot bring data back due to the exponential back-off timer. Specifically, peer 1 initially downloads data from the seeder. When peers 2 and 3 download the first file in the torrent, they announce the torrent prefix. Therefore, faces at the routers towards them become available and are probed by the forwarding plane. At that time, peer 1 is requesting data close to or even beyond the torrent midpoint, while peers 2 and 3 have files at the beginning of the torrent (sequential data fetching). In the same manner, peer 3 initially downloads data from peer 1, but when peer 2 announces the torrent prefix, a few requests of peer 3 are used as probes towards peer 2 that still has files at the beginning of the torrent. When peer 2 joins the swarm though, sequential fetching "masks" off part of the negative effect of the approximate announcements; the data is already available at peers 1, 3 and 4, therefore peer 2 downloads the torrent without triggering any local error recovery.

Table 3.1: Local error recovery amount (percent of peer Interests resulted in NACKs)

| **Peer** | **Precise routing announcements (per file)** | **Approximate routing announcements (per torrent)** |
|---|---|---|
| Peer 1 | 0 % | ~0.009 % |
| Peer 2 | 0 % | 0 % |
| Peer 3 | 0 % | ~0.005 % |

The torrent distribution duration is illustrated in Table 3.2. This duration is measured from the moment the first peer starts downloading data until the last peer downloads a copy of the torrent. The results for both the cases of routing announcements are close, since the required local error recovery amount for approximate announcements is low.

Table 3.2: Duration until all the peers download the torrent

| | |
|---|---|
| **Precise routing announcements (per file)** | $124 \pm 2$ sec |
| **Approximate routing announcements (per torrent)** | $127 \pm 3$ sec |

### 3.4.3 Multi-path Forwarding And Download Speed Maximization

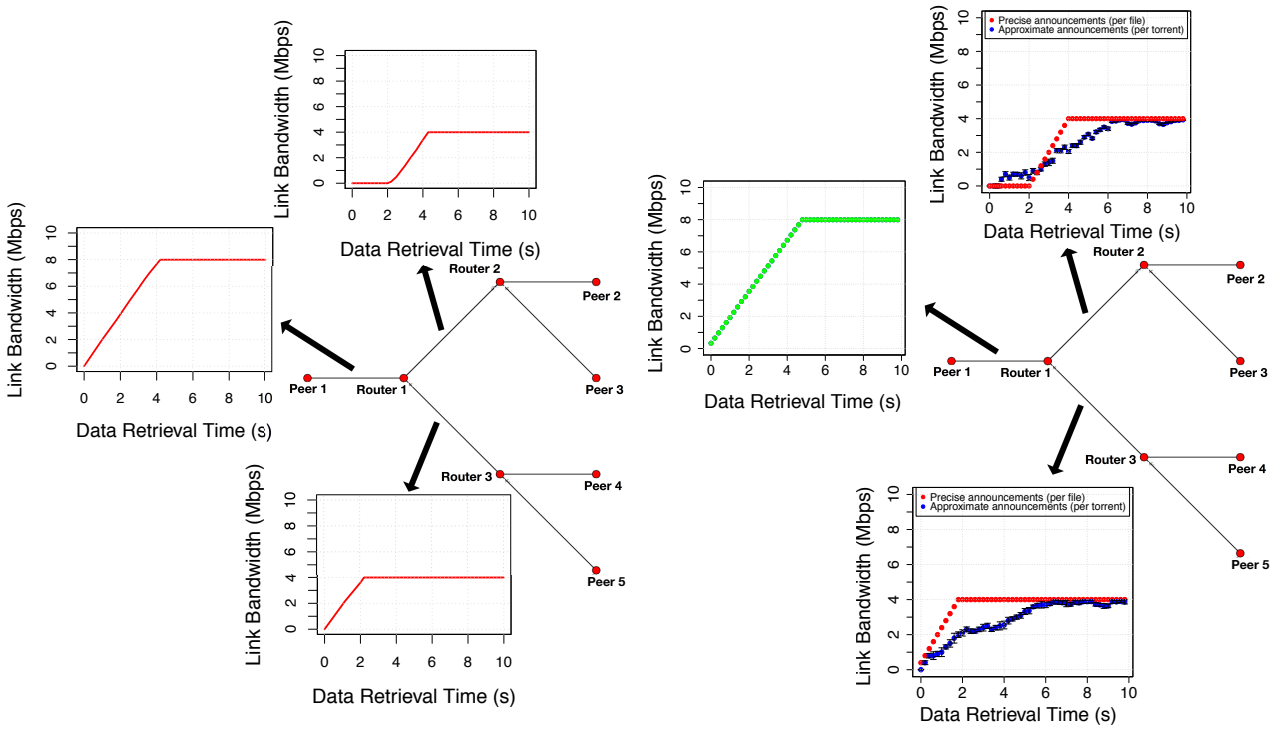
To study how NDN multi-path forwarding enables the utilization of all the available paths and the maximization of download speed by nTorrent, we use the topology illustrated in Figure 3.7b. Peer 1 expresses Interests for packets in the torrent with a rate that linearly increases the utilization of the link reaching router 1. We have disabled the generation of background traffic to focus on the speed achieved by nTorrent.

We first consider the case, where peers 2, 3, 4 and 5 act as seeders that have the entire torrent content. In Figure 3.8a, we illustrate the bandwidth of the links between peer 1 and router 1, router 1 and each of routers 2 and 3 respectively. The results are the same for both cases of routing announcements. When a router receives more Interests than it can forward, it sends a NACK downstream to control the incoming rate. Therefore, nTorrent can fully utilize all the available paths starting from the one with the best data plane performance (RTT). When a path is fully utilized, Interest forwarding is expanded to the path with the next best performance; all the previously selected paths stay fully utilized and the remaining traffic is forwarded to the new one.

We conducted the same experiment with peers 2, 3, 4 and 5 acting as leechers; we randomly distributed the torrent packets among them, so that each packet is available at exactly one leecher. As illustrated in Figure 3.8b, the achieved bandwidth approaches the maximum possible, while the convergence to this maximum is faster for precise announcements, since no local error recovery is required.

### 3.4.4 Flash Crowd Scenario

In this experiment, we simulate a flash crowd scenario (topology 3.7c). Such scenarios are challenging for BitTorrent, since the upload bandwidth of a seeder is dominated by leechers'




(a) nTorrent download speed (seeder peers having all the torrent content)

(b) nTorrent download speed (leecher peers)

Figure 3.8: nTorrent download speed results

requests. As a result, the seeder serves data to only a subset of the requesting peers, while the rest of them have to wait until data is replicated and served by others. For nTorrent, we expect that in-network caching can reduce the number of requests forwarded to peers, thus reducing the overall download time when simultaneous downloads take place.

We randomly select 1 initial torrent seeder and leechers at random positions. We vary the number of leechers and the size of in-network caches (Least Recently Used (LRU) replacement policy) to study the impact of caching and simultaneous downloading to the overall performance of the swarm. In Table 3.3, we present the download time until all the peers download a copy of the torrent and we compare the results to the ideal case (every peer downloads data from the closest, in terms of RTT, peer or cache) and BitTorrent. In Table 3.4, we present the percent of requests received by peers in nTorrent (in BitTorrent, all the requests are received by peers). Below, we present two of our experiments as examples

Table 3.3: Duration until all the peers download the torrent (seconds)

| | 25 leechers | | | 50 leechers | | | 100 leechers | | |
|---|---|---|---|---|---|---|---|---|---|
| Cache size | 50 MB | 100 MB | 200 MB | 50 MB | 100 MB | 200 MB | 50 MB | 100 MB | 200 MB |
| **Ideal Case - nTorrent** | ~171 | ~171 | ~171 | ~195 | ~195 | ~195 | ~213 | ~213 | ~213 |
| **Precise announcements** | $204 \pm 2$ | $197 \pm 2$ | ~176 | $225 \pm 2$ | $211 \pm 1$ | ~199 | $244 \pm 2$ | $224 \pm 1$ | ~214 |
| **Approximate announcements** | $216 \pm 3$ | $210 \pm 2$ | ~181 | $238 \pm 2$ | $222 \pm 2$ | ~202 | $259 \pm 2$ | $233 \pm 1$ | ~216 |
| **BitTorrent** | $225 \pm 3$ | $225 \pm 3$ | $225 \pm 3$ | $261 \pm 3$ | $261 \pm 3$ | $261 \pm 3$ | $319 \pm 3$ | $319 \pm 3$ | $319 \pm 3$ |

Table 3.4: Percent of requests received by peers

| | 25 leechers | | | 50 leechers | | | 100 leechers | | |
|---|---|---|---|---|---|---|---|---|---|
| Cache size | 50 MB | 100 MB | 200 MB | 50 MB | 100 MB | 200 MB | 50 MB | 100 MB | 200 MB |
| **nTorrent** | ~55 % | ~28 % | ~16 | ~33 % | ~16 % | ~7 | ~19 % | ~7 % | ~2 % |

to elaborate on those results.

For our first experiment, we gradually increase the number of leechers and keep the CS size constant. We randomly select 25 leechers, while the cache size is 100 MB. nTorrent achieves faster data downloading than BitTorrent and almost 4 times fewer requests reach the peers.

We randomly select additional leechers to reach a total number of 50 and 100 downloaders respectively. The results show that the larger number of simultaneous downloaders of the same torrent has a beneficial effect on the overall download time (it gets closer the ideal case), since the data is available in more caches across the network. Therefore, fewer requests are satisfied by peers.

For our second experiment, we keep the previous distribution of 100 peers and vary the size of in-network caches. We first run our simulation with caches of 50 MB. The results show that the nTorrent download time diverges from the ideal case. We observed that the LRU replacement policy in combination with sequential fetching by peers can result in either constructive or destructive caching depending on whether peers fetch cached data before background traffic evicts cached torrent data. Specifically, if peers fetch data before any evictions take place, the LRU policy evicts background traffic (constructive caching). If evictions happen before the peers fetch cached data, the LRU policy evicts torrent data,

Table 3.5: Total Generated Traffic (Gigabytes)

| | 100 leechers | | |
|---|---|---|---|
| Cache size | 50 MB | 100 MB | 200 MB |
| **nTorrent - Precise announcements (per file)** | ~33.7 | ~30.5 | ~28.9 |
| **nTorrent - Approximate announcements (per torrent)** | ~34.5 | ~31.1 | ~29.7 |
| **BitTorrent** | ~65.8 | ~65.8 | ~65.8 |

and, since peers request data sequentially, their requests result in consecutive cache misses (destructive caching).

We increase the size of caches to 200 MB; now there is enough space in cache for both torrent data and background traffic, therefore, the download time approaches the ideal. The data is served almost exclusively by caches and only about 2 % of the requests are satisfied by peers.

In Table 3.5, we present the total traffic amount generated by nTorrent and BitTorrent for 100 peers and varying cache size. The results show that nTorrent generates about half the traffic compared to BitTorrent even if the cache size is smaller than the size of the torrent.

Overall, peers achieve fast data retrieval because of in-network caching when simultaneous downloads take place. As the number of simultaneous downloaders grows, the download time approaches the ideal. Cache size also plays an important role in the swarm performance, since it reduces the generated traffic amount and the number of requests served by peers.

### 3.4.5 Scaling nTorrent Through Forwarding Hints

#### 3.4.5.1 Feasibility Study - Naive Topology

To examine the feasibility of the design, we conducted initial experiments using the topology 3.7a. We divide this topology into 4 separate network domains (at the topology edges) and a backbone network. We deploy a nTorrent service instance in the middle of the topology. We simulate two cases of churn for peers: (i) light churn, where the peer inter-arrival gap is 200 seconds and the peer availability is 95%, and (ii) heavy churn, where the peer inter-arrival gap is 100 seconds and peer availability is about 60% (peers might get discon-

nected a number of time during the file downloading process, on average 50 seconds each time). For this experiment, we have disabled in-network caching.

In Figure 3.9, we present the impact of the soft-state timer value for the routable prefixes stored by the service on the average torrent download time by peers. Our results show that if the soft-state duration of the routable prefixes is lower or equal to the inter-arrival gap of peers, there might be times, where peers perform a lookup for routable prefixes, but the service does not have any prefixes currently available. However, when the soft-state duration is longer than the peer inter-arrival gap, the service will always have prefixes available to help peers bootstrap when needed.

The option of having peers provide implicit or explicit notifications to the service does not result in significant difference. This is due to the fact that multiple peers may be available under a single routable prefix. As a result, even if peers that do not have all the file data inform the service about their routable prefix, there might be others under this prefix that have complete data.

In Figure 3.10, we present the download time for nTorrent and an IP-based Distributed Hash Table (DHT) solution, following the Chord design. The results demonstrate that nTorrent takes advantage of the stateful NDN forwarding plane to deal with peer disconnections and churn, since under a single routable prefix, multiple peers may be available. To this end, it achieves faster download times than DHT, which runs as an overlay on top of the point-to-point IP network. During peer disconnections, the DHT overlay might temporarily break, as a result peers cannot discover the desired data through this overlay. Moreover, DHT as an overlay has no knowledge about how far the communicating peers might be. Contrary to that, nTorrent utilizes the NDN forwarding plane to discover data sources close to the requester.

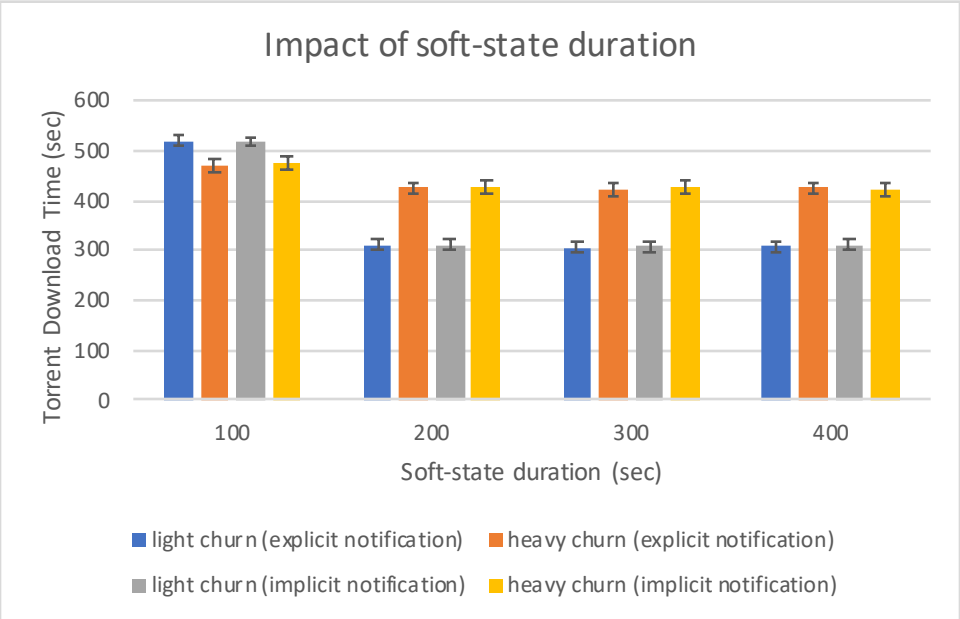



Figure 3.9: Scaling nTorrent through forwarding hints - Download time

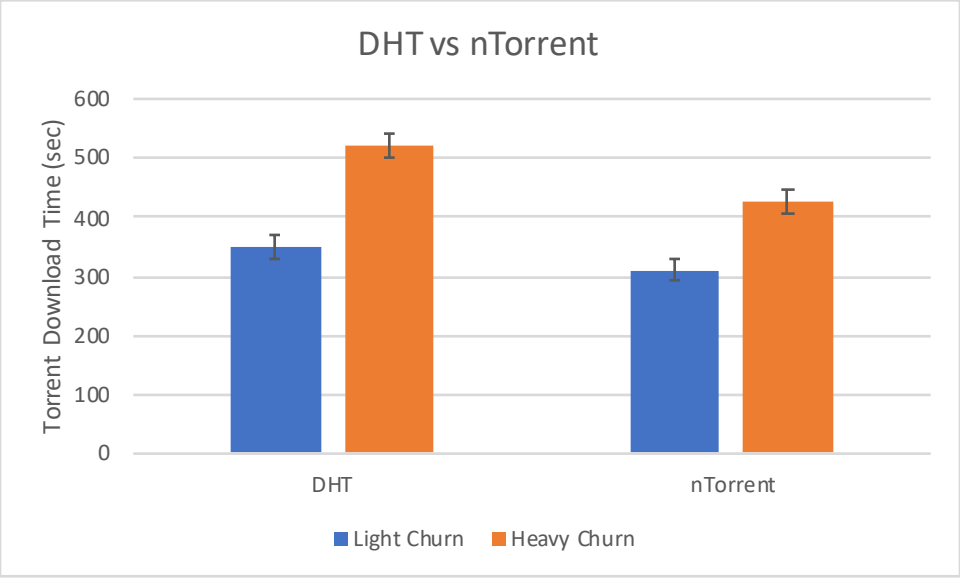



Figure 3.10: Comparison between IP-based DHT and NDN-based nTorrent

#### 3.4.5.2 Large Topology

In this experiment, we use the topology of Figure 3.7c, which we divide into 10 separate network domains (at the topology edges, each of size 10-15 nodes) and a backbone network[2]. We assume that only globally routable names (i.e., names of the domains) are in the FIB of routers across the backbone network and we deploy a nTorrent service instance approximately in the middle of the backbone network. We simulate two cases of churn for peers: (i) light churn, where the peer inter-arrival gap is 200 seconds and the peer availability is 95%, and (ii) heavy churn, where the peer inter-arrival gap is 100 seconds and peer availability is about 60% (peers might get disconnected a number of time during the file downloading process, on average 50 seconds each time). The CS size is 100MB, while the FIFO replacement strategy is used.

In Figure 3.11, we present the impact of the soft-state timer value for the routable prefixes stored by the service on the average torrent download time by peers (implicit notifications). Our results follow the same trend as the ones presented in Section 3.4.5.1. The results also indicate that heavy churn results in roughly 15-20% higher download times. This is due to two major reasons: (i) the network needs additional time to discover peers that have the data and are available at a specific moment in time. Based on our experiments, we conclude that if at least a single peer is available under a routable prefix for a given data packet, the network will be able to discover this peer and retrieve the requested data, and (ii) even if the retrieval of data under a specific routable prefix fails, the forwarding plane uses other prefixes attached to the Interest, if any, or notify the requester with a NACK. In this case, the requester selects another routable prefix and re-expresses the Interest. To this end, peers should, in general, identify and use a set of working routable prefixes, while the network will take the responsibility to locate and retrieve the closest copy of the requested data.

In Figure 3.12, we present the torrent download time for a varying number of forwarding

[2]We conducted experiments for a variable number of network domains at the edge of the topology, as well as size of such network domains. The conclusion of these experiments was that if the overall number of peers is the same, the results follow the same trend. Increasing the number of peers would result in higher performance for nTorrent due to the increased utilization of in-network caching by peers that download the torrent at the same time.

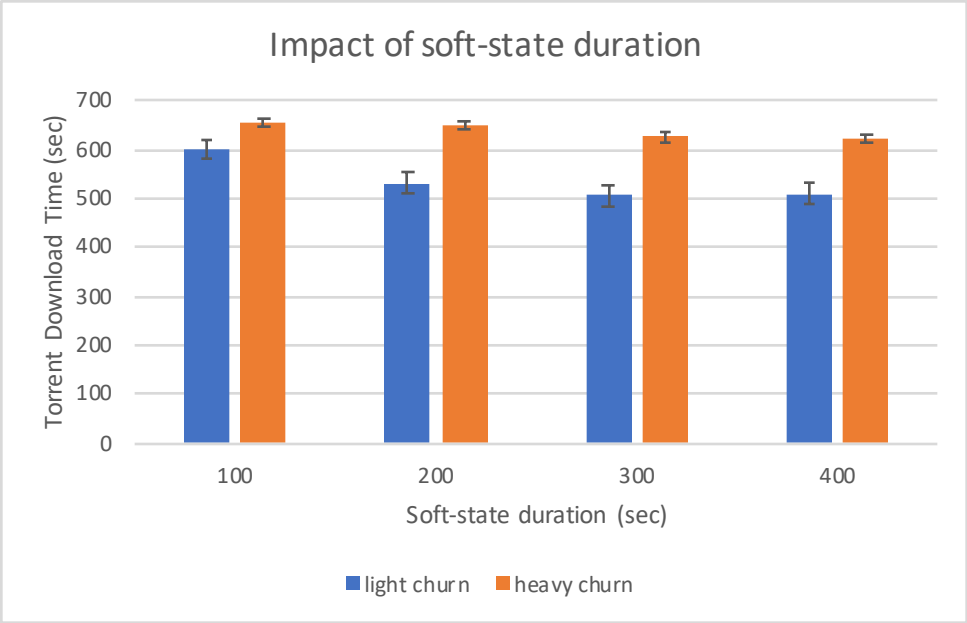



Figure 3.11: nTorrent - Download time (single forwarding hint)

hints attached to the Interests expressed by peers. The results show that multiple forwarding hints provide more flexibility to the NDN forwarding plane to discover the requested data in case of peer disconnections and churn. Even if the requested data is not available under a specific routable prefix attached to an Interest, the forwarding plane can search for the data under a different routable prefix in the Interest. However, the download time does not significantly decrease as we increase the number of attached hints (e.g., four forwarding hints). This indicates that most of the data can be found under the second or the third routable prefix attached to the Interests.

In Figure 3.13, we compare the performance of nTorrent to an IP-based DHT solution. The results show that the nTorrent achieves faster downloads than DHT. There are three reasons for that: (i) nTorrent takes advantage of the stateful NDN forwarding plane to deal with peer disconnections and churn. For example, multiple peers may be available under a single routable prefix, while peers can also attach multiple prefixes to the expresses Interests in order to help the network discover data under multiple domains. On the other hand, DHT often breaks when peers get disconnected, (ii) nTorrent utilizes NDN in-network caching, while DHT cannot, and (iii) as an overlay, DHT is unaware of how close or far the

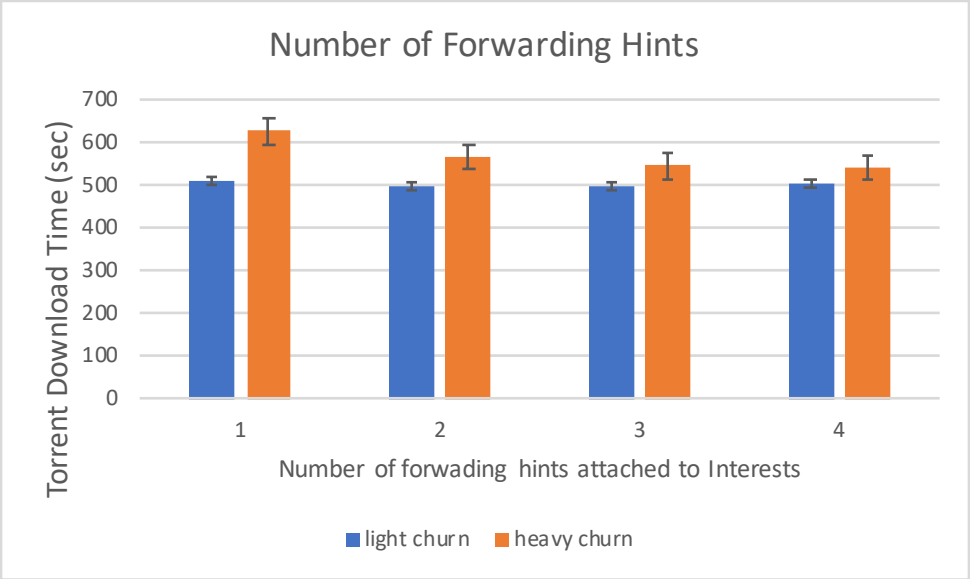



Figure 3.12: Torrent download time for a variable number of forwarding hints attached to expressed Interests

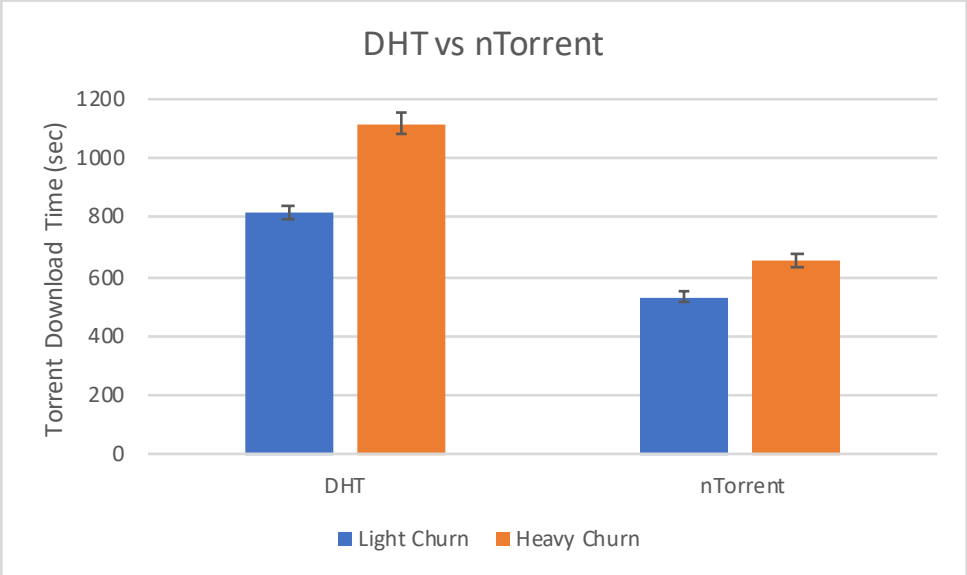



Figure 3.13: Comparison between IP-based DHT and NDN-based nTorrent (single forwarding hint)

communicating peers are, however, the NDN network discovers the copy of requested data closest to the requesting peer.

On the other hand, the DHT implementation, following the Chord design, performs discovery of individual peers at the application layer. Specifically, a peer needs to contact a number of other peers that will eventually redirect it to someone that has the desired data. If any peer during this redirection process goes offline, the DHT breaks and the discovery process fails until it recovers and the peers are connected again. This process requires more time than the in-network data discovery performed by the NDN forwarding plane.

In Figure 3.14, we present the overhead of the nTorrent service approach for a varying soft-state duration of the routable prefixes on the service side. Peers periodically (with a period of 50 seconds) perform a service lookup to discover potentially new routable prefixes. Moreover, when peers cannot reach others for data downloading, they perform a service lookup every 10 seconds. The results show that as we increase the inter-arrival gap among peers to be greater than the soft-state duration of the prefixes, the overhead decreases. This is due to the fact that the service stores the routable prefixes of the peers long enough, so that when new peers join, they can discover others right away and start downloading the torrent from them.

In Figure 3.15, we compare the overhead of nTorrent to the DHT overhead. In DHT, peers need to periodically (with a period of 10 seconds) stabilize DHT overlay in order to deal with peer disconnections and failures. DHT stores items on a per torrent piece granularity. When peers download a piece from another peer, they also exchange the pieces that each one has. If the desired piece is not available among known peers, peers have to perform discovery again through the DHT. Moreover, peers may also need to perform discovery again when the peer they are connected to goes offline. For all these reasons, DHT results in about 3x higher overhead than nTorrent, where a routable prefix may guide Interests to multiple individual peers, thus peers need to query the service in order to discover new prefixes at a relatively low frequency.

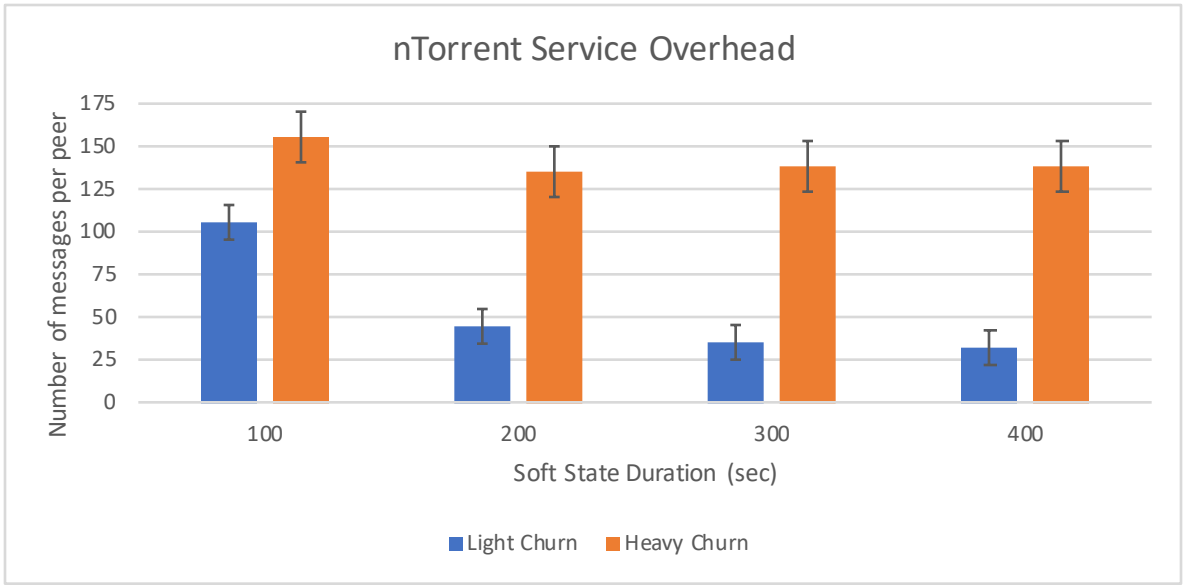



Figure 3.14: Overhead of nTorrent service

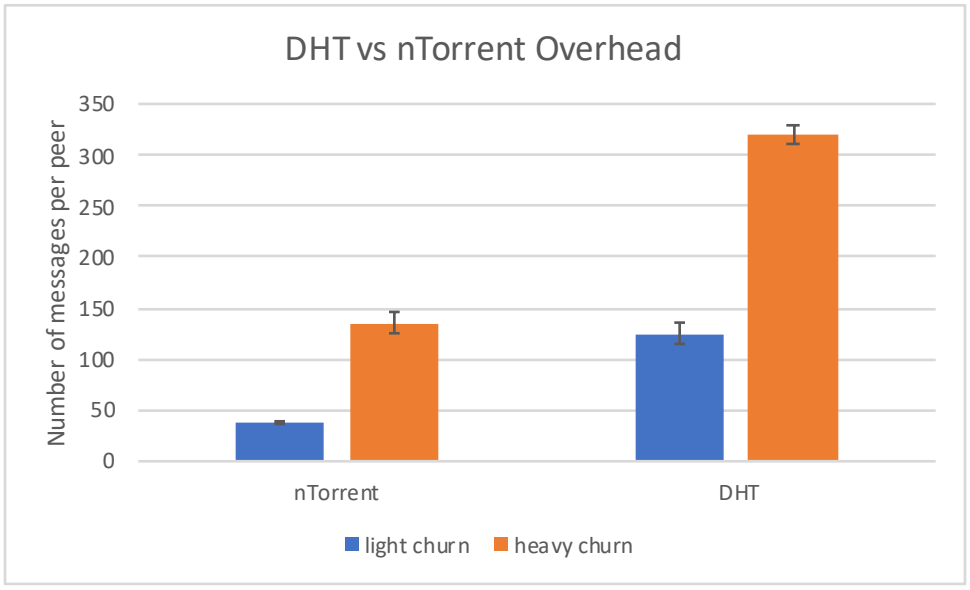



Figure 3.15: Overhead comparison between IP-based DHT and NDN-based nTorrent

| nTorrent | Download time per torrent (s) | FIB size (entries) |
|---|---|---|
| Forwarding directly on application names | 518.5 | 100 |
| Forwarding hints | 521.2 | 10 |

Table 3.6

#### 3.4.5.3 Comparison to Forwarding Directly Based on Application Names

We repeat the above experiment with the large rocketfuel AT&T topology for nTorrent, where routing/forwarding of Interests happen directly on nTorrent data names (not forwarding hints). We compare the average FIB size of routers and the download time for 100 torrents to the nTorrent design that uses forwarding hints (Table 3.6). The results indicate that in terms of downloading time both designs offer almost the same performance. In terms of the forwarding state that routers need to maintain, in the case of routing directly on data names routers have one FIB entry per torrent, however, in the forwarding hint based design, one FIB entry is needed for each globally routable prefix.

### 3.4.6 Testbed Deployment

We have performed small scale deployment experiments of nTorrent (2-8 peers) on the NDN testbed, which consists of 31 nodes and 84 links (Figure 3.16). The goal of this deployment was primarily to validate the design features of nTorrent. More specifically:

- We developed and tested a segmentation mechanism of real large files (e.g., Linux distribution images) into NDN packets. In this way, peers can split large files into network-layer data packets, as well as can name and sign these packets properly.
- We validated that torrent-files and file manifests can be encoded/decoded and exchanged correctly, so that peers are able to discover the names of the files in a torrent

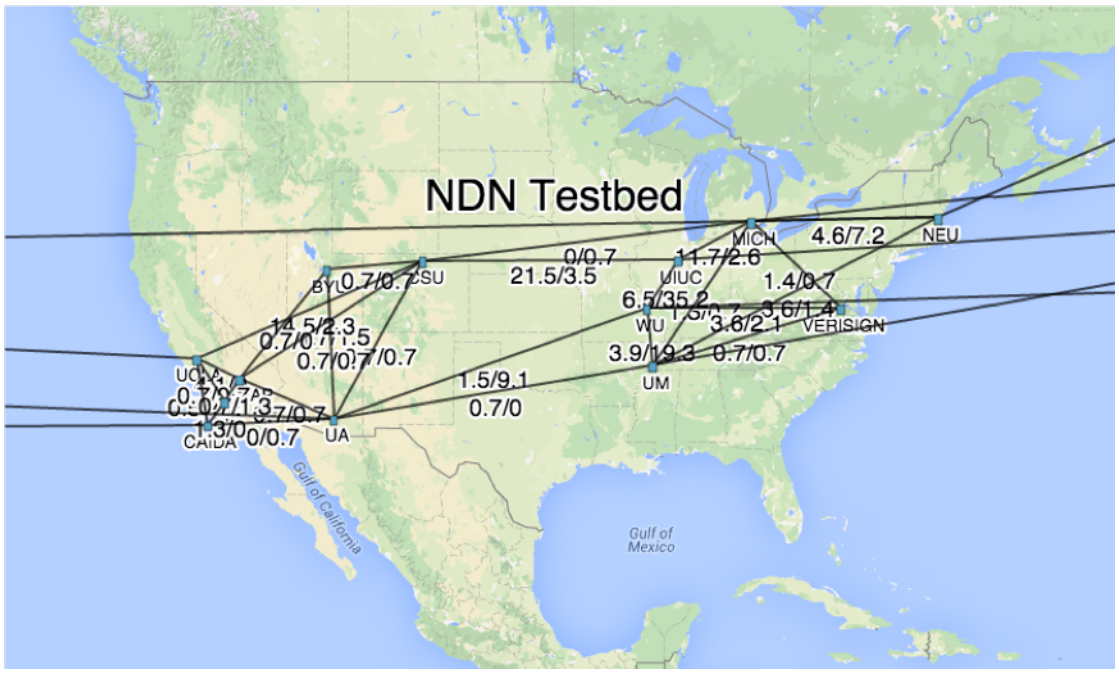



Figure 3.16: NDN testbed topology

and the data packets in a file.

- We tested and evaluated the sequential data fetching mechanism.
- We verified that in cases of simultaneous downloads, peers can take full advantage of NDN in-network caching to reduce the traffic on the testbed and the load on peers that have the requested data.
- We verified that routing announcements are properly propagated on the testbed and experimented with different granularities.
- We validated the design of scaling nTorrent with the use of forwarding hints.

## 3.5 Discussion

### 3.5.1 Open Issues

In this section, we discuss remaining open issues with the current nTorrent design. The most critical open issue of the nTorrent design might be the fact that it does not offer mechanisms to incentivize peers to share their data as BitTorrent does through the tit-fo-tat mechanism. Specifically, nTorrent peers might receive Interests for torrent data they have announced to the network that they have, but simply ignore them. As a result, peers that are willing to share data may be excessively loaded by receiving and responding to requests, due to the fact that other peers might not be willing to serve requests for torrent data. Early on in our design process, we speculated that in-network caching and request aggregation could mitigate this problem. Later on, we realized that this speculation might not be entirely true, since in-network caching is opportunistic and Interest aggregation applies only to cases that Interests are received almost simultaneously by the same router. To this end, peers could exploit the nTorrent system by expressing excessive requests for torrent data, without having to contribute by responding to requests from other peers.

One way that the data sharing incentives could be enforced is through the forwarding strategy component. Specifically, the nTorrent forwarding strategy on routers that are directly connected to peers could keep track of the forwarded Interest rate and the received data rate to detect peers that receive requests for torrent data, but do not respond to them. In such cases, routers could rate limit requests that come from peers not willing to share torrent data, offering functionality similar to the tit-fo-tat mechanism of BitTorrent.

Another open issue to investigate is whether an abstraction similar to the BitTorrent piece (i.e., an abstraction that groups together multiple network layer packets, without identifying an entire file) is needed. Such an abstraction could potentially offer a feasible alternative for local routing announcements of torrent pieces. These announcements will be of higher granularity than the coarse-grain announcements of an entire individual file in a torrent as discussed in Section 3.2.5.

### 3.5.2 Tradeoffs of Shifting Intelligence to the Network

nTorrent is an application example, where peers do not explicitly have to track other peers. They just need to know a routing indirection (i.e., forwarding hint) and use it to request data. The NDN network will make sure that data is retrieved from closest source that can offer it (other peer or an in-network cache), deal with cases of flash crowd scenarios through Interest aggregation and in-network caching, as well as take advantage of its stateful nature to hide the dynamic nature of peers and individual peer disconnection from data requesters.

These benefits, of course, come at the cost of requiring the network to do more work and maintain more state on behalf of applications. Moreover, the nTorrent forwarding strategy described in Section 3.2.4 needs to be deployed by network operators at the routers of each individual network (e.g., "`/UCLA`"). Note that this strategy does not track connections to individual peers, but rather offers some general-purpose stateful and adaptive Interest forwarding benefits. Despite that part of the complexity is indeed shifted to the network, we argue that the network is the proper architectural layer that has direct access to underlying connectivity information in order to aid robust and efficient file sharing operations. This argument was verified by our experimental results (Section 3.4), since nTorrent demonstrated an adaptive behavior, outperforming BitTorrent and DHT under a number of different scenarios.

### 3.5.3 Contributions of the NDN Architecture & of Forwarding Hints to the nTorrent Design

The NDN architecture itself offers a data-centric network substrate to nTorrent, which directly uses data names defined by nTorrent for torrent downloading purposes. This simplifies the nTorrent operation, since peers do not have to keep track of other peers as in BitTorrent; they simply express requests for data and the network finds and returns the data. NDN also offers a stateful and adaptive forwarding plane, which makes nTorrent robust in the face of peer disconnections (Section 3.4.2) and flash crowd scenarios (Section 3.4.4), while at the same time enables nTorrent to fetch data in parallel from multiple peers to maximize downloading speeds (Section 3.4.3). Finally, NDN offers data-centric network layer security,

which is the fundamental building block of nTorrent's application security, so that peers can verify the integrity of the downloaded data and decide whether they trust the peer that produced it.

The forwarding hints help nTorrent scale with the number of torrents that are available by adding one level of forwarding indirection to the NDN network. As a result, the FIBs of routers across the global Internet do not need to contain all the available torrent prefixes to forward requests for these torrents; they rather need to contain a smaller set of globally routable prefixes (e.g., "`/UCLA`", "`/NIST`"), under which the torrent data can be found. As we have mentioned in Sections 2.1.3 and 3.3, the problem of routing scalability is not a new problem in network and the proposed approach of forwarding hints is similar in spirit to the map and encap approach to scale IP routing [Dee96]. For example, without the approach of forwarding hints, if there were 10M torrents available all over the world, routers would need to have one FIB entry per torrent, resulting in 10M entries in total, to be able to retrieve data for these torrents. Note that this forwarding state would be needed only for a single NDN application. Maintaining such forwarding state for all NDN applications obviously does not scale. On the other, if forwarding hints are universally used by all NDN applications, a routable set of names will be used as the forwarding state of all routers, therefore routers will not have to maintain application-specific forwarding state. Our experimental evaluation (Section 3.4.5.3) also demonstrated that the use of forwarding hints does not impede the nTorrent performance compared to the approach of routing directly on application names.

# CHAPTER 4

# DAPES: Peer-to-Peer File Sharing in Mobile Ad-hoc Networks

## 4.1 NDN-Native Peer-to-Peer File Sharing in Mobile Ad-hoc Networks

In this section, based on the challenges of peer-to-peer file sharing in IP-based MANETs presented in Section 2.4, we present the benefits that NDN can offer.

By utilizing NDN's name-based data fetching, a node does not need to know where the data producer is, since data can be stored on any node. As an Interest propagates in the network, any node having the corresponding data can respond. The data follows the PIT entry of the intermediate nodes to come back to the consumer. As a result, NDN's stateful forwarding enables data delivery over multiple wireless hops in a highly dynamic environment without running a traditional IP-based routing protocol. Moreover, each NDN node has built-in storage for both pending Interests (the PIT) and Data (the Content Store). Thus, it can carry them around and utilize any ad-hoc connectivity to further forward them when possible. This essentially makes forwarding immune to network partitions, without requiring the existence of end-to-end paths.

In terms of node configuration, the NDN network layer forwards Interests based on semantically meaningful application-defined names. These names identify the data directly and are independent of the location of the node that has produced the data. The only necessary configurations are trust anchors and node certificates, as described in Section 4.3.

Finally, NDN builds data-centric security directly at the network layer of the architecture,

which is universally provided to all each and every communication exchange among MANET nodes. Each NDN data packet is signed, thus receiving nodes can verify the integrity of the received data, as well as decide whether they trust the node that generated this data.

## 4.2 Baseline Scenario

Our scenario assumes a rural area, where residents need to share with other residents information about damaged parts of the infrastructure (e.g., a damaged bridge). This information needs to be resiliently, securely, and efficiently shared with as few transmissions as possible (i.e., minimal overhead and energy consumption), given that the resident devices may have limited battery power (e.g., mobile phones, tablets). Figure 4.1 illustrates such a scenario, where the communicating parties form connected groups that change over time as they move. A resident takes a picture of the damaged bridge (file with name "`bridge-picture`") and, along with information about the bridge location (file with name "`bridge-location`"), packages this data as a *collection of files*. Each file in the collection consists of a number of individual data packets signed by this resident, who acts as the collection producer and starts sharing the file collection data. The data is disseminated through interactions among residents, data repositories ("repos" for short) locally deployed to enhance data availability through collecting as well as serving data from/to residents, and "store-and-forward" data carriers that transfer data from one connected group to another.

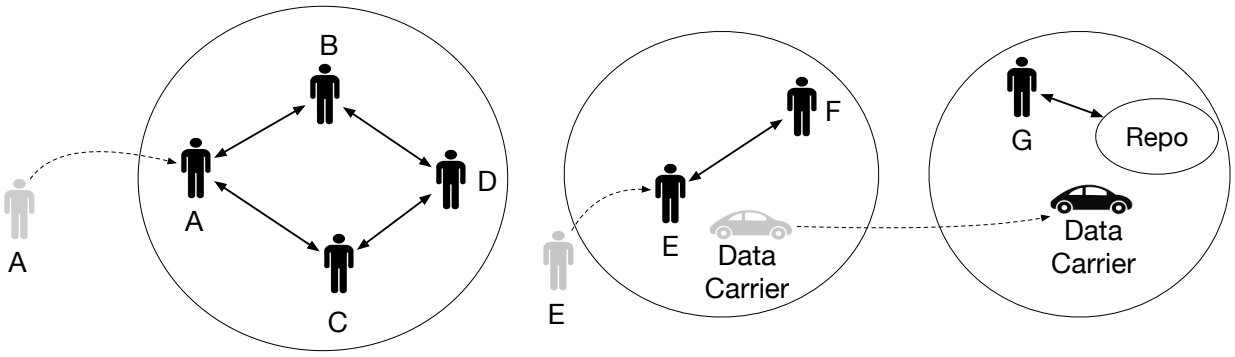



Figure 4.1: Off-the-grid baseline scenario

## 4.3 Design Overview

Our design aims to achieve secure and efficient data sharing among multiple peers under dynamic and adverse network conditions through: (i) a semantically meaningful and hierarchical namespace (Section 4.4.1) utilized directly by the NDN network, (ii) secure initialization of the data sharing process through cryptographically signed metadata (Section 4.4.3), (iii) compact encoding of what collection data peers have (Section 4.4.4), and (iv) mechanisms for efficient data discovery and sharing, and collision avoidance (Sections 4.4.2, 4.4.5, and 4.4.6 respectively).

Given that peers may constantly move, a mechanism is needed to discover when peers are within the communication range of each other and what file collections they have (step 1 in Figure 4.2). Peers need to securely initialize their data sharing process by: (i) authenticating that the file collection producer can be trusted, and (ii) learning the names of the data to request in order to retrieve a file collection and verifying the integrity of the retrieved data packets. To achieve this, the file collection producer generates and signs a metadata file for the collection. When peers discover for the first time a file collection of interest through an encountered peer, they retrieve and authenticate the collection metadata (step 2 in Figure 4.2).

To reduce bandwidth consumption and communication delay, peers exchange compactly encoded information about the data they have, called "data advertisements" (step 3 in Figure 4.2). They prioritize the retrieval of rare data in the context of off-the-grid communication through variations of the basic RPF strategy (step 4 in Figure 4.2), and use a random timer for collection data transmissions to avoid collisions. To ensure that peers are aware of as many of the available packets as possible within their communication range, they make use of a prioritization scheme for data advertisement transmissions. For MAC layer communication, peers use IEEE 802.11 in ad hoc mode under the same SSID and channel number.

**Security Assumptions/Considerations:** We assume that each peer has a pair of public and private keys to sign the packets that it generates. To verify the authenticity of others, including the producer of the file collection, peers need to have a common "local" trust anchor (e.g., among the residents of the rural area) established [ZYZ18, ZZN18]. Based on this common trust anchor, peers verify the metadata signature and decide whether they trust the collection producer. Note that NDN assumes pre-existing trust relations among peers, and under this assumption, NDN provides the means to reflect these relations during communication and data sharing. Related work [SKS14, TYS16] has indicated that in an off-the-grid communication scenario, trust anchors and certificates can be pre-loaded into peer devices.

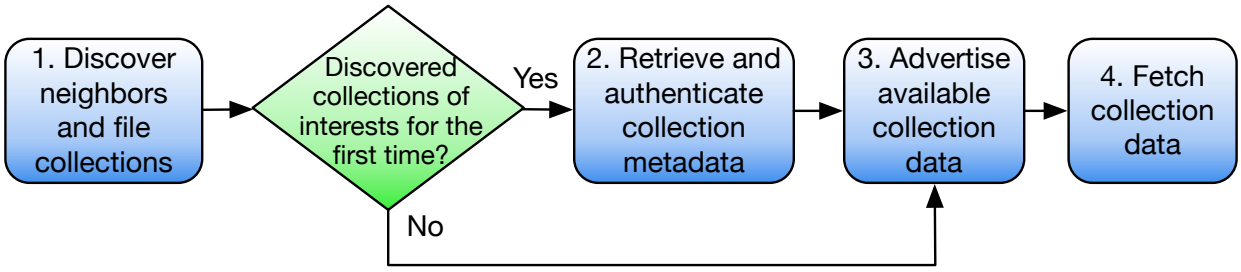



Figure 4.2: DAPES design overview

## 4.4 DAPES Design Components

In this section, we present the DAPES design components.

### 4.4.1 Namespace Design

Our goal is to enable peers to identify specific data of interest in a data collection (e.g., peers might not be interested in all the files in a collection, but only in specific files from the collection). Therefore, we design a semantically meaningful and hierarchical namespace, so that peers can identify the name of a file collection, a specific file in the collection, and a data packet in a file. We identify individual packets through a sequence number, which allows us to compactly encode information about which packets in each

file peers have, as we explain in Section 4.4.4. In our scenario, the collection of files has a name: "`/damaged-bridge-1533783192`", which includes a unix-timestamp that specifies when the collection was generated. The first packet in the first file (picture with name "`bridge-picture`") has a name: "`/damaged-bridge-1533783192/bridge-picture/0`".

### 4.4.2 Peer & File Collection Discovery

Peers need to be aware when others are within their communication range, so that they exchange as much data as possible. To achieve that, each peer periodically broadcasts signaling Interests, called *discovery Interests*. Peers, receiving a discovery Interest, send a *discovery data packet* back that contains the name of the metadata files for the file collections they have[1]. In this way, peers can discover all the file collections that others can offer data for. The discovery namespace consists of the application name and the name component "discovery" ("`/dapes/discovery`").

This mechanism is implemented at the application layer. We aim to run on top of the existing IEEE 802.11 in ad hoc mode, which incorporates a beaconing mechanism for clock synchronization and the discovery of neighbors, however, it does not provide any feedback to the upper layers of the network architecture. In our scenario (Figure 4.1), each peer periodically broadcasts discovery Interests, which helps F and E discover each other and the collections they have after E moves within the communication range of F.

To mitigate potentially higher overheads caused by a high frequency for the periodic transmission of discovery Interests, peers dynamically adjust the frequency for sending discovery Interests. Specifically, peers increase their frequency when they have recently encountered neighboring peers and they reduce their frequency once they are isolated.

[1]The sender of a discovery data packet can learn the name of the metadata files of peers in its neighborhood by sending its own discovery Interest.

### 4.4.3 Metadata For Secure Initialization

When peers discover for the first time a file collection of interest through an encountered peer, they need to securely initialize their data sharing process. To achieve that, peers retrieve and authenticate the metadata file, which consists of one or more NDN data packets generated and signed by the collection producer. The metadata also helps peers to discover the data namespace (name of files and individual packets) and verify the integrity of each data packet in the file collection, without having to verify its signature, which would be computationally more expensive. In the rest of this section, we present alternative metadata encoding formats (Figure 4.3) and how each one achieves the above goals. These encodings involve a trade-off between the size of the metadata and how soon the integrity of each received packet can be verified.

**Packet digest based format:** Based on our hierarchical namespace, all the files in a collection and the packets in an individual file share a common name prefix. As a result, the metadata file can contain a list of "subnames" in the form of "`[packet-index]/[implicit-digest]`" for each individual file. The subname refers to the index of a specific packet in the file followed by the packet's digest. To construct each packet's name, a peer first appends each subname to the name of the corresponding file, and then appends the resulting name to the collection name. Each subname's digest enables peers to verify the integrity of a packet as soon as it is received. This approach can result in large metadata files that need to be segmented into multiple network layer packets.

**Merkle tree based format:** Peers verify the integrity of the packets through a Merkle tree [Mer87], whose root hash is generated by the collection producer and is included in the metadata content. There can also be multiple Merkle trees (e.g., one per individual file in the collection), thus, multiple root hashes can be included in the metadata. The metadata can typically fit into a single network layer packet, but all the packets in a tree need to be retrieved before their integrity can be verified. To construct each packet's name, peers first append the packet index (e.g., for a file with 100 packets, the index of the first packet is 0 and of the last packet is 99) to the name of the corresponding file, and then append the

resulting name to the collection name.

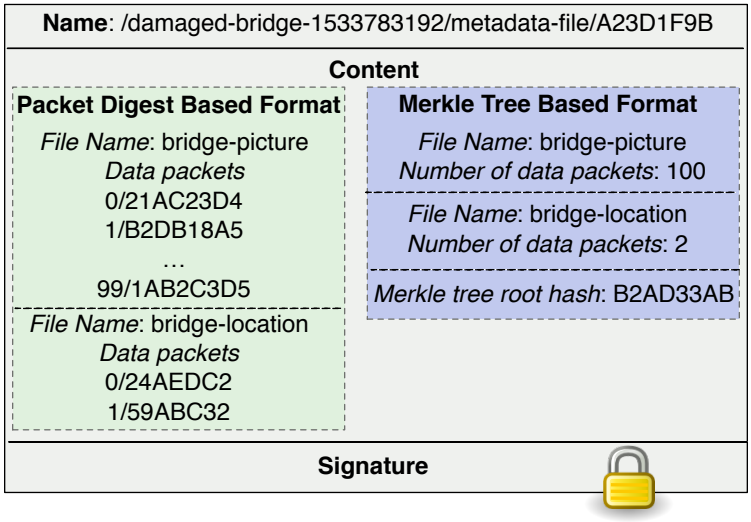



Figure 4.3: Metadata file formats

### 4.4.4 Data Advertisements

Peers advertise what data they have for a collection in a compact and space-efficient way. Given that data is hierarchically and sequentially named, we use a *bitmap*; each bit refers to an individual packet, having a value of 1 if the peer has this packet and 0 if this packet is missing. After a peer downloads the collection metadata, it creates a bitmap of 0s for this collection by ordering the data packets based on the relative position of the files in the metadata and the position of the packets in each file. For the metadata of Figure 4.3, the first bit of the bitmap refers to the first packet of the first file ("`bridge-picture`"), the second bit to the second packet of this file, etc. The first packet of the second file ("`bridge-location`") corresponds to the 101st bit of the bitmap, and the second to its 102nd bit.

After discovering the file collections an encountered peer has, peers send an Interest, called a *bitmap Interest*, for each collection they are interested in. Each such Interest carries the sender's bitmap for the corresponding collection. A peer receiving such an Interest sends its own bitmap back in the content of a *bitmap data packet*. The data advertisement namespace consists of the file collection name, the component "bitmap", and the bitmap of the Interest sender. In our scenario, peer E receives F's discovery data and sends a bitmap

Interest carrying its bitmap. F receives E's bitmap Interest and sends back a bitmap data packet containing its own bitmap.

**Encounters of multiple peers:** When multiple peers meet each other, for example, A moves into the communication range of B, C and D, peers can fetch the bitmap of only some or all the others within their communication range. They may also select to first exchange bitmaps and then start exchanging data or interleave bitmap and data exchanges. These decisions involve a trade-off between the number of bitmaps peers retrieve, which indicates the knowledge they have about the available data (the more knowledge they have, the more efficient the downloading process will be), and how much time they have to download data.

**Analysis:** Let us assume that a peer is connected with others for a time interval $t$ and the transmission delay is $d$. Let $\mathrm{T_{delay}}$ be the average delay for a peer to successfully transmit a bitmap (we further elaborate on $\mathrm{T_{delay}}$ in Section 4.4.6). If peers exchange $b$ bitmaps before they start fetching data, the average time interval they will have for data fetching is:

$$T_{\text{data fetching}} = \begin{cases} t - (T_{\text{delay}} + d) * b, & \text{if } (T_{\text{delay}} + d) * b < t \\ 0, & \text{if } (T_{\text{delay}} + d) * b \geq t \end{cases}$$

This equation shows that peers will have time for data fetching only if their overall encounter lasts more than the time they need for bitmap exchanges.

When peers interleave their bitmap and data exchanges, after they fetch the first bitmap, they have an equal chance of sending a bitmap Interest or an Interest for data until $b$ bitmaps are exchanged. Assuming that $b'$ is the actual number of bitmaps exchanged during an encounter and $b$, where $b' \leq b$ and $0 \leq b' \leq \lfloor \frac{t}{T_{\text{delay}}+d} \rfloor$, the average time interval that peers have for data fetching is:

$$T_{\text{data fetching}} = \begin{cases} t - (T_{\text{delay}} + d) * b', & \text{if } T_{\text{delay}} + d \leq t \\ 0, & \text{if } T_{\text{delay}} + d > t \end{cases}$$

In other words, peers will not have time for data fetching only for very small connections (i.e., they do not have enough time for a single bitmap exchange).

### 4.4.5 Data Fetching Strategy

After exchanging advertisements, peers know what collection data is available around them and proceed to downloading. To ensure that rare packets are retrieved first and are replicated among peers, a strategy based on RPF is used to decide which collection data packets to download. Since we focus on environments with intermittent connectivity, we propose 2 variants of the RPF strategy: (i) RPF across a peer's communication range (local neighborhood), and (ii) RPF based on the history of peer encounters.

**Local neighborhood RPF:** It estimates the rarity of each packet based on how many peers within the local neighborhood do not have this data. When two or more peers exchange their bitmaps, each of them computes the rarity of each packet based on how many of the received bitmaps show a packet as missing. Each peer then creates a list of missing packets; on the top of the list are packets with higher rarity value, which will be requested first. This list is specific to each set of connected peers, and expires after the peers get disconnected, thus no long term state is maintained.

**Encounter-based RPF:** It estimates the rarity of each packet based on the history of encountered peers in the swarm, providing an estimation of how many peers in the swarm do not have this packet. Each peer maintains a list of [peer prefix, bitmap] pairs for a number of peer encounters. Whenever a peer encounters others, it updates the list to reflect the received bitmaps. The rarity of each packet is estimated based on all the bitmaps in the list, and peers request the packets with the highest rarity values first.

**Trade-offs:** The two approaches offer different ways to estimate how rare a packet is. The first one allows peers to fetch data needed by as many neighbors as possible, reducing the overall number of transmissions, without requiring peers to store long term state. The second approach prioritizes data based on peers in the swarm as a whole and requires peers to store and manage local state among multiple encounters.

### 4.4.6 Data Advertisement Transmission Prioritization & Collision Mitigation

**Data advertisement transmission prioritization:** For the transmission of the first bitmap during an encounter, the peer that has most of the data receives priority. This is useful when a peer having few (or no) data encounters a peer that has most (or all) of the packets, so that the latter disseminates as much data as possible to the former. For each subsequent transmission, our prioritization strategy is based on the number of packets each peer has that are missing from all the previously transmitted bitmaps. This scheme[2] maximizes the number of available packets that peers are aware of within their neighborhood.

**Collision mitigation:** Peers can prioritize their bitmap transmissions linearly by dividing a default transmission window by the percent of the packets they have that are missing from previously transmitted bitmaps. This, however, results in frequent collisions when peers have a similar number of packets that are missing from previous bitmaps. To mitigate that, we propose a variant of the Ethernet exponential backoff algorithm, which we call "Priority-based Exponential Backoff Algorithm (PEBA)". PEBA separates peers into groups of transmission slots created through the exponential backoff algorithm, prioritizing peers that have a larger number of missing packets from all the previously transmitted bitmaps. The priority groups and the number of transmission slots are created on a per-encounter basis.

**Example:** In Figure 4.4, we assume that peers have a transmission timer and no collisions have occurred. When no collisions have been detected, peers prioritize their bitmap transmissions by dividing the transmission window by the percent of the packets they have that are missing from previously transmitted bitmaps. Therefore, given that A has most of the data, it schedules its transmission timer to expire before others. When peers receive A's bitmap, they cancel their current transmission and reset their timer based on how many packets they have that were missing from A's bitmap. C's timer expires before others, however, B's timer expires before hearing C's transmission. B and C collide and once they detect

[2]The prioritization scheme applies only to the transmission of bitmaps, since the RPF strategy determines the order to retrieve data. Collisions during the transmission of Interest and data packets determined by RPF are mitigated through the use of a random transmission timer by each peer.

the collision, the backoff algorithm creates 2 slots.

We assume that the slots for the peers that collide are divided into 2 priority groups. Peers that have, at least, half of the missing packets randomly select a slot in the first group, while peers that have less than half of the missing packets randomly select a slot in the second group. In our example, there are 6 packets missing from A's bitmap. C has 3, B has 2, and D has 1 of them. Thus, C will be in the first group, while B and D will be in the second group. C transmits during the first slot, while B and D transmit during the second slot, colliding with each other. In this case, the algorithm creates 4 slots for B and D. There are 3 packets missing from A's and C's bitmaps, therefore B will be in the first group, transmitting during the first or second slot, and D in the second, transmitting during the third or fourth slot.

**Analysis:** Let us assume that there are $L$ transmission slots in total and peers are divided into $k$ priority groups, thus there are $n = \lfloor \frac{L}{k} \rfloor$ slots per group. Peers in the $jth$ group select a random slot $s$, where $j * n \leq s < (j+1) * n$. Zhu et al. [Zhu11] proved that the average backoff number before a successful transmission is $N_{\text{backoff}} = \sum_{i=1}^{\infty} iP_{\text{i}}$, where $P_{\text{i}}$ is the probability that a peer has collided $i$ times before a successful transmission. The average delay for a peer to successfully transmit its bitmap is $T_{\text{delay}} = \frac{L_{\text{average}}-1}{2}\tau$, where $L_{\text{average}} = \frac{n-1}{2}$ is the peer's average contention window size and $\tau$ the slot duration.

## 4.5 Design Extensions

In this section, we extend the DAPES design to achieve: (i) discovery and downloading of multiple file collections, and (ii) communication across multiple hops through one or more intermediate nodes that may or may not understand the DAPES semantics.

### 4.5.1 Discovery of Multiple Collections

In the case that peers would like to advertise a large number of collections, the discovery mechanism presented in Section 4.4.2 might not be enough, since the name of a discovery

| Peer | Bitmap |
|---|---|
| A | 1001011000 |
| B | 0110001000 |
| C | 0000000111 |
| D | 100110000 |

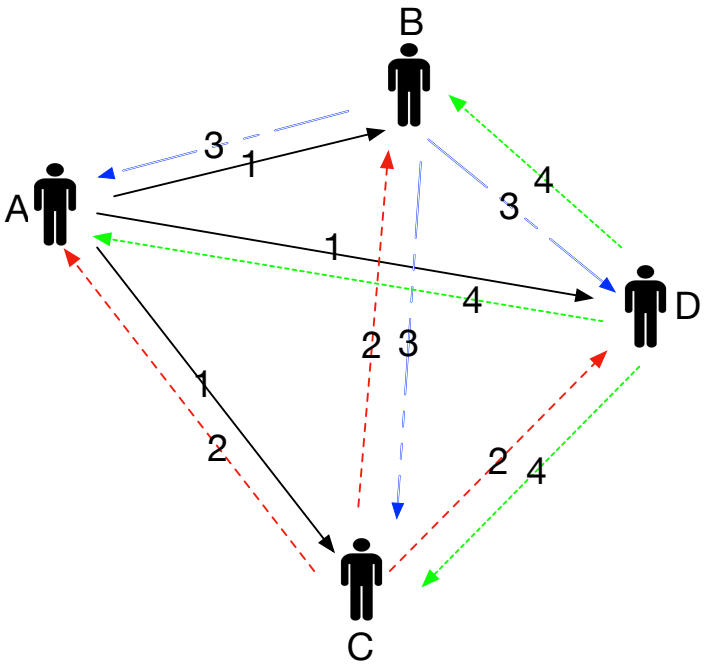



Figure 4.4: Bitmap prioritization & collision mitigation

Interest can carry the name of a limited number of file collections. To deal with this challenge, we propose that peers maintain a metadata file about all the file collections they are aware of (peers may or may not have data about each of these collections), which they send in response to discovery Interests. This metadata file, which is essentially *a collection of file collections*, is named under the "`dapes/discovery`" namespace. The name also carries the prefix of the peer that sends the metadata. It contains the names of the metadata files of file collections that a peer is aware of, along with a flag that indicates whether the peer has data for every given collection (with a value of 1 if the peers has data, 0 otherwise) and the priority of each collection defined by the original producer (Figure 4.5).

Peers can add new file collections to the metadata file as they become aware of them, or delete existing file collections from the metadata when others stop requesting a collection. Moreover, when peers overhear a "collection of file collections" from others, they merge the metadata file they maintain locally with the overheard file. Specifically, they add the name of new file collections found in the overhead metadata file to their own file.

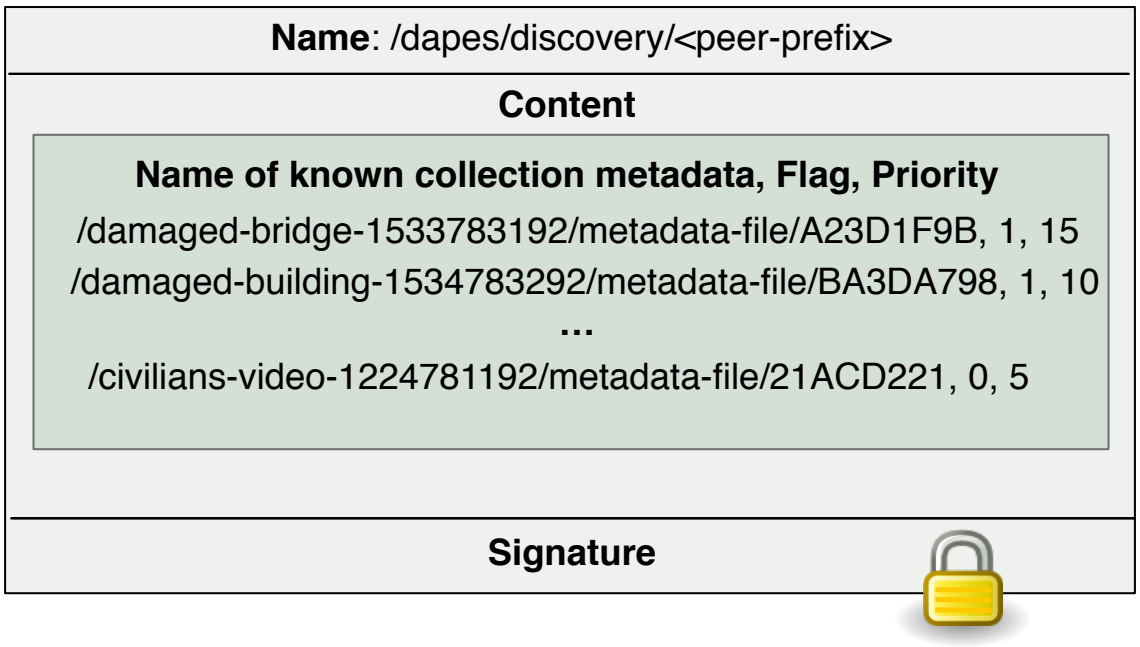



Figure 4.5: An example of a metadata file for a "collection of file collections"

### 4.5.2 Downloading of Multiple Collections

Given that peers can discover multiple file collections, they should also be able to exchange data for different collections. There are two alternative approaches to address this problem depending on our assumptions on whether all collections have the same priority or there can be collections with different priorities:

- Collections with different priorities: If some collections are considered to be more important than others, peers can prioritize them. When peers encounter others and exchange bitmaps, they identify which are the important collections requested within their neighborhood and prioritize their exchange. Specifically, the priority of each collection can be embedded in the bitmap expressed by each peer; peers first exchange bitmaps and then exchange the collection with the highest priority needed by most of the encountered peers.

- Collections with equal priority: In this case, peers exchange bitmaps and then retrieve: (i) the rarest data for the collection requested in the bitmap of most peers, or (ii) the rarest data among all the collections requested in peers's bitmaps.

### 4.5.3 Multi-hop Communication

#### 4.5.3.1 Pure Forwarding Intermediate Nodes

Peers may meet nodes that do not understand the DAPES semantics (e.g., users that have not installed the application on their device), but understand the NDN semantics, which we call "pure forwarding nodes". Given the broadcast nature of the wireless medium, such nodes store and carry received data in their CS, thus satisfying requests with cached data. They also opportunistically forward received Interests based on a probabilistic scheme to: (i) avoid flooding Interests across the network, and (ii) discover data available more than one hop away. Pure forwarders wait for a random amount of time before forwarding an Interest to: (i) avoid collisions with others, and (ii) avoid unnecessary transmissions, since another node within their communication range might respond to the Interest. When they forward an Interest, but do not receive a response, pure forwarders start a suppression timer for the Interest name, not forwarding future Interests with the same name until the timer expires. This timer acts as soft state that determines whether certain data is currently reachable through a pure forwarder.

In Figure 4.6, the dark (A, D, F, H, K) and grey nodes (C, E, G) are interested in different file collections, while node B is a pure forwarder. We assume that A sends an Interest, which can be a discovery or a bitmap Interest, or an Interest for data. Node B receives this Interest, and further forwards it based on some probability. We assume that B forwards this Interest towards direction 1, without receiving a response, thus it starts a suppression timer for the Interest name.

#### 4.5.3.2 Intermediate Nodes Running DAPES

Nodes running DAPES store information about the data their neighbors have and the collections their neighbors are interested in. This helps them make forwarding decisions about the Interests they receive from others. In this way, peers are able to reach others through one or more intermediate peers that are interested in the same or a different file collection.

**Same file collection:** Intermediate peers interested in the same file collection make forwarding decisions based on their knowledge about the available collection data across their neighbors. In Figure 4.6, K is a direct neighbor of A and both are downloading the same file collection. K knows whether there are peers toward direction 3 that download the same collection and what data they have, thus K forwards received Interests from A only when it speculates that they can bring a response back. For example, when A requests data that K does not have, but J does, or when it is beneficial for A to learn J's bitmap rather than K's. In our example, we assume that K speculates that forwarding A's Interest to J will not be bring a response back, therefore, K suppresses the Interest.

**Different file collections:** Intermediate peers interested in a different file collection make forwarding decisions based on messages they overhear about other collections across their neighbors[3]. In Figure 4.6, A and F are 2 hops away, but can reach each other through C, which is downloading a different collection than them. When C receives A's Interest, it can decide whether to forward it. For example, if C has overheard the messages between F and H, it knows that F is interested in the same collection as A, thus forwarding A's Interest toward direction 2 will likely retrieve data. If C has overheard F's bitmap, it knows which packets F has, being able to accurately decide whether to forward A's Interest.

## 4.6 Experimental Evaluation

We performed a simulation study of DAPES to evaluate the impact of different design choices (Section 4.6.3) and compare its performance and overhead with existing IP-based solutions (Section 4.6.4). We also performed a DAPES feasibility study through real-world experiments on the UCLA campus (Section 4.6.5).

[3]If they have no knowledge about the requested data (e.g., have not overheard any related messages recently), they follow the probabilistic scheme of pure forwarders, suppressing Interests that do not bring data back.

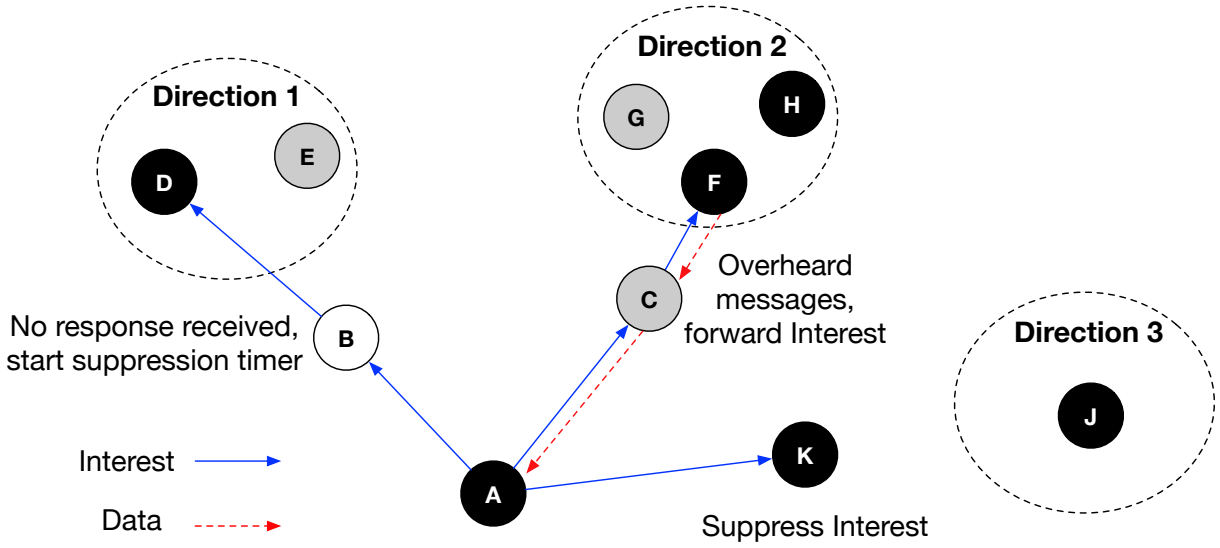



Figure 4.6: DAPES multi-hop communication

### 4.6.1 Prototype Implementation

Our DAPES prototype consists of 5K lines of C++ code and uses the ndn-cxx library [ASY15] to ensure compatibility with NFD. It includes 3 main components: i) a library that provides the fundamental data structures (e.g., metadata file) and abstractions (e.g., RPF strategy) common for peer-to-peer file sharing in both wired and wireless environments, ii) a software adaptation module that adapts the library abstractions to our network environment (e.g., the baseline RPF strategy from the library to provide the different RPF flavors), and iii) an application module that uses abstractions either from the library or the adaptation module to implement the DAPES logic.

### 4.6.2 Experimental Setup

We used a collection of 5 files (each file is 1MB and each data packet is 1KB). We present the 90th percentile of the results collected after 10 trials for simulations and 5 trials for real-world experiments.

#### 4.6.2.1 Simulation Experiments

We ported our DAPES prototype to the ndnSIM simulator [MAZ17]. ndnSIM features software integration with the real-world NDN software prototypes (ndn-cxx and NFD) to offer high fidelity of simulation results.

Our topology (Figure 4.7) consists of 4 stationary and 40 mobile nodes. The mobile nodes randomly choose direction and speed. The speed ranges from 2m/s to 10m/s, representing both humans and vehicles, and the direction from 0 to 2 $\pi$ (loss rate equal to 10%). Nodes communicate through IEEE 802.11b 2.4GHz (data rate of 11Mbps). We perform experiments with varying WiFi ranges, which leads to different sizes of connected groups over time. The 4 stationary nodes and 20 of the mobiles nodes (randomly chosen) download the collection of interest. We present results for the time needed for these 24 nodes to download the file collection (Sections 4.6.3 and 4.6.4).

**NDN-based experiments:** In our topology (Figure 4.7), 10 of the remaining nodes are pure forwarders and the last 10 nodes understand the DAPES semantics and act as intermediate nodes. For neighbor discovery, peers broadcast a discovery Interest every 2sec. Peers use a transmission window of 20ms and select a random value within this window for every transmission other than bitmap transmissions, which are prioritized. Unless otherwise noted, peers use the local neighborhood RPF strategy, interleave data fetching with data advertisements, and fetch advertisements from all the peers within their range. They also communicate over multiple hops (the probability of intermediate nodes to forward an Interest is 20%).

**IP-based experiments:** We compare DAPES with two IP-based peer-to-peer file sharing solutions for MANET; Bithoc [Kri09, Sba08] and Ekta [Puc04]. Bithoc peers perform periodic scoped flooding of "HELLO" messages to discover others and the data they have. They separate others into "close" (at most 2-hops away) and "far" neighbors (more than 2-hops away). Peers follow an RPF strategy to fetch data from close neighbors, while they fetch data not available in their nearby neighborhood from far neighbors. Bithoc uses DSDV [He02] as the underlying routing protocol and TCP over IP for reliability. Ekta offers a Distributed

Hash Table (DHT) substrate for the search of data objects in MANET by integrating the DHT protocol operations with DSR [Joh01] at the network layer. Ekta uses UDP over IP as the transport layer protocol. In our topology (Figure 4.7), 10 of the remaining nodes act as forwarders and the last 10 nodes understand the Bithoc and Ekta semantics. All of these 20 nodes forward received packets based on their routing table.

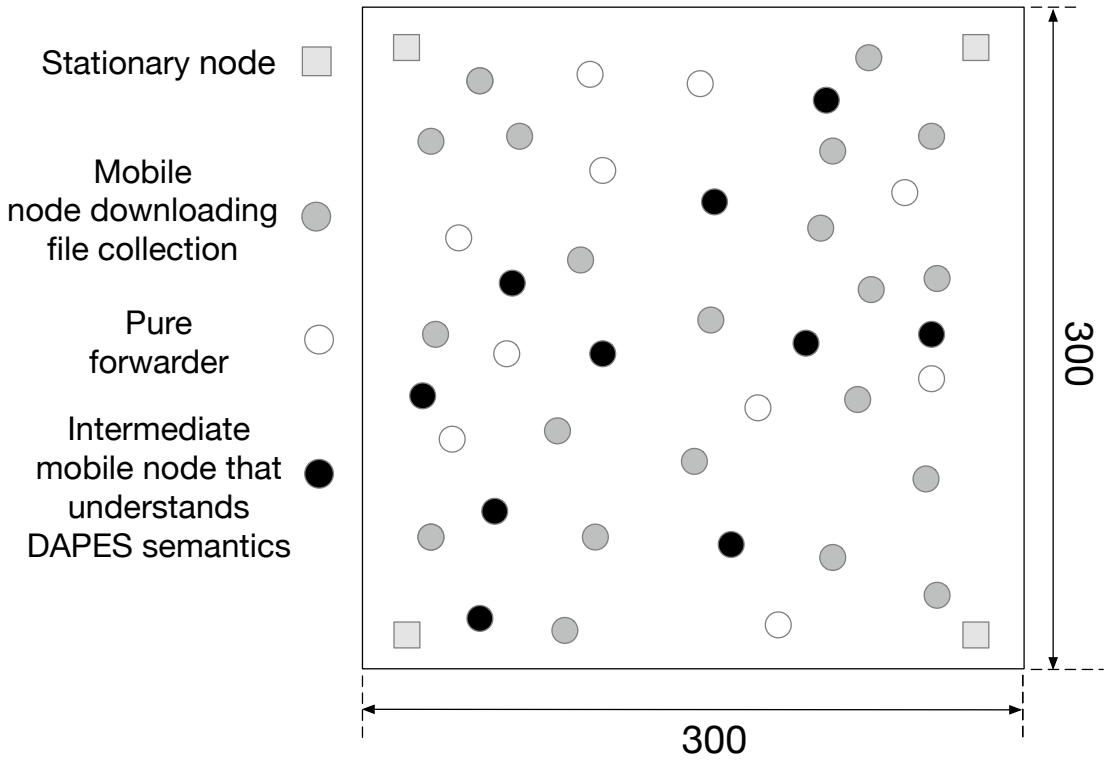


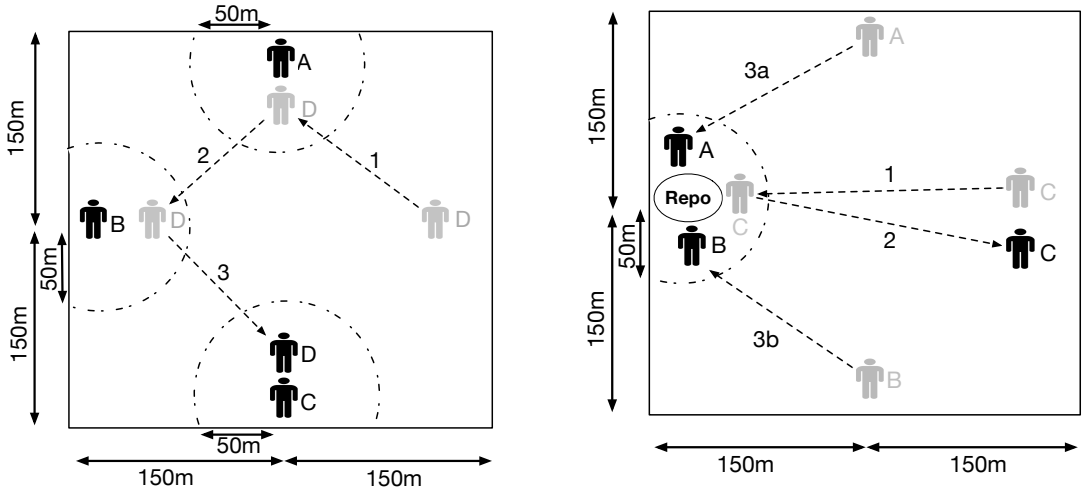

Figure 4.7: Simulation topology snapshot

#### 4.6.2.2 Real-world Experiments

We used 5 MacBooks (macOS 10.13), each equipped with an 1.7GHz Intel i7 processor and 8 GB of memory. We ran NDN on top of IEEE 802.11b 2.4GHz (each MacBook had a WiFi range of about 50m). Peers interleaved data fetching with data advertisements and fetched advertisements from all the entities within their range. Peers also used the RPF strategy across their local neighborhood.

We experimented with 3 different scenarios. In the first one (Figure 4.8a), peer A gen-

erates a file collection. Data carrier D fetches the collection from A and carries it to other network segments, where peers B and C fetch it. In the second one (Figure 4.8b), C generates a collection. The repo downloads the collection from C, while A and B download the collection from the repo. In the third one (Figure 4.8c), A generates a collection that shares with B, C, and D (peers are moving across an area with no infrastructure). To demonstrate how DAPES leverages the broadcast nature of the wireless medium and multi-hop communication, in this scenario, there are moments that all the peers are disconnected and moments that they are within the communication range of each other.




(a) Data sharing through a carrier

(b) Data sharing through a repository

(c) Data sharing among moving nodes

Figure 4.8: Real-world experimental scenarios

### 4.6.3 DAPES Design Trade-offs

**Data Fetching Strategy:** In Figure 4.9a, we present the download time for the encounter-based and the local neighborhood RPF strategies when peers first fetch the bitmap of all the others within their communication range and then download data. The results show that the local neighborhood strategy performs about 12-14% better than the encounter-based. The former strategy focuses on first retrieving the data that is missing from most of the peers within their current neighborhood, while the latter also considers previous encounters among peers that might not be within the communication range of each other anymore. As a result, fewer transmissions take place among the peers when the local neighborhood strategy is used (Figure 4.9b).

The results also show that when peers start their downloading process with a random rather than the same packet of the file collection, they are able to download the collection about 10% faster. Starting with a random packet in the file collection helps every peer retrieve different blocks of data than others, thus disseminating diverse data that others may not have as peers move around. Note that as we increase the WiFi range, which results in a greater number of peers being directly connected to each other, the download time decreases at a slower rate. We conclude that this is due to collisions, given that the number of transmissions for both strategies increases with the WiFi range as shown in Figure 4.9b.

**Collision Mitigation:** Figure 4.9b shows the number of transmissions for both flavors of the RPF strategy with and without PEBA (Section 4.4.6). Without PEBA, both strategies result in a large number of transmissions as the WiFi range increases. This is due to collisions for the bitmap transmissions, since peers prioritize these transmissions by dividing the transmission window by the percent of the packets they have that are missing from previously transmitted bitmaps. Specifically, as the WiFi range increases, more peers are directly connected to each other. This results in more peers that have similar data, thus, scheduling their transmissions very close to each other. On the other hand, PEBA reduces the number of transmissions by 21-26%; PEBA mitigates bitmap transmission collisions through an exponential backoff mechanism and provides prioritization of bitmaps that

increase the available packets that peers are aware of within their neighborhood.

**Data Advertisement Exchange Strategy:** In Figure 4.9c, we present the download time when peers first exchange their bitmaps and then download data for a varying number of exchanged bitmaps. The results show that peers minimize their download time when they have enough knowledge about the available data around them, so that the RPF strategy to make effective decisions (2-3 bitmaps for short WiFi ranges, 4 bitmaps for long WiFi ranges). Peers move out of each other's communication range if they spend too much time exchanging bitmaps before downloading data (e.g., in the illustrated "all bitmaps" case, where peers fetch the bitmap of every other peer within their communication range). This conclusion verifies the analysis of Section 4.4.4.

In Figure 4.9d, we present the download time when peers interleave their bitmap exchanges with the data exchanges. The results show the benefit from fetching data as soon as peers have any knowledge about the available data around them. As they collect more bitmaps, the RPF strategy becomes more effective and peers can download the data faster. This interleaved bitmap and data fetching strategy results in a 12-16% shorter download time than the strategy of fetching bitmaps first and then data.

**Impact of Intermediate Nodes:** In Figure 4.9e, we present the download time for a varying forwarding probability by intermediate nodes, while Figure 4.9f shows the total number of packets transmitted for the file collection retrieval. When pure forwarders and intermediate nodes with no knowledge about the requested data forward 20-60% of the received Interests, the file collection download time decreases by 12-22% compared to the baseline results for the DAPIS single-hop design, while the number of packet transmissions increases by 14-38%. Overall, the results show that it is adequate for these nodes to forward 20-40% of the received Interests. Forwarding a larger amount of Interests results in little performance gain and substantial overhead increase.

**Exchange of Multiple Collections:** For this experiment, we divide the 20 mobile nodes and the 4 stationary nodes mentioned in Section 4.6.2.1 into 4 groups of 6 nodes each. Each group downloads a different file collection. In Figures 4.10a and 4.10b, we present re-

sults for the collection downloading time when all 4 collections have equal and different priorities, where *importance(file-collection1) > importance(file-collection2) > importance(file-collection3) > importance(file-collection4).* As expected, the results show that the more important a collection is, the faster it is disseminated.

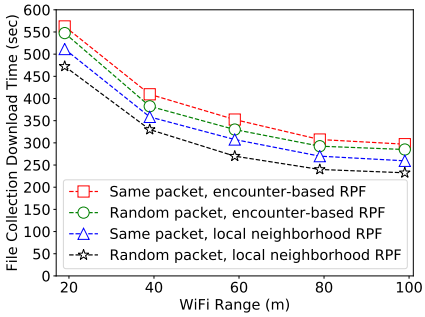
### 4.6.4 Comparison with IP-based Solutions

In Figure 4.11a, we present the download time results, which show that DAPES achieves 15-33% lower download times than Bithoc and Ekta. These solutions identify a specific receiver (based on the receiver's IP address) for each packet they send. Even if multiple peers that miss common data are within the communication range of the sender, a separate packet has to be sent to each one of them. Paths to each peer need to be established first, then discover what data a peer has, and then begin the actual data retrieval. On the other hand, identifying data by name at the network layer enables peers to send a single data packet, which satisfies the pending Interests of multiple other peers in their communication range[4]. Named and secured data decouples data sharing from the location of the entities within the network; data requests can be satisfied with data close to the requester, and intermediate nodes act as data mules to satisfy Interests with cached data overheard from others.

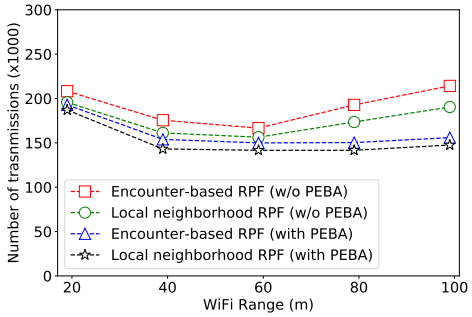
Figure 4.11b presents the number of transmissions for each solution. For DAPES, this includes the discovery Interests and data, bitmap Interest and data, and the Interests/data packets transmitted for the file collection sharing (including the forwarding transmissions by intermediate nodes). For Bithoc, the transmissions include the packets generated by DSDV, the application-layer flooding, and the TCP overhead for the collection retrieval. For Ekta, the transmissions include the packets generated by DSR for route discovery and maintenance, messages among peers on the DHT to find data packets, and the packets needed to retrieve the file collection.

DAPES results in about 50-59% lower overhead than Ekta and 62-71% lower overhead

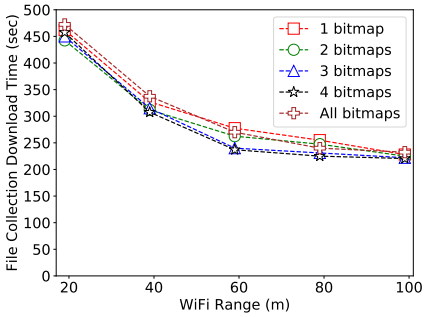
[4]Note that UDP multicast satisfies multiple communicating parties through a single packet transmission, however, it does not receive the benefits of name-based security and caching.



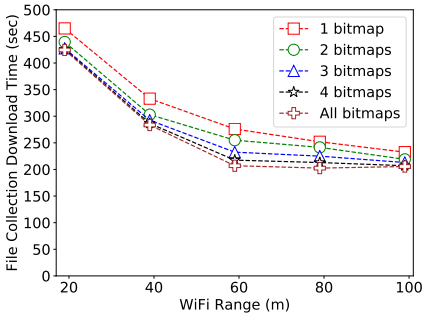

(a) Performance of RPF strategies



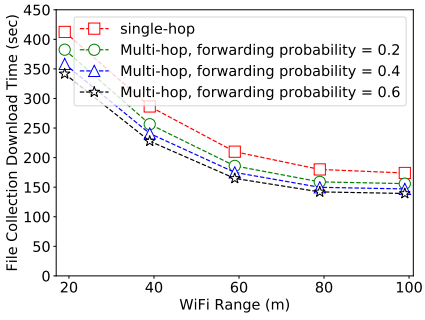

(b) Packet transmissions



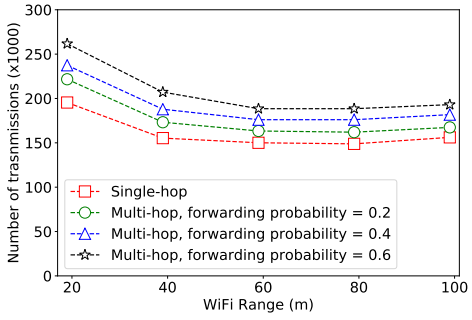

(c) Bitmap exchange before data download




(d) Bitmap exchange during data download

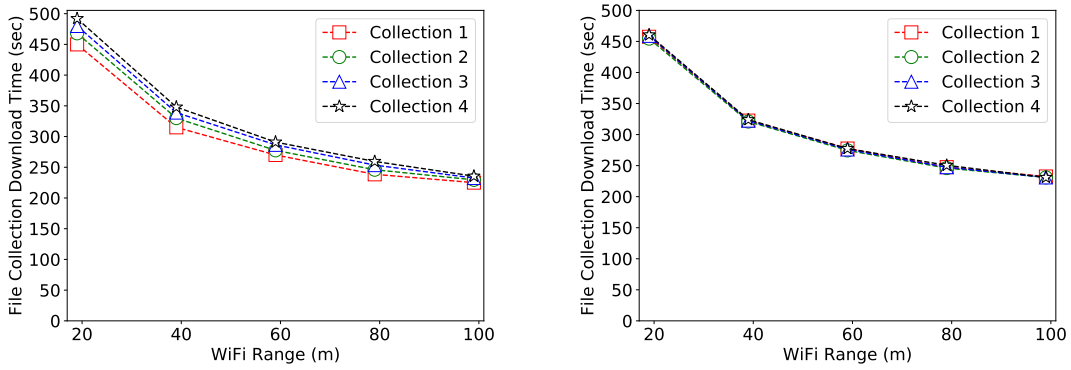



(e) Performance vs forwarding probability




(f) Transmissions vs forwarding probability

Figure 4.9: DAPES design trade-off results




(a) Download time, collections with different priorities

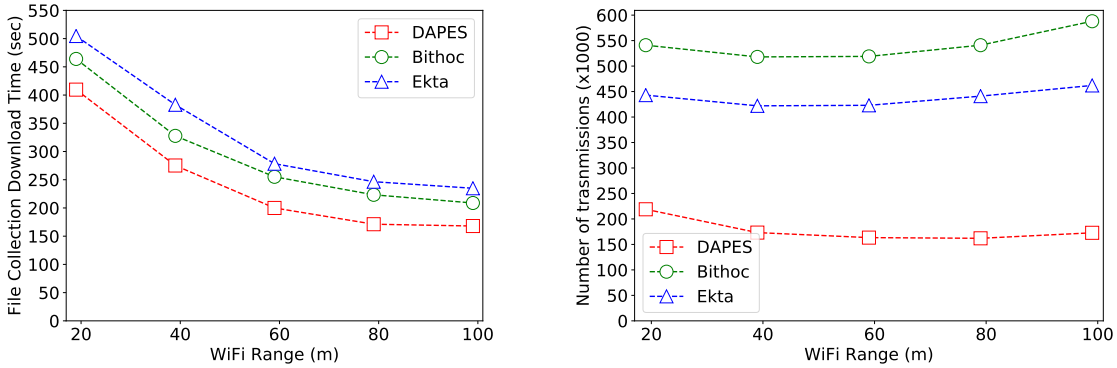
(b) Download time, collections with equal priority

Figure 4.10: Download time, exchange of multiple file collections

than Bithoc. Bithoc requires a proactive routing protocol to maintain routes to each peer, and application-layer messages to discover the data each peer has. Due to intermittent connectivity, established routes break and the TCP performance degrades across multiple wireless hops [HV02]. Ekta is based on a reactive routing protocol, resulting in lower overheads than Bithoc, since routes are maintained on-demand. DAPES features data fetching through semantically meaningful names; intermediate nodes decide whether received Interests should be forwarded or suppr0-essed based on the data names available around them. This results in accurate forwarding decisions (73% of the forwarded Interests were able to successfully bring data back) and low overheads.

### 4.6.5 Real-World Feasibility Study

In Table 4.1, we present the results for each scenario of Figure 4.8. Overall, these results show that NDN's ability to identify data by name directly at the network layer enables DAPES to take full advantage of the broadcast nature of the wireless medium. A single Interest/Data exchange can offer data needed by multiple peers, resulting in lower download times and fewer transmissions. We also conclude that multi-hop communication comes with its own cost in terms of system load, since peers need to store information about the data




(a) Download Time

(b) Packet transmissions

Figure 4.11: Comparison with IP-based solutions

names that are available over multiple wireless hops around them.

| Scenario | Download Time (sec) | Total Number of Transmissions | Memory Overhead (MB) | Context Switches | System Calls | Page Faults |
|---|---|---|---|---|---|---|
| 1 | 454 | 30,841 | 14.75 | 56,413 | 214,313 | 4,742 |
| 2 | 418 | 24,243 | 14.65 | 53,472 | 202,542 | 4,683 |
| 3 | 213 | 16,102 | 18.65 | 46,619 | 186,548 | 4,274 |

Table 4.1: Real-world feasibility study results

In the first scenario, the communication involves only 2 parties (the data carrier and a peer in each connected group), therefore, more time and transmissions are needed for all the entities to download the file collection. In the second scenario, peers A and B fetch the file collection from the repository at the same time. Data requested by either A or B can satisfy both, therefore, the collection can be downloaded faster with fewer transmissions. In the third scenario, peers take advantage of the time that are within the range of each other, and the multi-hop communication to further optimize data sharing. In this scenario, the results also indicate that the system load, in terms of memory consumption and page faults, system calls, and context switches per second, increases. We believe that this is due to the greater amount of multi-hop communication among peers, which requires peers to store state about the data that is available over multiple hops around them.

# CHAPTER 5

# ndnSIM: the NDN-based Simulation Framework

## 5.1 Design Overview

In this section, we present the overall structure of ndnSIM environment (Figure 5.1), consisting of NS3, NFD, and ndn-cxx, as well as an NDN simulation layer, ndnSIM-specific and real-world applications ported to ndnSIM, and a number of plug-and-play simulation scenarios. We also identify a set of features that makes it a useful tool to the research community and present the design workflow by discussing the process of exchanging NDN packets between two simulated nodes.

First, we would like to explain the term "open-source simulation package": the term "open-source" refers to the fact that the ndnSIM codebase is available to the public and users are welcome to download and modify it based on their individual needs. Users are encouraged to participate to the simulator development. The term "simulation package" demonstrates that ndnSIM consists of multiple software components that have been integrated altogether to provide a concrete framework for high-fidelity NDN simulations.

### 5.1.1 ndnSIM Structure

In its core, ndnSIM is based on NS3 simulation framework and leverages it in the following ways:

- To create simulation topologies and specify topology parameters (e.g., link bandwidth, node queue size, link delays).
- To simulate available link-layer protocol models (e.g., point-to-point, wireless, CSMA).

- To simulate the exchange of NDN traffic among the simulated nodes.
- To trace simulation events and (optionally) visualize the simulation execution.

Therefore, ndnSIM simulations can use any of the existing modules, models, NetDevice implementations, and integrated components of NS3.

To realize the core NDN forwarding functions, ndnSIM integrates NFD and ndn-cxx, rewiring key logic elements such as the event processing and network operations to the NS3 specific routines. The result of this integration is that the code used for experiments with NDN forwarding in ndnSIM can be directly used by the real NFD implementation and vice versa. Moreover, ndnSIM allows simulating the real-world NDN applications based on ndn-cxx library (with a few constraints described in Section 5.1.2.2).

On top of the NFD integration, ndnSIM includes an addition NDN simulation layer to streamline creation and execution of simulations and to obtain key metrics. ndnSIM package also offers a collection of tutorial simulation scenarios that provide examples of ndnSIM features.

The following summarizes ndnSIM components and their features:

- **Core (Integration and Models)**: the NDN protocol stack, the realization of NFD's Transport to provide communication on top of NS3 NetDevice and Channel abstractions, the realization of NFD's LinkService to facilitate direct communication between ndnSIM-specific applications and local forwarder instances, and the global routing controller to facilitate static configuration of FIB (based on the Dijkstra's shortest path algorithm).
- **Utilities:** a number of packet tracers to obtain simulation results (link-, network-, and application-level tracing) and topology readers to simplify definition of simulation topologies.
- **Helpers:** a set of helpers to install and configure NDN stack and simulated applications on nodes, to manage (statically or during simulation) FIB, forwarding strategies,

and cache replacement policies, to simplify modifying states (up/down) of the links in the simulated topologies.

### 5.1.2 Applications

ndnSIM can simulate two distinct types of NDN applications: ndnSIM-specific and real-world applications.

#### 5.1.2.1 ndnSIM-Specific Applications

The ndnSIM-specific applications are a convenient way to generate basic Interest/Data packet flows for various network-level evaluations, including behavior of forwarding strategies, cache policies, etc. These applications are realized based on NS3's Application abstraction and include several built-in tracing capabilities, including times to retrieve data.

The built-in ndnSIM-specific applications include:

- *ConsumerCbr*: the consumer application that generates Interest traffic with constant-frequency pattern.
- *ConsumerZipfMandelbrot*: the consumer application that generates Interests with name popularities following the Zipf-Mandelbrot distribution [zip].
- *ConsumerBatches*: the consumer application that generates a specified number of Interest packets at certain points of the simulation execution. It accepts a pattern for Interest generation specifying a set of points of time in the simulation and a number of Interests to be generated at those points.
- *ConsumerWindow*: the consumer application that generates Interests based on a sliding window mechanism.
- *Producer*: the application that responds to each received Interest with a data packet carrying the same name as the Interest and with a specified size.

A few examples of NDN application designs that have been implemented and evaluated as ndnSIM-specific applications are: an application to achieve peer-to-peer file sharing in NDN (nTorrent) [MAY16], a framework that features a number of adaptive multimedia streaming (amus-ndnSIM) [CH], a protocol for the retrieval of the latest real-time data [MGA18], a best effort protocol for link layer reliability [VMA16, AMG15], network management protocols [DOS17a,DOS17b,MAP18], and the design of fuzzy Interest forwarding [CKM17, MCK18].

#### 5.1.2.2 Real-World Applications

The real-world applications are generic applications and libraries that fully leverage the high-level NDN and asynchronous input/output APIs provided by the ndn-cxx library. In other words, these applications can express Interests and dispatch the retrieved Data to the supplied callbacks (as opposed to pre-defined callback for ndnSIM-specific applications), detect Interest timeouts, register prefixes with local NFD, use packet signing and verification APIs. Because ndnSIM integrates with the specially adjusted ndn-cxx library for the simulation environment, such applications can be first developed against the real prototypes and then run inside the simulation environment. Alternatively, the existing applications can be ported to run in ndnSIM to evaluate them at scale.

It is important to note that the existing applications may require several modifications to satisfy requirements imposed by the nature of discrete event simulations. Specifically, an application:

- should not use global variables to define its state, since simulations may need to create multiple instances of the same application that share the same memory;
- should not use disk operations, unless application instances access unique parts of the file system, since in the simulated environment, all application instances access the same local file system;
- should only use ndn-cxx APIs for network-level (ndn-cxx Face), event scheduling (ndn-

cxx Scheduler), and absolute time operations, as otherwise the simulations may incorrectly combine simulated and real-world functions;

- cannot contain any GUI or command-line interactions; and
- the entry point to the application should be configurable (e.g., provided as a C++ class), allowing customized instantiations of the application. This is a generic requirement, which in most cases is satisfied if the application/library is already provisioned for unit-testing.

So far, we and the researchers in the community have adapted and then evaluated with ndnSIM several real-world applications, including:

- *NLSR [HAA13, nls]*: NDN Link State Routing Protocol daemon.
- *ChronoSync [ZA13, chrb]*: one of the earliest distributed protocols for dataset synchronization in NDN.
- *RoundSync [HCS17a, rou]*: a revised design of the dataset synchronization to achieve fast synchronization in face of simultaneous data productions.

Based on our experience in assisting the community with NDN application development, we have noticed that the majority of researchers prefer the simplified ndnSIM-specific application prototypes. While these applications are a good fit to drive the network-level evaluations, they have a limited approximation of NDN application semantics and require substantial effort to implement even simple application behavior (e.g., they need custom timers to handle Interest timeouts, custom callback dispatch mechanism, etc.). Therefore, we would like to encourage the community to experiment with the development of real-world NDN applications that simplify realization of NDN semantics and can be tested both inside the simulator and on the NDN testbed [tes].

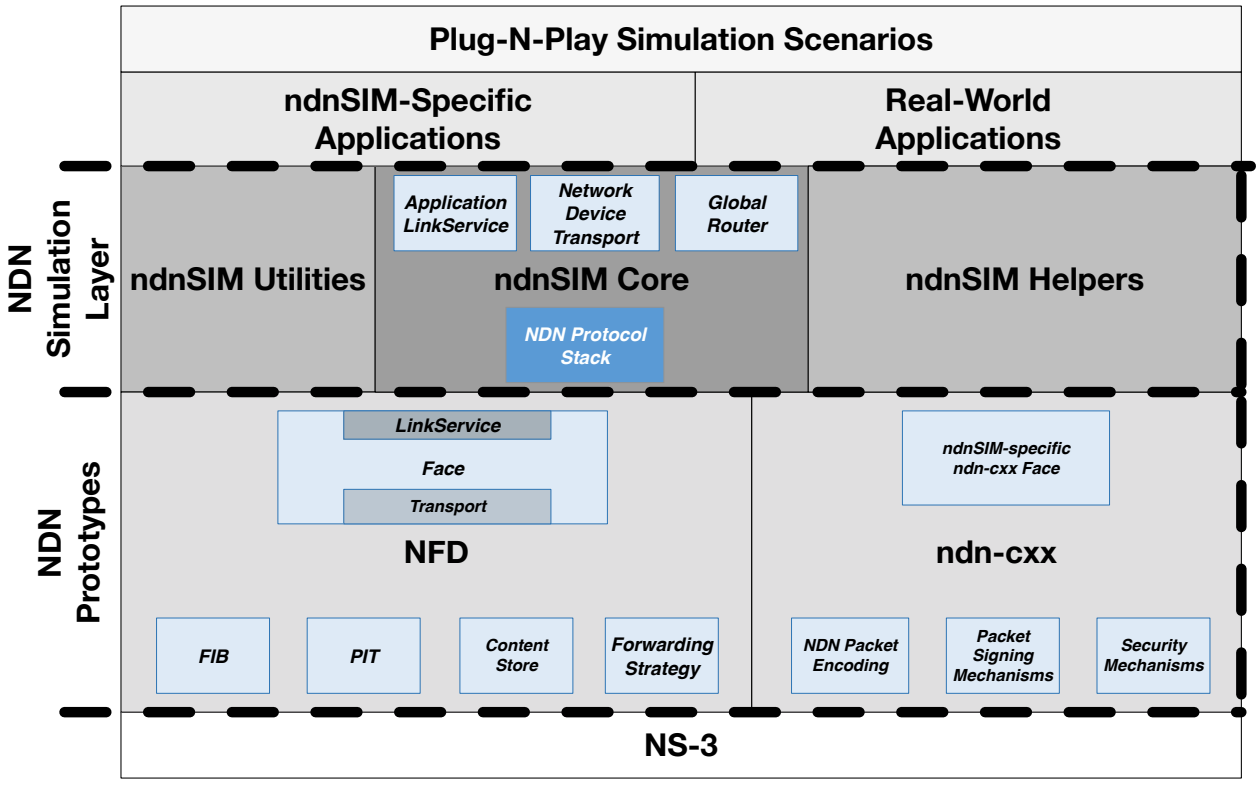



Figure 5.1: Structure of the ndnSIM simulation package

### 5.1.3 NDN Packet Flow in ndnSIM with Integrated ndn-cxx and NFD

The packet flow in ndnSIM involves multiple elements, including NS3's packet, device, and channel abstraction, ndnSIM core, and processing by the integrated NFD with the help of ndn-cxx library. An example of the overall workflow to forward NDN packets between two simulated nodes is illustrated in Figure 5.2.

The whole process is initiated by an application expressing an Interest or publishing a Data packet (after receiving an Interest for it) and includes generation of Interest and Data packets (and appropriately signing them) using abstractions provided by the integrated ndn-cxx library. Using the specialized ndnSIM Face (using AppLinkService) or a customized version of the ndn-cxx Face (using InternalClientTransport and InternalForwarderTransport underlying abstractions), these packets are injected into the NFD instance installed on the corresponding simulated node. After that, NFD will apply the necessary processing logic and will determine how to process the packet, i.e., it will create a PIT entry and forward to

an outgoing Face determined by the strategy for the Interest's namespace, aggregate or drop Interest, satisfy Interest from the internal cache, cache the received Data packet and forward it to the incoming Faces, or drop the Data packet. If a packet is determined to be send out to a Face that corresponds to a simulated link, this packet is getting encoded into NS3's Packet and scheduled for transmission over NS3's NetDevice/Channel using a Face that leverages the specialized NetDeviceTransport abstraction. On the other end, NDN packet is extracted form the NS3 Packet and injected into the NFD instance for further processing.

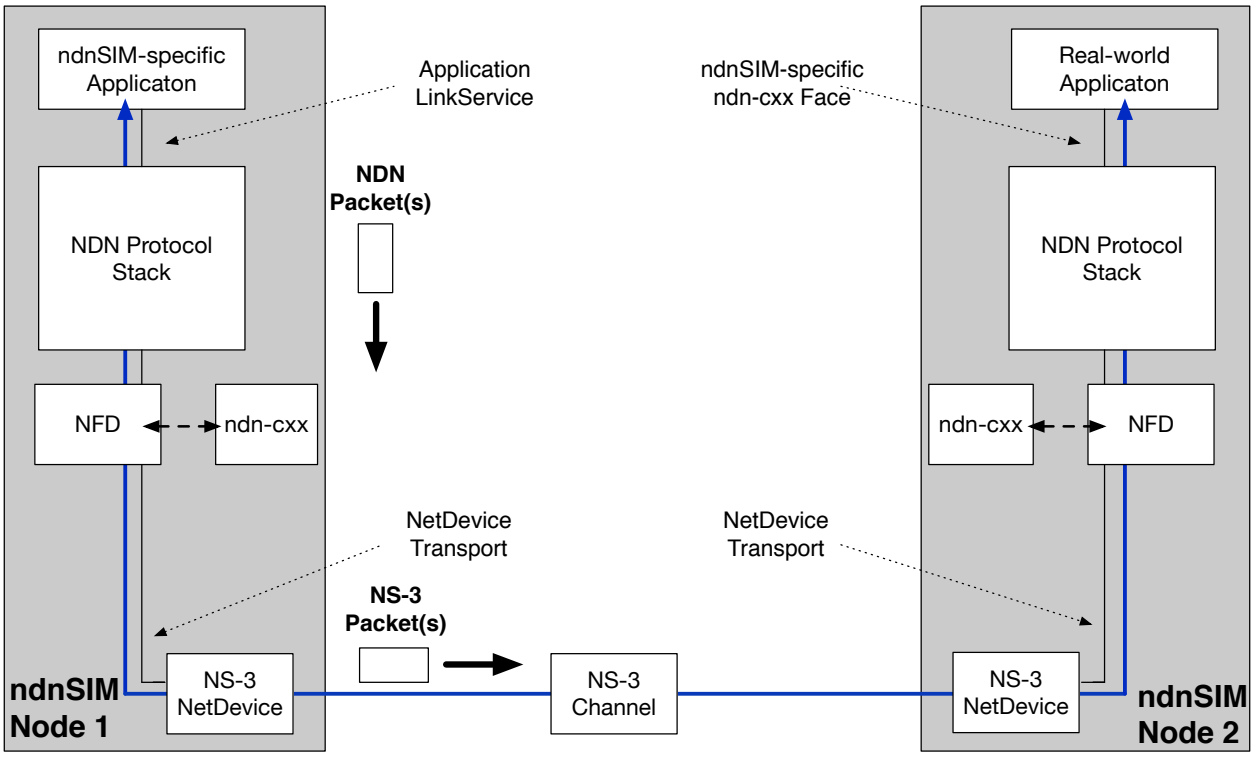



Figure 5.2: NDN packet exchange process between two simulated nodes

## 5.2 Prototype Integration Process

In 2014, the NDN team started development of the new reference implementation of NDN forwarder and supporting library to enable multiple new NDN features (per-namespace strategies, formalized forwarding pipelines, data-centric security abstractions with trust schemas, forwarding hints, etc.) and allowing extensions through the modularity. To ensure that

these advancements can be properly and with high-fidelity evaluated in the simulation environment we made a decision for ndnSIM 2.x to integrate these codebases inside ndnSIM, instead of reimplementing the same features specifically for NS3 (as was the case for ndnSIM 1.x). In the rest of this section, we discuss the challenges and trade-offs of this integration and then present our experimental evaluation of the integration overheads.

### 5.2.1 Integration Challenges

To fit the real prototypes into the NS3 world, we had to:

- redirect the scheduler and logger of NFD and ndn-cxx to use the scheduler and logger of NS3 respectively,
- bind the events in NFD with the tracing facilities of NS3 and extend its data structures and pipelines to add NS-3 tracing points, and
- enable simulation time in NFD by using an abstraction provided by ndn-cxx to convert NS-3 time to NFD system time.

Additional challenge of instantiating real NFD instances on simulated nodes is additional memory and processing requirements for simulation scenarios. To partially mitigate these overheads, we implemented a custom KeyChain to create "mocked" signatures of Data and Interest packets without incurring cost of cryptographic operations and enabled facilities to opt-out of some of the NFD components if they are not needed in specific simulations.

### 5.2.2 Integration Trade-offs

The integration of NFD and ndn-cxx into ndnSIM faced a number of trade-offs. The major change of the packet processing flow broke compatibility with the previous versions of ndnSIM. Because of the tight integration with the real NDN library that assumes properly allocated memory blocks for NDN packets, ndnSIM 2.x can no longer take advantage of the virtual payload abstraction, increasing memory requirements for packet processing and

storage. In addition, the memory overhead takes additional increase because the integrated prototype forwarder realized fully featured data structures of PIT, FIB, CS, and various management structures.

At the same time, ndnSIM is now fully up-to-date with the latest advancements of NDN architecture, allowing flexible two-way experimentation and evaluation—i.e., the prototyped code can be simulated in ndnSIM and simulated code can be evaluated in real (or emulated) environments. Moreover, the integration eliminated the overhead of maintaining and synchronizing two independent codebases and united the research community. As anecdotal evidence of the latter, many questions on ndnSIM mailing list are getting answered by developers and users of NFD, some resulting in discovery of issues and requests of new features for the reference implementations.

### 5.2.3 Integration Overhead Evaluation

To quantify effects of the major re-design of ndnSIM 2.x compared to ndnSIM 1.x, we evaluated several basic use-cases of the simulator: cache replacement policies, forwarding strategies, and applications. We used a simple topology consisting of two connected nodes and measure the system execution time and average memory requirements for CS and PIT. Unless otherwise stated, each node in our evaluations had installed NDN protocol stack (an instance of NFD and the specialized Faces to communicate with NDN applications and remote simulation nodes), one node with a ConsumerCbr application instance generating 1,000 Interests/sec, and one with a Producer instance responding to the Interests. The experiments were performed on an Intel Core i7 processor (2.4 GHz) with 8 GBytes of memory machine, each experiment simulating 30 seconds of packet exchanges. We repeated each simulation ten times and we report on the minimum, maximum, and average values of all the runs.

#### 5.2.3.1 Cache Replacement Policy Development Overhead

ndnSIM 2.x uses the CS implementation of NFD,[1] therefore, to create a new cache replacement policy, users need extend NFD's Policy class to implement new callbacks that are invoked when a new data packet is inserted to the CS, an existing data packet is deleted from the CS, and a data packet is about to be returned after a lookup match.

To develop a custom cache replacement policy, the API provided by NFD can be modified more easily than implementing a new C++ template class required in ndnSIM 1.x, especially for users not so experienced with software development in C++. On the other hand, NFD' cache policies framework incurs additional overhead for virtual function dispatch, resulting in a small processin penalty.

Table 5.1 highlights the memory overhead per CS and PIT entry, and the system execution time with the Least Recently Used (LRU) and First In First Out (FIFO) replacement policies for ndnSIM 1.x and ndnSIM 2.x. The CS has a capacity of 100,000 entries, enough to hold all Data packets during each simulation run, allowing us to measure the memory overhead as the CS size grows. As expected from the loss of virtual payload capability, we observe that the memory overhead per CS entry in ndnSIM 2.x is higher than of ndnSIM 1.x. At the same time, the observed seven-fold average increase is larger than what we expected and we are currently investigating the underlying reasons for this change, including effects of the memory allocation for C++ STL data structures, memory fragmentation, and potential memory leaks in the prototype implementation. The system execution time for ndnSIM 2.x is as expected longer, as it runs the real NFD prototype code that performs more sophisticated processing than the simplified packet forwarding logic in the original ndnSIM 1.x.

#### 5.2.3.2 Forwarding Strategy Development Overhead

In ndnSIM 1.x, the forwarding plane used a single forwarding strategy chosen for the whole duration of the simulation. Among available strategies were Multicast (a.k.a. Broadcast),

[1]Version 2.x of ndnSIM includes support for the ndnSIM 1.x version of cache policies, which will be phase out in future releases

Table 5.1: Cache Replacement Policy Development Overhead between ndnSIM 1.x & ndnSIM 2.x

| | **ndnSIM 1.x (min/max/avg)** | | **ndnSIM 2.x (min/max/avg)** | |
|---|---|---|---|---|
| Cache Replacement Policy | LRU | FIFO | LRU | FIFO |
| PIT Entry Overhead (Kilobytes) | 5.22/5.33/5.29 | 5.20/5.31/5.24 | 37.82/40.21/39.23 | 38.04/40.05/39.12 |
| CS Entry Overhead (Kilobytes) | 0.75/0.79/0.77 | 0.74/0.77/0.75 | 5.41/6.92/6.24 | 5.41/7.38/6.91 |
| System Time (s) | 12.84/14.13/13.56 | 12.49/14.71/13.92 | 23.56/24.93/24.03 | 27.57/29.11/28.53 |

BestRoute, and adaptive "Green-Yellow-Red" variants of both.[2] Among the important facilities that can be leveraged by the strategy (not yet available in NFD and ndnSIM 2.x) was per-face and per-FIB entry rate limits on number of Interests. In ndnSIM 2.x, a forwarding strategy is implemented as a part of the forwarding plane of NFD and can be selected to activate in a specific namespace. To add a new forwarding strategy, users need to extend the Strategy class of NFD and implement certain callbacks that will be invoked when an Interest is received, before an Interest is satisfied by a data packet, and when a NACK is received.

The logic of adding a new forwarding strategy in ndnSIM 1.x and 2.x is similar, however, the APIs provided by NFD are more specialized to make determination where/whether to forward the Interest, while ndnSIM 1.x provided more generic APIs for strategies to control all aspects of all type of NDN packet forwarding. In other words, NFD clearly separates the generic logic of Interest (duplicate suppression, aggregation, etc), Data, and Nack packet processing (so called "pipelines" in [AS15]) and customizable decision (and feedback) to forward Interests, compared to ndnSIM 1.x that combined all of these under a single umbrella of extensible forwarding strategy API. Table 5.2 shows the memory overhead per CS and PIT entry and the system execution time in the case of the BestRoute and Multicast forwarding strategies for ndnSIM 1.x (with Green-Yellow-Red scheme) and ndnSIM 2.x (non-adaptive variants). The results are similar to those presented in section 5.2.3.1 with similar conclusions and future ares of improvements.

[2]Additional strategies are available as part of car-to-car [WAK12], SAF [PRH17], and several other research efforts.

Table 5.2: Forwarding Strategy Development Overhead between ndnSIM 1.x & ndnSIM 2.x

| | **ndnSIM 1.x (min/max/avg)** | | **ndnSIM 2.x (min/max/avg)** | |
|---|---|---|---|---|
| Forwarding Strategy | Best Route | Multicast | Best Route | Multicast |
| PIT Entry Overhead (Kilobytes) | 5.23/5.39/5.30 | 5.22/5.32/5.28 | 37.73/40.12/39.35 | 38.81/40.21/39.51 |
| CS Entry Overhead (Kilobytes) | 0.74/0.77/0.76 | 0.73/0.77/0.75 | 6.34/6.45/6.40 | 6.38/6.41/6.39 |
| System Time (s) | 12.87/14.58/13.71 | 13.04/14.72/14.09 | 22.67/23.98/23.12 | 21.72/22.x9/22.01 |

#### 5.2.3.3 Application Development Overhead

ndnSIM 2.x enables applications to communicate directly with the local NFD instance, therefore it supports simulations of both ndnSIM-specific and real-world applications. In ndnSIM 1.x, an application communicates with a simulated forwarding plane, therefore, it can only simulate ndnSIM-specific applications but cannot run real-world applications.

To make a fair comparison, Table 5.3 presents the memory overhead per CS and PIT entry and the system execution time for ndnSIM 1.x and ndnSIM 2.x when running two ndnSIM-specific applications: 1) the ConsumerCbr application with a constant rate of 2000 Interests/second, and 2) the ConsumerZipfMandelbrot application generating 1000 Interests/second, where the names of the generated Interests are based on an NS3 ZipfRandomVariable instance with $\alpha$ = 1.x and N = 100.

Because of the methodology of our experiment (we are measuring the overall memory use that includes tge overhead of all other auxiliary data structures), the overall overhead of ConsumerZipfMandelbrot application experiment is higher for both ndnSIM 1.x and 2.x that that of ConsumerCbr because of the smaller number of generated of Interest and Data packets. Beyond that, the overhead for memory use and processing time follows the same pattern as reported in previous sections, with higher numbers for ndnSIM 2.x than in ndnSIM 1.x because of the additional requirements of the fully featured NFD implementation.

Table 5.3: Application Development Overhead between ndnSIM 1.x & ndnSIM 2.x

| | ndnSIM 1.x (min/max/avg) | | ndnSIM 2.x (min/max/avg) | |
|---|---|---|---|---|
| Consumer Application | ConsumerCbr | ConsumerZipfMandelbrot | ConsumerCbr | ConsumerZipfMandelbrot |
| PIT Entry Overhead (Kilobytes) | 5.13/5.24/5.18 | 8.08/9.45/9.03 | 38.02/40.22/39.54 | 287.69/299.167/294.21 |
| CS Entry Overhead (Kilobytes) | 0.73/0.76/0.75 | 2.59/3.64/3.06 | 6.17/6.42/6.31 | 82.83/84.08/83.54 |
| System Time (s) | 25.22/29.69/27.36 | 3.44/3.72/3.51 | 45.65/48.13/46.22 | 2.58/2.87/2.71 |

## 5.3 Software Development, Governance Model & Community Adoption

In this section, we present statistics from our GitHub repository [ndna] and our mailing list [ndnb] along with the the number of technical report citations (according to Google Scholar) to measure the ndnSIM software development effort and adoption in terms of community growth respectively. We also present the open-source governance model and stakeholders of ndnSIM.

### 5.3.1 Software Development Effort

In Figure 5.4, we present the number of commits, lines of code added, and deleted and total contributors per year from our GitHub repository [ndna].

During the first 3 years of development (before the integration with the prototypes), we had to maintain simulation-specific software, which required a lot of effort in terms of coding. In 2014, NFD and ndn-cxx were initially sub-folders of the simulator codebase (their commit history was added to the commit history of ndnSIM, resulting in the corresponding spike in Figure 5.4). Starting from 2015, to allow easier integration of every new release with the simulator, we decided to maintain them as separate GitHub submodules (a number code lines were deleted), therefore, the coding effort itself became less demanding. However, the cooperative software design effort with the rest of the NDN team significantly increased to make sure that the features developed for NFD and ndn-cxx are compatible with ndnSIM and NS-3.

The NDN Team follows the full cycle of software development (issue tracking, code review, unit-testing, etc.) used in the software development industry. The code review process introduced for ndnSIM in 2015 to ensure that each commit is comprehensive and well-designed before it is pushed to our GitHub repository.

We should note that we receive a limited number of GitHub pull requests and issues, therefore, these metrics are not representative of the ndnSIM software development effort. The majority of our users submit their questions and code related issues on our mailing list, which has created a user community at large, where people help out with each other's questions. Users also typically fork the official ndnSIM GitHub repository and do their development on their personal repositories, while they share code patches and extensions through the mailing list and submit their code to our code review system to get it merged to our official GitHub repository.

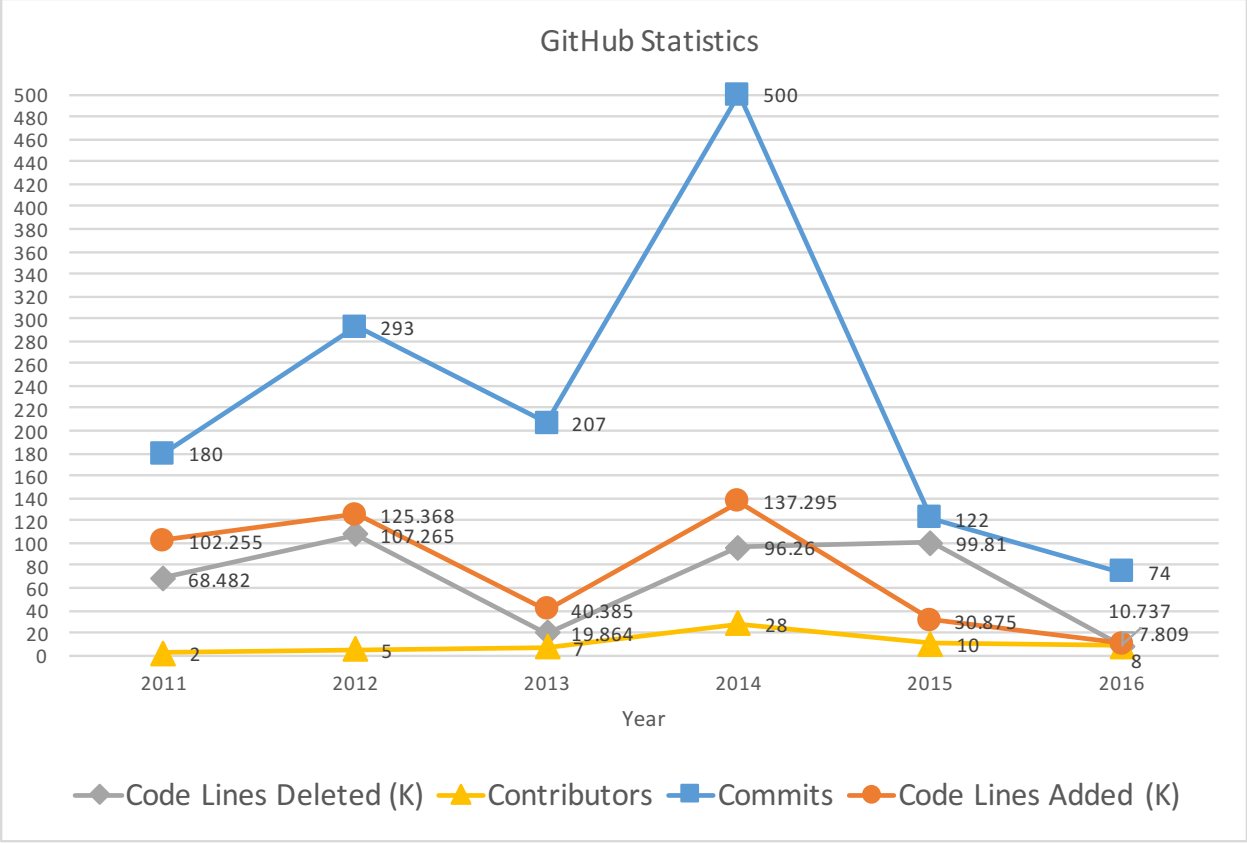



Figure 5.3: Statistics from the ndnSIM GitHub repository

### 5.3.2 Open-Source Governance Model & Stakeholders

ndnSIM is an open-source project, where every researcher can commit their code after going through the code review process (the committers do not need to be NDN Team members). As shown in Figure 5.4, a number of committers external to the NDN Team have contributed to the development of the simulator over the years.

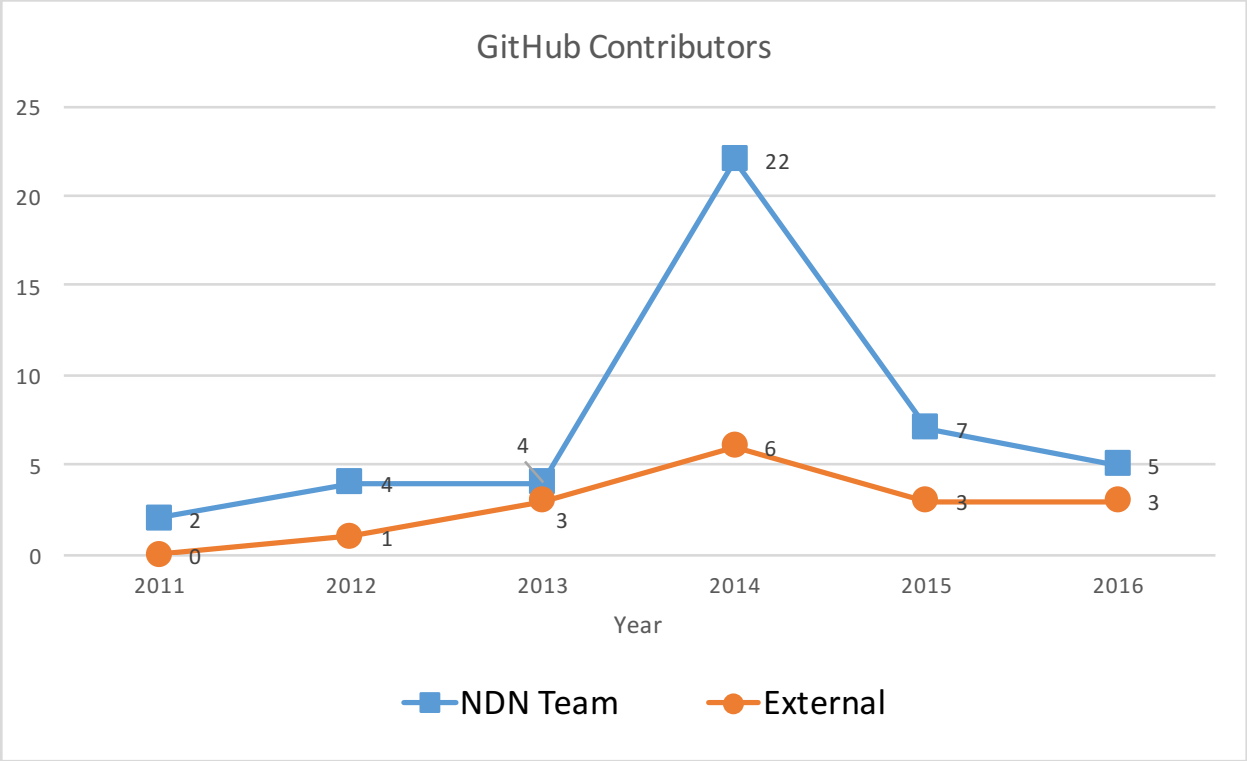



Figure 5.4: ndnSIM internal (NDN team) and external contributors

In Figure 5.5, we present the ndnSIM stakeholders, their relationship and the facilitated development workflow. At the bottom left, we have the users that can: 1) directly submit feature and bug reports to our Redmine issue tracking system, 2) submit their code for review to our Gerrit code review system, and 3) send an email to the mailing list with development related questions, feature requests and bug reports.

At the bottom right, we have the ndnSIM team, which is a part of the overall NDN project team. The ndnSIM team works closely with the NDN protocol architects and the NFD team to participate to the protocol design effort and ensure that the software changes

done in NFD are compatible with ndnSIM. The ndnSIM team with the participation and help of the entire NDN team responds to the user emails received on the list, reviews the already submitted and creates new issues on Redmine, reviews already submitted (either by users or members of the NDN and ndnSIM team itself), and submits new code patches (commits) to Gerrit.

Once a commit is submitted to Gerrit, it is automatically submitted to continuous integration system powered by Jenkins-CI, where it is compiled and tested on a number of different operating systems including several Ubuntu and macOS distributions. Once the commit is verified by Jenkins-CI and the code review process is complete, it is merged to the ndnSIM official GitHub repository.

We should note that users can also participate in discussion related to the specific Redmine issues (in Redmine and the mailing list) and Gerrit code review process, ensuing critical bugs fixed and important features are implemented in a timely manner.

### 5.3.3 Community Growth

The ndnSIM community has grown from a few dozens to some hundreds of members over the last five years. At the time of this writing, the ndnSIM mailing list has approximately 700 subscribers and the technical reports have been cited over 700 times in total. As it is shown in Figure 5.6 and Figure 5.7, the mailing list becomes more active every year (steady increase in the number of threads and individual emails on the mailing list over the last couple of years), while ndnSIM is used and cited by more researchers.

We would like to thank all the members of the community for their help and feedback and, especially, our much-appreciated contributors; Jiangzhe Wang, Cheng Yi, Saeid Montazeri, Xiaoke Jiang, Saran Tarnoi, Hovaidi Ardestani Mohammad, Michael Sweatt, Wentao Shang, Christian Kreuzberger, Yuanzhi Gao, and Mohammad Sabet, Junxiao Shi, Susmit Shannigrahi, John Baugh, Ashlesh Gawande, and many others who have reported bugs, submitted patches, and helped the ndnSIM users on the mailing list by responding to their questions.

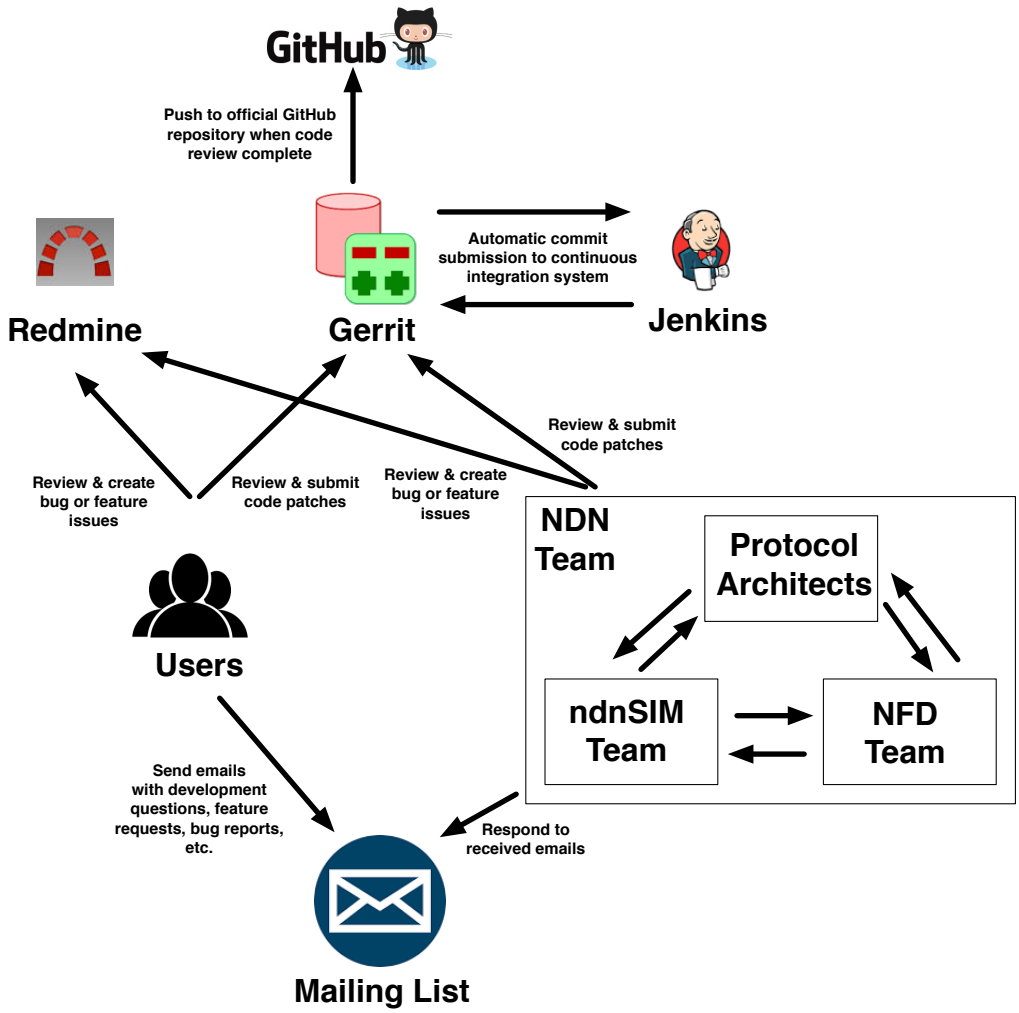



Figure 5.5: Stakeholders, their relationship and the development workflow facilitated by ndnSIM

## 5.4 Lessons Learned

Developing an open-source simulation platform used by a growing and active user community is a rewarding process that has helped to us learn a number of lessons. In this section, we would like to share those lessons with the research community.

**A well-designed simulation platform facilitates the protocol design effort.** It helps researchers understand the architectural trade-offs by enabling large scale experimentation; some design deficiencies come up only after intensive experimentation. A recent example is ChronoSync [chra], where a design bug that could lead to large delays in the case of multiple simultaneous data generations was discovered and fixed only after large scale

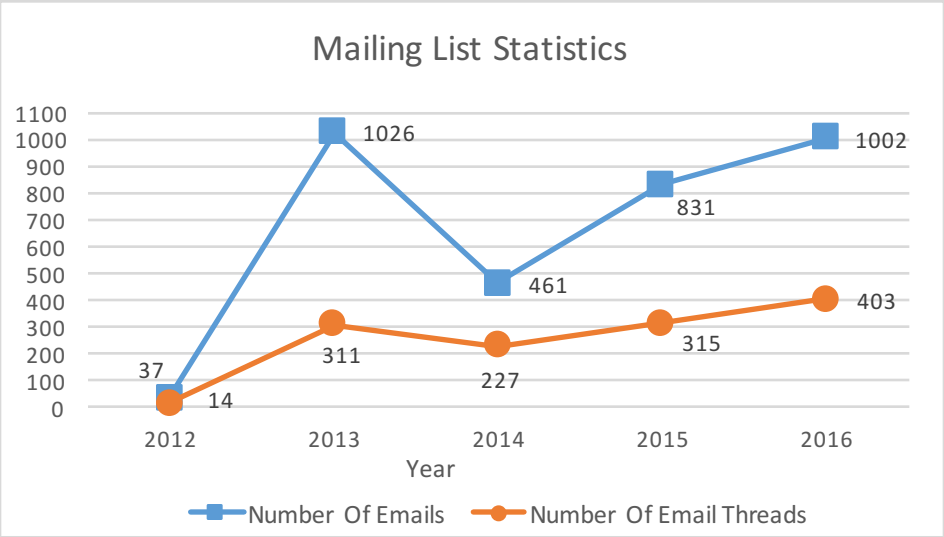



Figure 5.6: Statistics from the ndnSIM mailing list

simulations.

**Researchers should be able to reproduce each other's experiments.** Reproducibility is one of the most important aspect of experiment-based research. Because of the continuous evolution of NDN, the reference implementation, and ndnSIM, it is critical to capture the specific version of ndnSIM used for simulations. To promote such recording and ensure that simulation results can be easily reproduced, we created a specialized template to simplify managing simulation scenarios and ensuring future-proof ability to re-run experiments. In addition to that, we are planning to setup a database to collect all the simulation scenarios used in scientific papers, so that our users have direct access to each other's experiments.

**An open communication channel with the user community is crucial for an open-source project.** A number of times, users have helped each other by responding to questions on the mailing list, but also contributed to the actual software development by implementing specific features.

We have established this channel in two ways:

- By extensively documenting ndnSIM and providing a large set of basic simulation scenario examples on ndnSIM website [ndnc], encouraging user participation and helping getting

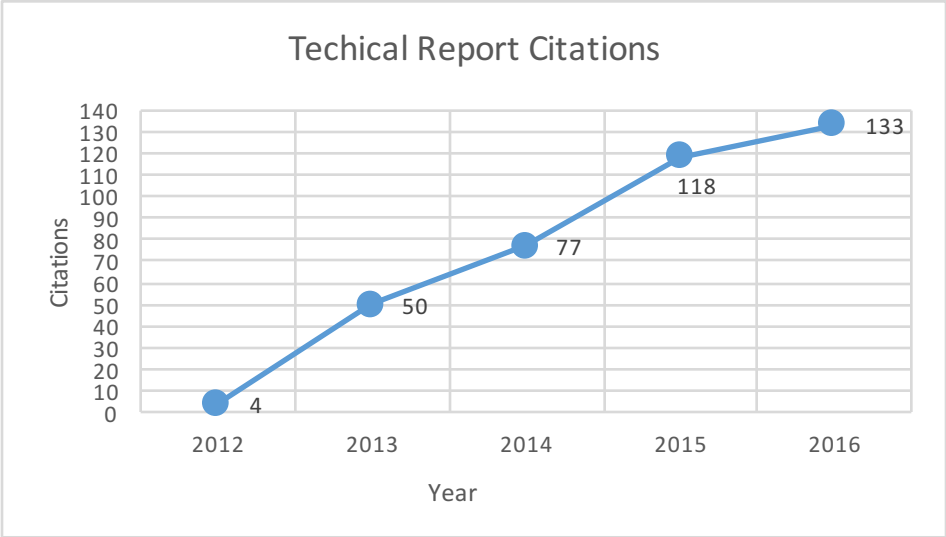



Figure 5.7: Number of citations of ndnSIM technical reports

familiar with the codebase.

- By maintaining a mailing list to allow further collaboration and assistance to questions that have not been addressed in the documentation. The mailing list has helped us further improve our documentation by identifying and addressing a number of frequently asked user questions that fall into one of the following categories: 1) understanding of the simulation outcomes related to packet tracing and simulation execution visualization, 2) experimentation with the NDN architectural parameters, and 3) software development questions, as users are required to have a good understanding of C++ and C++11 for NFD and ndn-cxx extensions.

**Developing an open-source software project is an iterative process.** The developed features may need to be redesigned, refactored, or extended based on the feedback from the users. For instance, right after the NFD integration, we received a number of emails on the mailing list (implicitly) requesting a later introduced API to optionally disable some of the NFD features not needed for basic scenarios. When the support for NACKs was added to NFD, a number of users requested this feature in the simulator, which required the following refactoring of ndnSIM internals with adjustment of several trade-offs (limiting ability to optimize memory overhead).

**The simulator and the prototypes facilitate and influence each other's development.** When the NDN team started working on ndn-cxx and NFD, parts of the ndnSIM codebase were used to facilitate their development. Eventually, when the prototypes were developed, they were integrated in ndnSIM.

Since the beginning of this integration, we have been working closely with the NFD Team to make the forwarder more modular and compatible with the simulator. An outcome of this collaboration is the higher level of modularity of NFD and ndn-cxx with additional parts that are conditionally compiled. Initially, we had to manually remove parts of the NFD and ndn-cxx codebase not compatible with the simulation environment , including implementations of TCP/UDP channels, support for Unix sockets, logging, etc. The latest version of NFD and ndn-cxx is integrated within ndnSIM with minimal changes.

## 5.5 Current ndnSIM Limitations

ndnSIM is currently supported on Linux and Mac OS platforms, but unavailable on Windows (NS3 in general has limited support on Windows platform). It also does not support connecting the simulation network with an NFD, ndn-cxx, or an application instance running on an external host.

As stated in section 5.1.2, real-world applications need to be modified in certain ways in order to run in ndnSIM. The memory requirements can become a limiting factor if one runs ndnSIM on devices with limited hardware resources (e.g., old or low-end laptops) to simulate large scale scenarios with more than a few hundred nodes. The lack of full backward compatibility of new releases may also limit the portability of user-implemented features.

Overall, development of ndnSIM faces a number of challenges, including:

- Every release of NFD and ndn-cxx need to be manually integrated with ndnSIM by applying a set of customization commits to NFD and ndn-cxx (although the set has been shrinking).

- To enable the visualization of NDN simulation scenarios and data structures (FIB,

PIT, CS), we need to patch and extend the provided NS-3 python bindings which requires substantial efforts, especially when NFD/ndn-cxx make significant changes in the supporting data structures.

- Unit-testing of software built on top of NS-3 is not always a straightforward process. The sequence of the events scheduled in the simulation environment may vary for each execution, therefore we need to ensure that existing and newly added unit tests are resistant against such random variations.

# CHAPTER 6

# Related Work

In this chapter, we first present a number of related approaches for peer-to-peer data sharing in wired infrastructure and mobile ad-hoc networks. We then present approaches for distributed dataset synchronization and explain how their objective is different than the objective of nTorrent and DAPES.

## 6.1 Peer-to-Peer Data Sharing in Wired Infrastructure Networks

BitTorrent [Coh03] is the most popular protocol for peer-to-peer file sharing designed to operate on top of TCP/IP's point-to-point packet delivery. In [DRB13], a scheme for peer-to-peer video streaming in Information Centric Networking (ICN) [ADI12] cellular environments is presented. In [LHB13], an application for peer-to-peer file synchronization in NDN is presented, but it does not provide an in-depth analysis of BitTorrent and does not deal with NDN routing scalability issues.

A protocol for peer-to-peer file sharing in ICN environments is presented in [KYK]. The "Peer-Aware Routing protocol" constructs a shared topology-aware tree connecting all the peers and each Interest is being flooded across it. This protocol does not take advantage of the hierarchical NDN names, but uses flat name-based routing to achieve file sharing.

In [LN13] - [LN12], a number of approaches for user-assisted caching in ICN are presented. They all conclude that the cache replacement policy affects the data fetching overhead, while user assistance could lead to the efficient utilization of network resources. For nTorrent, user-assisted caching could further reduce the number of requests satisfied by the peers and, thus, the torrent download time.

An advancement in the BitTorrent technology is the introduction of Distributed Hash Tables (DHTs) for decentralized peer discovery [Loe08]. DHTs provide tracking functionality similar to the one provided by the tracker but in a distributed fashion. In this design, every peer can act as a tracker by providing contact information about known peers to peers that want to participate in the swarm. Essentially, the existence of a centralized point such as a tracker is not needed. A new peer needs to know one or more "rendezvous" or "bootstrap" points (i.e., initial knowledge) in order to acquire the contact information of more peers when it first joins the swarm.

## 6.2 Peer-to-Peer Data Sharing in Off-the-Grid Scenarios

Data sharing in off-the-grid scenarios has been an active research area. Previous work in MANET [Raj06, Kri09, Sba08, Puc04, Gad09], running on top of the IP-based architecture, has relied on routing [He02, Joh01, PBD03] for communication across multiple hops. Mobility [ZXM18] introduces additional challenges, since established paths break and new ones have to be established. Previous work [Nan05, Lee06, Kle03] has advocated that alternative solutions for multi-hop communication need to be explored (e.g., application-layer gossiping, network coding, link-layer flooding). Security is not considered in the routing and the data sharing process [YLY04], while IP address configuration in such infrastructure-free environments is a challenge on its own [NP02, McA00, Mis01].

Previous work has also studied the design of general purpose architectures for MANET in NDN. E-CHANET [Ama13] is such an architecture for MANET that runs on top of IEEE 802.11 and focuses on providing stable paths and reliable transport functions toward named data. Varvello et al. [Var11] performed an analytical study of various design choices for resource discovery and forwarding in NDN-based MANETs. Meisel et al. [Mei10] proposed a design for multi-hop MANET forwarding in NDN. In this design, called "Listen First, Broadcast Later" (LFBL), nodes learn about the available data around them by overhearing the transmissions broadcasted by others.

DAPES differs from the efforts above. It tackles the problem of data sharing in off-

the-grid scenarios through semantically meaningful naming semantics understood by both applications and the NDN network. The NDN stateful forwarding plane and the naming semantics enable DAPES peers to identify what data is available across multiple hops over time, thus achieving multi-hop communication without the need of a "traditional" MANET routing protocol. Our multi-hop communication design was inspired by LFBL, however, DAPES extends LFBL's principles to fully utilize the potential understanding of intermediate nodes about the DAPES operations. Furthermore, the cryptographic primitives built into NDN provide a solid foundation for securing the data sharing process.

## 6.3 Distributed Dataset Synchronization Protocols

A number of distributed dataset synchronization (sync for short) protocols have been proposed over the last few years. Such protocols implement the transport layer of the NDN architecture and have the objective of ensuring that content generated over time by parties in a communication group will be received by all the parties in the group.

Each sync protocol proposes a different way of achieving dataset synchronization. For example, RoundSync [HCS17b] makes use of a cryptographic digest, DDSN [LMX18], and VectorSync [SAZ17] use a vector, and pSync [ZLW17] uses a bloom filter to represent the data generated by all the parties in the group.

Both nTorrent and DAPES have focused on the dissemination of static (pre-existing) content. They are also implemented as applications, which run on top of the NDN network layer.

## 6.4 Evaluation Frameworks

Three most common approaches to experimental evaluation of network architecture designs include testbed deployment, emulation, and simulation.

### 6.4.1 Testbed Deployment

The NDN team has been running a testbed since the beginning of the NDN project. The testbed currently consists of 35 nodes spanning four continents. It runs the latest versions of NFD and is open to all interested researchers to use for their own NDN experiments. However, one needs to coordinate with the testbed operators first if one's experiment requires modifications to any parts of the NFD and ndn-cxx instances on any of the testbed nodes.

The NDN project team also offers NDN experimentation on the Open Network Lab (ONL) [onl], which contains 14 programmable routers and over 100 client nodes. Compared to the testbed, ONL offers a more tightly controlled experimental environment with a rich set of measurement and monitoring tools.

Years of research efforts have resulted in a number of emulation-based (i.e., based on various virtualization and containerization technologies) testbeds being developed and deployed, such as 1) NITOS [PKI14], a facility for cloud-based wireless experimentation; 2) GENI [BCL14], a federated testbed for network experimentation; 3) Planetlab [CCR03], a general purpose, overlay testbed for broad-coverage network services; 4) Motelab [WSW05], a testbed consisting of wireless sensors; and 5) ORBIT [RSO05], a radio grid facility for wireless protocols.

The testbed approach has the advantage of requiring no changes to the software being tested, which simulation approaches often require porting prototype software to simulation tools. However it only allows for experimentation with the scale to the number of the testbed nodes (in some cases, it may require manual setup of the experimentation software on each node). For experimentation with larger networks, researchers need to resort to simulations.

### 6.4.2 Emulation

In addition to the testbed, the NDN team also developed an NDN emulator, called mini-NDN [min], which is based on the Mininet emulator [Tea14]. A number of other networking emulator extensions have been built on top of Mininet as well: 1) Mini-CCNx [CRM13], an emulator for Content Centric Networking (CCN) [ADI12]; 2) Mininet-WiFi [FAB15], an

emulator for Software Defined Wireless Networks; 3) SDDC [DAJ15], a software defined datacenter experimental framework; and 4) Maxinet [WDS14], a distributed emulator of software-defined networks.

Generally speaking, an emulation framework provides more realistic experimental conditions than a simulation framework. In the case of the NDN frameworks, mini-NDN and ndnSIM provide comparable result fidelity and result reproducibility. NFD, NLSR and real-world applications can run on mini-NDN without any changes, making an emulation experimentation easier than using ndnSIM. However, mini-NDN can scale up to medium-sized networks (up to a couple hundreds of nodes), therefore ndnSIM is again needed for larger scale experimentations.

### 6.4.3 Simulation

ccnSim [CRR13] is a chunk-level simulator for CCN networks [ADI12] (a realization of ICN). It is written in C++ under the Omnet++ framework [VH08]. Its implementation focuses on the analysis of in-network caching performance, without being a fully-featured ICN simulator. CCNPL-SIM [ccna] is another CCN simulator leveraging a platform-specific implementation of the CCN principles; every time the CCN architecture changes, the simulator codebase needs to be manually updated to include the new features, since it does not support the integration of the CCN prototype software into the simulator.

The approach of integrating prototype software in NS-3 has been taken by a few other simulators as well. OFSwitch13 [CGM16] is a simulation framework that enhances NS-3 with OpenFlow 1.3 support. Both OFSwitch13 and ndnSIM utilize the standard NS-3 Channel and NetDevice abstractions to create communication channels and make use of an external library, ofsoftswitch13 and ndn-cxx, respectively. However, OFSwitch13 models OpenFlow hardware operations and extends the NS-3 Queue class to provide some basic QoS, while ndnSIM does not deal with any hardware operations.

The NS3 DCE CCNx [dce] project leverages the Direct Code Execution (DCE) module of NS-3 to simulate the CCNx prototype [ccnb], which is the software implementation of the

legacy version of the CCN protocol. NS3 DCE CCNx uses the TapBridge model provided by NS-3 to connect a real-world host with the simulation network, while ndnSIM exclusively uses the NS-3 NetDevice abstraction and does not support the connection with an external Linux process (e.g., an external NFD instance). The DCE module is known to cause a number of scaling issues, since every node in the simulation has to run a full-sized instance of the simulated protocol code plus a DCE software layer on top of that.

DCE-Cradle [TUT13] is a simulation framework that extends NS-3 to enable the simulation of native Linux protocol stacks by reusing their original code. DCE Cradle replaces the NS-3 Socket abstraction to enable NS-3 applications to access the Linux network stack, which is similar to the direction that ndnSIM takes by allowing both ndnSIM-specific and real-world applications to access the original NDN protocol stack, thus reusing the prototype code.

# CHAPTER 7

# Conclusions & Future Work

## 7.1 Conclusions

In this dissertation, we focused on peer-to-peer file sharing systems and applications in NDN. To better understand the requirements and the challenges of such systems and applications, we first described the problems that existing peer-to-peer file sharing applications face because of running on top of the point-to-point TCP/IP network architecture. Then we presented nTorrent and DAPES, peer-to-peer file sharing applications in NDN for wired infrastructure and mobile ad-hoc networks respectively. Both applications were able to take full advantage of the NDN features, such as named-based adaptive and stateful forwarding, data centric-security, and inherent caching, to successfully deal with challenges that existing IP-based applications cannot.

Specifically, nTorrent is able to: (i) achieve traffic localization by obeying the established routing policies, (ii) adapt and operate under conditions of heavy peer churn by leveraging the NDN forwarding plane, (iii) verify the integrity of the shared data on a per network layer packet granularity, and (iv) take advantage of Interest aggregation and in-network caching to operate under flash crowd scenarios, where files become massively popular over a short period of time, thus peers that can provide the data are significantly fewer than data requesters.

DAPES provides a set of abstractions to name file collections and mechanisms to: (i) discover neighboring peers and the data they have, (ii) provide a compressed view of the data each peer has, (iii) prioritize the retrieval of rare pieces, and (iv) mitigate collisions in case of simultaneous transmissions. DAPES does not rely on MANET routing to achieve

communication over multiple wireless hops by leveraging the name-based nature of the NDN communication.

At the end of this dissertation, we presented the design of ndnSIM, the de-facto network simulator for NDN. The development and wide adoption of ndnSIM has been a result of the need for extensive experimentation and evaluation of NDN, since it fundamentally departs from the principles of the current TCP/IP network architecture. ndnSIM offers high fidelity of simulation results by offering full software integration with the real-world NDN prototype software. This integration comes at its own cost, since the increased simulation fidelity resulted in higher memory overheads compared to previous versions of the simulator. Despite this increased overhead, we are glad that ndnSIM has been an indispensable tool over the years, which has been widely used by the broader research community to facilitate their experimentation effort with the NDN architecture.

## 7.2 Future Work

The work presented in this dissertation has explored a new direction of NDN research. However, there is still a number of interesting extensions and research directions that are worth investigating in the future.

### 7.2.1 nTorrent - Future Directions

A direction of future research based on the nTorrent design would be to investigate the design choices and understand the lessons learned from nTorrent, which can help us build NDN-based transport protocols for popular existing peer-to-peer file sharing projects, such as IPFS [Ben14]. Moreover, nTorrent should be deployed and used at a larger scale on the NDN testbed. After extensive use of the nTorrent system by the NDN team and the broader research community, we will be able to fully understand the scalability potential and the trade-offs of the nTorrent design. We should also better understand the impact of in-network caching to the performance of nTorrent in cases that the content store is shared among multiple files and/or applications. We hope that nTorrent will become the basis

for the design and implementation of more advanced peer-to-peer file replication/exchange protocols in the future.

### 7.2.2 DAPES - Future Directions

Moreover, we plan to deploy the DAPES prototype onto raspberry pis and android phones and perform further experiments under diverse and adhere network conditions on the UCLA campus. We also plan to investigate the benefits/tradeoffs of data-centric cross-layer MAC protocols that filter incoming frames by name and provide an interaction interface with upper layers, such as VMAC [Elb18]. VMAC is able to achieve higher throughput and lower loss rates compared to IEEE 802.11 in ad-hoc mode. However, the data-centric design of VMAC might result in scalability problems at the MAC layer for large collections of files and/or file sizes.

Furthermore, we would like to investigate the scalability of DAPES in terms of the number of file collections that can be shared simultaneously among peers, as well as the impact of multi-hop forwarding in actual disaster recover use-cases. Another extension that is worth looking into is the scalability of the forwarding plane. More specifically, a critical factor to consider is the granularity of the data knowledge maintained by the forwarding plane, so that the network is able to scale up to a thousands of file collections.

### 7.2.3 ndnSIM - Future Directions

For ndnSIM, we plan to continue developing the codebase by adding more features, as they are requested by our user community and investigate how to minimize the effort needed by users in order to port external components into ndnSIM. We also plan to fully integrate ndnSIM as a module of the main ns-3 codebase and make ndnSIM available as an application package to the ns-3 application store, aiming to maximize the exposure of users to ndnSIM.

To make the research community aware of the lower level details of the simulation development process, we plan to extend our documentation and publish programming "HOW-TOs" on the ndnSIM website. We also plan to enrich our collection of plug-n-play simulation

scenarios to demonstrate new interesting use cases and network environments that can take advantage of NDN's communication model. We have been also working closely with the NFD Team to enable conditional compilation of NFD components for ndnSIM. This can be extended to the implemented NFD and ndn-cxx optimizations for ndnSIM, so that every new release of the NDN prototypes can be made automatically compatible with ndnSIM without requiring manual integration.

To further improve ndnSIM's scalability, we plan to investigate in detail the memory consumption of each simulated forwarder instance, and come up with a concrete plan to reduce this memory consumption. We also plan to make the backward compatibility a high priority in future releases. To the extent possible, we will work with the ndnSIM user community to minimize, if not eliminate, the need for users to modify their scenarios and ndnSIM extensions used in the previous simulations.

# REFERENCES


[AAH13] Fadi M Al-Turjman, Ashraf E Al-Fagih, and Hossam S Hassanein. "A value-based cache replacement approach for Information-Centric Networks." In *Local Computer Networks Workshops (LCN Workshops), 2013 IEEE 38th Conference on*, pp. 874–881. IEEE, 2013.

[AB12] RJ Atkinson and SN Bhatti. "Identifier-locator network protocol (ILNP) architectural description." Technical report, 2012.

[ADI12] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. "A survey of information-centric networking." *IEEE Communications Magazine*, **50**(7), 2012.

[Ama13] Marica Amadeo et al. "E-CHANET: Routing, forwarding and transport in Information-Centric multihop wireless networks." 2013.

[AMG15] Christos Argyropoulos, Spyridon Mastorakis, Kostas Giotis, Georgios Androulidakis, Dimitrios Kalogeras, and Vasilis Maglaris. "Control-plane slicing methods in multi-tenant software defined networks." In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 612–618. IEEE, 2015.

[AS15] Alexander Afanasyev, Junxiao Shi, et al. "NFD Developer's Guide." Tech. Rep. NDN-0021, NDN, 2015.

[ASY15] A Afanasyev, J Shi, Y Yu, and S DiBenedetto. "ndn-cxx: NDN C++ library with eXperimental eXtensions: Library and Applications Developer's Guide." *NDN Project (named-data. net)*, 2015.

[AYW15] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. "SNAMP: Secure Namespace Mapping to Scale NDN Forwarding." In *Proc. of IEEE Global Internet Symposium*, April 2015.

[BCC06] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, et al. "Improving traffic locality in BitTorrent via biased neighbor selection." In *Proc. of International Conference on Distributed Computing Systems*, 2006.

[BCL14] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. "GENI: A federated testbed for innovative network experiments." *Computer Networks*, **61**:5–23, 2014.

[BDN12] Mark Baugher, Bruce Davie, Ashok Narayanan, and Dave Oran. "Self-verifying names for read-only named data." In *INFOCOM Workshops*, volume 12, pp. 274–279, 2012.

[Ben14] Juan Benet. "Ipfs-content addressed, versioned, p2p file system." *arXiv preprint arXiv:1407.3561*, 2014.

[CAG05] Stuart Cheshire, Bernard Aboba, and Erik Guttman. "Dynamic configuration of IPv4 link-local addresses." Technical report, 2005.

[CB08] David R Choffnes and Fabián E Bustamante. "Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems." In *ACM Comp. Comm. Review*, 2008.

[ccna] "CCNPL-SIM simulation framework." http://systemx.enst.fr/ccnpl-sim.

[ccnb] "CCNx Project." http://blogs.parc.com/ccnx/.

[CCR03] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. "Planetlab: an overlay testbed for broad-coverage services." *ACM SIGCOMM Computer Communication Review*, **33**(3):3–12, 2003.

[CGM16] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. "OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 Support." In *Proceedings of the Workshop on ns-3*, pp. 33–40. ACM, 2016.

[CH] Benjamin Rainer Christian Kreuzberger, Daniel Posch and Hermann Hellwagner. "Demo: amus-ndnSIM - Adaptive Multimedia Streaming Simulator for NDN.".

[chra] "ChronoSync Redmine Issue 3928." https://redmine.named-data.net/issues/3928.

[chrb] "ChronoSync Simulation Repository." https://github.com/spirosmastorakis/ChronoSync.

[CKM17] Kevin Chan, Bongjun Ko, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "Fuzzy Interest Forwarding." In *Proceedings of the Asian Internet Engineering Conference*, pp. 31–37. ACM, 2017.

[Coh] Bram Cohen. "The BitTorrent protocol specification.".

[Coh03] Bram Cohen. "Incentives build robustness in BitTorrent." In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pp. 68–72, 2003.

[CRM13] Carlos Cabral, Christian Esteve Rothenberg, and Maurício Ferreira Magalhães. "Reproducing real NDN experiments using mini-CCNx." In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pp. 45–46. ACM, 2013.

[CRR13] Raffaele Chiocchetti, Dario Rossi, and Giuseppe Rossini. "ccnsim: An highly scalable ccn simulator." In *Communications (ICC), 2013 IEEE International Conference on*, pp. 2309–2314. IEEE, 2013.

[CT90] David D Clark and David L Tennenhouse. "Architectural considerations for a new generation of protocols." *ACM SIGCOMM Computer Communication Review*, **20**(4):200–208, 1990.

[DAJ15] Ala Darabseh, Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. "SDDC: A software defined datacenter experimental framework." In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pp. 189–194. IEEE, 2015.

[dce] "NS3 DCE CCNx Quick Start." http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/index.html.

[Dee96] Steve Deering. "The Map & Encap Scheme for scalable IPv4 routing with portable site prefixes." *Presentation, Xerox PARC*, 1996.

[DOS17a] R Droms, D Oran, and ICN Ping Protocol Specification. "ICNRG S. Mastorakis Internet-Draft UCLA Intended status: Experimental J. Gibson Expires: February 26, 2018 I. Moiseenko Cisco Systems." 2017.

[DOS17b] R Droms, D Oran, and ICN Traceroute Protocol Specification. "ICNRG S. Mastorakis Internet-Draft UCLA Intended status: Experimental J. Gibson Expires: March 25, 2018 I. Moiseenko Cisco Systems." 2017.

[DRB13] Andrea Detti, Bruno Ricci, and Nicola Blefari-Melazzi. "Peer-to-peer live adaptive video streaming for Information Centric cellular networks." In *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*, pp. 3583–3588. IEEE, 2013.

[Elb18] Mohammed Elbadry et al. "Poster: A Raspberry Pi Based Data-Centric MAC for Robust Multicast in Vehicular Network." In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 714–716. ACM, 2018.

[FAB15] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. "Mininet-WiFi: Emulating software-defined wireless networks." In *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 384–389. IEEE, 2015.

[FLM13] Dino Farinacci, Darrel Lewis, David Meyer, and Vince Fuller. "The locator/ID separation protocol (LISP)." 2013.

[Gad09] Neelakantam Gaddam et al. "Study of BitTorrent for file sharing in Ad Hoc networks." In *IEEE Conference on Wireless Communication and Sensor Networks*, 2009.

[HAA13] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. "NLSR: Named-data Link State Routing Protocol." In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pp. 15–20. ACM, 2013.

[HCS17a] Pedro de-las Heras-Quirós, Eva M. Castro, Wentao Shang, Yingdi Yu, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "The Design of RoundSync Protocol." Technical Report NDN-0048, NDN, April 2017.

[HCS17b] Pedro de-las Heras-Quirós, Eva M Castro, Wentao Shang, Yingdi Yu, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "The design of RoundSync protocol." Technical report, Technical Report NDN-0048, NDN, 2017.

[He02] Guoyou He. "Destination-sequenced distance vector protocol." *Networking Laboratory, Helsinki University of Technology*, 2002.

[HSH08] D Harrison, S Shalunov, and G. Hazel. "BitTorrent Local Tracker Discovery Protocol.", October 2008.

[HV02] Gavin Holland and Nitin Vaidya. "Analysis of TCP performance over mobile ad hoc networks." *Wireless Networks*, **8**(2-3):275–288, 2002.

[Joh01] David Johnson et al. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks." *Ad hoc networking*, 2001.

[Kle03] Alexander Klemm et al. "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks." In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 4, pp. 2758–2763. IEEE, 2003.

[Kri09] Amir Krifa et al. "Bithoc: A content sharing application for wireless ad hoc networks." In *Pervasive Computing and Communications.*, 2009.

[KYK] Younghoon Kim, Ikjun Yeom, and Yusung Kim. "Scalable and Efficient File Sharing in Information-Centric Networking.".

[Lee06] Uichin Lee et al. "Code torrent: content distribution using network coding in vanet." In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, 2006.

[LHB13] Jared Lindblom, M Huang, Jeff Burke, and Lixia Zhang. "FileSync/NDN: Peer-to-peer file sync over Named Data Networking." Tech. Rep NDN-0012, NDN, 2013.

[LLD11] Stevens Le Blond, Arnaud Legout, and Walid Dabbous. "Pushing BitTorrent locality to the limit." *Computer Networks*, 2011.

[LMX18] Tianxiang Li, Spyridon Mastorakis, Xin Xu, Haitao Zhang, and Lixia Zhang. "Data Synchronization in Ad Hoc Mobile Networks." In *Proceedings of the 5th ACM Conference on Information-Centric Networking*. ACM, 2018.

[LN12] HyunYong Lee and Akihiro Nakao. "Efficient user-assisted content distribution over information-centric network." In *NETWORKING 2012*, pp. 1–12. Springer, 2012.

[LN13] HyunYong Lee and Akihiro Nakao. "User-assisted in-network caching in information-centric networking." *Computer Networks*, **57**(16):3142–3153, 2013.

[Loe08] Andrew Loewenstern. "DHT protocol." *BitTorrent. org*, 2008.

[MAM15] 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.

[MAM16] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. "ndnSIM 2: An updated NDN simulator for NS-3." Technical report, Technical Report NDN-0028, Revision 2. NDN, 2016.

[MAP18] Spyridon Mastorakis, Tahrina Ahmed, and Jayaprakash Pisharath. "ISA-Based Trusted Network Functions And Server Applications In The Untrusted Cloud." *arXiv preprint arXiv:1802.06970*, 2018.

[MAY16] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, Michael Sweatt, and Lixia Zhang. "nTorrent: BitTorrent in Named Data Networking." March 2016.

[MAY17a] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, Michael Sweatt, and Lixia Zhang. "nTorrent: BitTorrent in Named Data Networking." ICCCN, 2017.

[MAY17b] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. "nTorrent: Peer-to-Peer File Sharing in Named Data Networking." In *26th International Conference on Computer Communications and Networks (ICCCN)*, 2017.

[MAZ17] Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "On the evolution of ndnSIM: An open-source simulator for NDN experimentation." *ACM SIGCOMM Computer Communication Review*, **47**(3):19–33, 2017.

[McA00] AJ McAuley et al. "Self-configuring networks." In *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, 2000.

[MCK18] Spyridon Mastorakis, Kevin Chan, Bongjun Ko, Alexander Afanasyev, and Lixia Zhang. "Experimentation With Fuzzy Interest Forwarding in Named Data Networking." *arXiv preprint arXiv:1802.03072*, 2018.

[Mei10] Michael Meisel et al. "Listen first, broadcast later: Topology-agnostic forwarding under high dynamics." In *Annual conference of international technology alliance in network and information science*, 2010.

[Mer87] Ralph C Merkle. "A digital signature based on a conventional encryption function." In *Conference on the theory and application of cryptographic techniques*, pp. 369–378. Springer, 1987.

[MGA18] Spyridon Mastorakis, Peter Gusev, Alexander Afanasyev, and Lixia Zhang. "Real-Time Data Retrieval in Named Data Networking." In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pp. 61–66. IEEE, 2018.

[min] "Mini-NDN GitHub." https://github.com/named-data/mini-ndn.

[Mis01] Archan Misra et al. "Autoconfiguration, registration, and mobility management for pervasive computing." 2001.

[Moi14] Ilya Moiseenko. "Fetching content in Named Data Networking with embedded manifests." Tech. Rep. NDN-0025, NDN, 2014.

[Nan05] Alok Nandan et al. "Co-operative downloading in vehicular ad-hoc wireless networks." 2005.

[ndna] "ndnSIM GitHub Repository." https://github.com/named-data-ndnSIM/ndnSIM.

[ndnb] "ndnSIM Mailing List." http://www.lists.cs.ucla.edu/mailman/listinfo/ndnsim.

[ndnc] "ndnSIM Website." http://ndnsim.net.

[NDN14] NDN Project. "NDN Packet Format Specification." Online: `http://named-data.net/doc/ndn-tlv/`, 2014.

[Net11] Named Data Networking. "NDN packet format specification.", 2011.

[nls] "NLSR-SIM." https://github.com/3rd-ndn-hackathon/ndnSIM-NLSR.

[NP02] Sanket Nesargi and Ravi Prakash. "MANETconf: Configuration of hosts in a mobile ad hoc network." In *INFOCOM 2002*, 2002.

[ns3] "ns-3." http://www.nsnam.org/.

[onl] "Open Networking Lab." http://onlab.us.

[PBD03] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. "Ad hoc on-demand distance vector (AODV) routing." Technical report, 2003.

[Per00] Charles E Perkins. "IP address autoconfiguration for ad hoc networks." *draft-ietf-manet-autoconf-01. txt*, 2000.

[Per08] Robin Perkins. "Zeroconf Peer Advertising and Discovery." BEP 26, 2008.

[PKI14] Katerina Pechlivanidou, Kostas Katsalis, Ioannis Igoumenos, Dimitrios Katsaros, Thanasis Korakis, and Leandros Tassiulas. "NITOS testbed: A cloud based wireless experimentation facility." In *Teletraffic Congress (ITC), 2014 26th International*, pp. 1–6. IEEE, 2014.

[PRH17] Daniel Posch, Benjamin Rainer, and Hermann Hellwagner. "SAF: Stochastic Adaptive Forwarding in Named Data Networking." *IEEE/ACM Transactions on Networking*, 2017.

[Puc04] Himabindu Pucha et al. "Ekta: An efficient dht substrate for distributed applications in mobile ad hoc networks." In *IEEE Workshop on Mobile Computing Systems and Applications*, 2004.

[Raj06] Sundaram Rajagopalan et al. "A cross-layer decentralized BitTorrent for mobile Ad Hoc networks." In *Mobile and Ubiquitous Systems*, 2006.

[rou] "RoundSync Simulation Repository." https://github.com/spirosmastorakis/RoundSync.

[RSO05] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols." In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pp. 1664–1669. IEEE, 2005.

[SAZ17] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. "VectorSync: distributed dataset synchronization over named data networking." In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 192–193. ACM, 2017.

[Sba08] Mohamed Karim Sbai et al. "BitHoc: Bittorrent for wireless ad hoc networks." *INRIA Sophia Antipolis, France*, 2008.

[SKS14] Jan Seedorf, Dirk Kutscher, and Fabian Schneider. "Decentralised binding of self-certifying names to real-world identities for assessment of third-party messages in fragmented mobile networks." In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 416–421. IEEE, 2014.

[Tar10] Sasu Tarkoma. *Overlay Networks: Toward Information Networking.* CRC Press, 2010.

[Tea14] Mininet Team. "Mininet." http://mininet. org, 2014.

[tes] "NDN Testbed." http://ndndemo.arl.wustl.edu.

[TUT13] Hajime Tazaki, Frédéric Urbani, and Thierry Turletti. "DCE cradle: simulate network protocols with real stacks for better realism." In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 153–158. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[TYS16] Atsushi Tagami, Tomohiko Yagyu, Kohei Sugiyama, Mayutan Arumaithurai, Kenichi Nakamura, Toru Hasegawa, Tohru Asami, and KK Ramakrishnan. "Name-based push/pull message dissemination for disaster message board." In *Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on*, pp. 1–6. IEEE, 2016.

[Var11] Matteo Varvello et al. "On the design of content-centric MANETs." In *International Conference on Wireless On-Demand Network Systems and Services*, 2011.

[VH08] András Varga and Rudolf Hornig. "An overview of the OMNeT++ simulation environment." In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[VMA16] Satyanarayana Vusirikala, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "Hop-by-hop best effort link layer reliability in named data networking." Technical report, NDN, Technical Report, NDN-0041, 2016.

[WAK12] Lucas Wang, Alexander Afanasyev, Romain Kuntz, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. "Rapid traffic information dissemination using named data." In *Proceedings of the 1st ACM workshop on emerging name-oriented mobile networking design-architecture, algorithms, and applications*, pp. 7–12. ACM, 2012.

[WDH10] Di Wu, Prithula Dhungel, Xiaojun Hei, Chao Zhang, and Keith W Ross. "Understanding peer exchange in bittorrent systems." In *Proc. of Peer-to-Peer Computing (P2P)*, 2010.

[WDS14] Philip Wette, Martin Draxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. "Maxinet: Distributed emulation of software-defined networks." In *Networking Conference, 2014 IFIP*, pp. 1–9. IEEE, 2014.

[WSW05] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. "Motelab: A wireless sensor network testbed." In *Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 68. IEEE Press, 2005.

[YAM13] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. "A case for stateful forwarding plane." *Computer Communications*, **36**(7):779–791, 2013.

[YLY04] Hao Yang, Haiyun Luo, Fan Ye, SW Lu, and Lixia Zhang. "Security in mobile ad hoc networks: challenges and solutions." 2004.

[ZA13] Zhenkai Zhu and Alexander Afanasyev. "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking." In *ICNP*, pp. 1–10, 2013.

[Zha14] Lixia Zhang et al. "Named data networking." *ACM SIGCOMM Computer Communication Review*, **44**(3):66–73, 2014.

[Zhu11] Yi-Hua Zhu et al. "Performance analysis of the binary exponential backoff algorithm for IEEE 802.11 based mobile ad hoc networks." In *IEEE ICC*, 2011.

[zip] "Zipf-Mandelbrot Law.".

[ZLW17] Minsheng Zhang, Vince Lehman, and Lan Wang. "Scalable name-based data synchronization for named data networking." In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pp. 1–9. IEEE, 2017.

[ZXM18] Yu Zhang, Zhongda Xia, Spyridon Mastorakis, and Lixia Zhang. "KITE: Producer Mobility Support in Named Data Networking." In *ICN'18: 5th ACM Conference on Information-Centric Networking (ICN'18)*, pp. 1–12, 2018.

[ZYZ18] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. "An overview of security support in Named Data Networking." *IEEE Communications Magazine*, **56**(11):62–68, 2018.

[ZZN18] Zhiyi Zhang, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. "Security support in named data networking." Technical report, Technical Report. Available online: https://named-data. net/wp-content . . . , 2018.