nTorrent: Peer-to-Peer File Sharing in Named Data Networking

Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, Lixia Zhang

University of California, Los Angeles
{mastorakis, aa, yingdi, lixia}@cs.ucla.edu

***Abstract*—BitTorrent is a popular application for peer-to-peer file sharing in today's Internet. In the core of BitTorrent is a data-centric data dissemination approach, where peers request pieces of the file(s) from each other, and each retrieved piece can be verified using cryptographic hashes. This process looks similar to that of the Named Data Networking (NDN) architecture, but is realized completely at the application level on top of the channel-based TCP/IP networking. Consequently BitTorrent has to maintain an overlay network of peers, discover and share IP addresses of peers, keep track the quality of established connections, and incentivize other peers to share data. This paper presents the design of nTorrent, a BitTorrent-like application that is based on the natively data-centric NDN network architecture. Through analysis and simulation-based experimentations, the paper exposes impacts of the network-level data-centricity on the design choices, implementation complexity, and protocol operations.**

***Index Terms*—Named Data Networking (NDN), Information Centric Networking (ICN), BitTorrent, Peer-to-Peer, File Sharing**

I. Introduction

BitTorrent [1] is a popular peer-to-peer file sharing application in today's Internet. The shared files are split into "named" blocks, each identified by its sequence number in the torrent. *Any* peer can request a data block from *any* other peers of the torrent. Each retrieved block can be checked for correctness using its cryptographic digest which is included as part of the torrent description. However, this data-centric view exists only at the *application layer*; peers need to discover IP addresses of other peers, select specific IP addresses to use to set up TCP connections, estimate quality of the connections, as well as incentivize data sharing through the tit-for-tat mechanism.

Named Data Networking (NDN) [2] is a proposed Internet architecture that makes named and secured data packets the centerpiece of the network architecture. NDN enables applications to focus exclusively on what data to fetch, leaving to the network to decide exactly from where to fetch the data.

In this paper, our main goal is to understand the impact of implementing the data-centric logic at different layers in the network architecture. We first analyze the difficulties that that BitTorrent faces as a data-centric application protocol running over TCP/IP (Section II). We then identify the similarities and differences between BitTorrent and NDN, and articulate how NDN may provide the BitTorrent-like data dissemination more effectively and efficiently (Section III). Based on the above understanding, we sketch out the design of nTorrent (Section IV) and conduct an in-depth evaluation of the design through comparative simulations between nTorrent and BitTorrent (Section V).

Our contribution of this work is three-fold. First, we analyze the challenges that BitTorrent faces under different lights: the incongruity between BitTorrent's data-centric view at the application layer and the point-to-point communication model TCP/IP supports. Second, we design nTorrent to achieve BitTorrent-like data dissemination in an NDN network, which results in a much simpler implementation. Third, we report our findings from the simulation studies which expose new issues raised by directly using application data names at network layer. We further discuss possible extensions to the nTorrent design to make it a useful tool for content sharing over NDN networks.

II. 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.

A. 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 [3]. 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.

B. 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) [4] and a Peer Exchange (PEX) [5] 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.

C. 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.

D. 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 [6], to ensure that each piece of the torrent is stored on as many available peers as possible.

E. 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.

F. 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) [6], and "unchoke" only if the peer shares data. The choking state can be temporary, as optimistic unchoking [6] ensures that a leecher will be unchoked when it increases its upload rate.

G. 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 [7], [8], [9]. 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 [10] 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 [11] 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.

III. BitTorrent in NDN terms

To help the reader gain insight into how the BitTorrent functionality naturally fits in NDN, in this section, we first provide an overview of NDN and we then present the similarities and differences between BitTorrent and NDN. We also highlight the essential NDN properties for an NDN-native BitTorrent implementation.

A. 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. An NDN router (Figure 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.

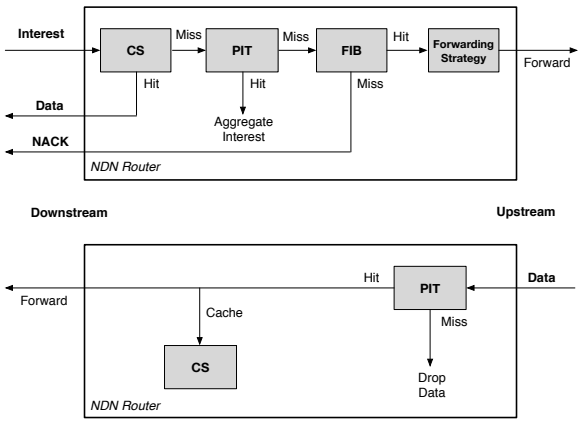


Fig. 1: Packet processing and forwarding by an NDN router

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. 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 both consumer applications and NDN routers 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 [12]).

2) NDN Routing and Forwarding: The NDN routing protocols (e.g., NLSR [13]) 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 [14]:

- **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 [14] 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.

B. 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 I.

TABLE I: 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

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) 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.

C. NDN-Native Peer-to-Peer File Sharing

Based on our previous analysis, we conclude that: 1) a peer in NDN acts as a consumer and a producer at the same time; as a consumer, it downloads data from others, and as a producer, it uploads data to others, 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 III-A2), 2) a peer in NDN does not have to explicitly discover others and evaluate their performance; it will express a request for data and the forwarding plane will bring this data back, 3) an NDN-native BitTorrent application can get away from needing a tracker; traffic localization and parallel downloading are done by the forwarding plane and 4) redundant fetching of the same data across the same path can be eliminated; NDN's native support for multicast data delivery and in-network caching reduce both network load and data retrieval delay.

IV. nTorrent design

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

A. .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 2a, 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 (section II-A).

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 [15], [16], [17]), called *file manifests* (Figure 2b). 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; peers then download the file manifests and eventually the individual packets of each file in the torrent.

For large torrents, the original torrent publisher, based on the concept of Application Layer Framing [18], 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.

B. Naming Conventions

The goal of the nTorrent naming conventions is twofold:

- Easily identify the name of the selected torrent, the files in a torrent and the individual packets in a file.
- Provide a full name for each packet, so that routers can directly verify the integrity of the packet (section IV-D).

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 4:

- 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.

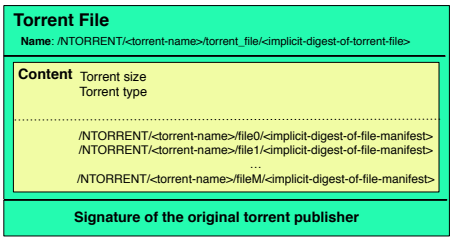
C. Sequential Data Retrieval

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

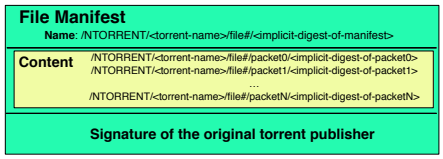
nTorrent should fetch data in a way that maximizes the utilization of in-network caching. Intuitively, fetching data sequentially is likely to contribute to the utilization of caches during simultaneous downloads. A peer requests data, starting from the first packet in the first file in the torrent, 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.

D. Packet Level Integrity

Peers request data using full names, enabling routers to verify data integrity on a packet-level basis and ensuring that an application never receives bogus data. This verification comes with minimal router processing cost (no signature validation per data packet); to satisfy a pending Interest with a full name, the router first computes the implicit digest of the incoming data packet and then searches for matching PIT entries. If the computed digest does not match the digest in the packet name, the router discards the packet.



(a) Structure of .torrent file



(b) Structure of file manifest

Fig. 2: .Torrent File & File Manifest

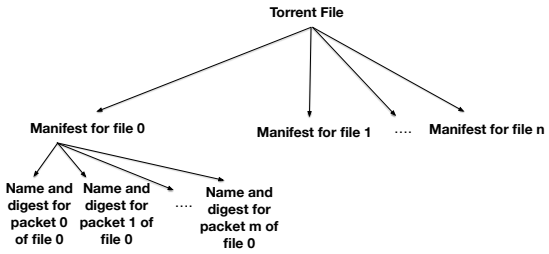


Fig. 3: Hierarchical process of secure torrent retrieval

Fig. 4: nTorrent Interest format

E. 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.

We study the effect of routing announcements on our design in section V.

F. Baseline Application Scenario

As illustrated in Figure 5, 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.

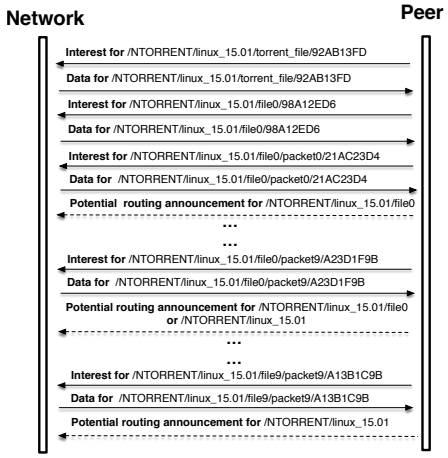


Fig. 5: Example of downloading a linux distribution torrent

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.

V. Simulation Study

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 IV-E, 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 [19], an NDN simulator based on NS-3 [20], featuring the forwarding pipelines of the NDN Forwarding Daemon (NFD) [21]. We also implemented the plain BitTorrent functionality with a rarest-piece-first selection strategy as a separate NS-3 module. We should note that the nTorrent application [22] has been deployed and can be used on the NDN testbed.

A. Simulation Setup

We use the topologies shown in Figure 6. The first one (Figure 6a) offers multiple paths with different costs (in terms of delays) to peers (the bandwidth of the links is the same). The second one (Figure 6b) 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. Figure 6c 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.

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 the forwarding strategy discussed in section III-A2, 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 IV-C.

B. Impact Of Routing Announcements

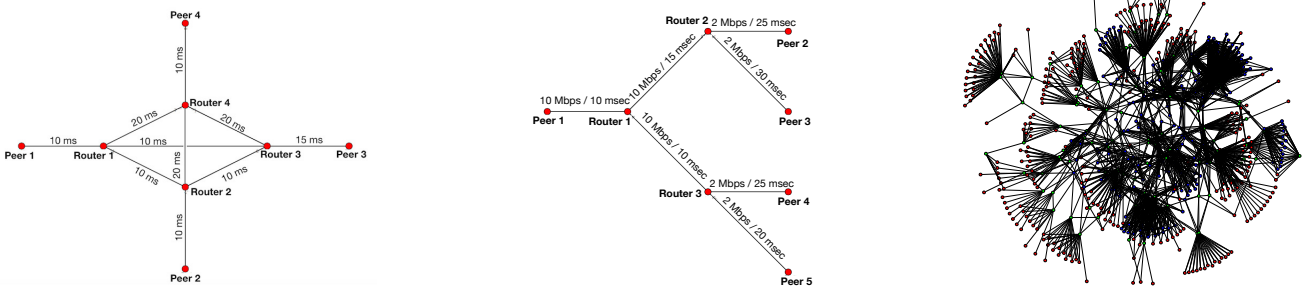
We first study the impact of routing announcements on nTorrent. We measure the amount of NACKs generated and the time needed for data retrieval. We use the topology illustrated in Figure 6a, where a seeder initially uploads a torrent and some leechers download it. We have disabled in-network caching 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 II); all the data can be retrieved through each outgoing FIB entry face.

For approximate routing announcements, local error recovery is low (Table II), 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.

The torrent distribution duration is illustrated in Table III. 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.



(a) Topology with router node degree equal to 4 (b) Topology with router node degree equal to 3 (c) Rocketfuel AT&T topology

Fig. 6: Simulation topologies

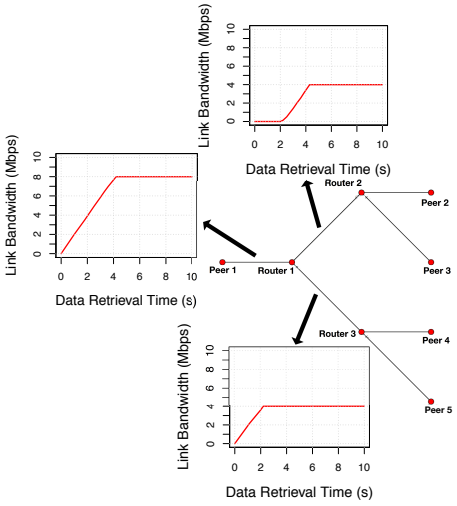


Fig. 7: nTorrent download speed (seeder peers having the entire torrent content)

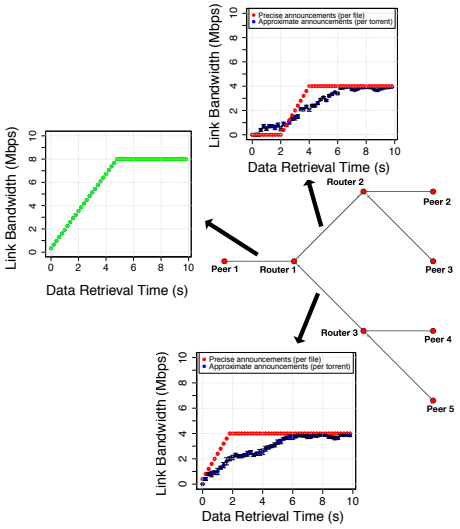


Fig. 8: nTorrent download speed (leecher peers)

TABLE II: 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 %

TABLE III: Duration until all the peers download the torrent

Precise routing announcements (per file)	124 ± 2 sec
Approximate routing announcements (per torrent)	127 ± 3 sec

C. 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 6b. 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 7, 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 8, 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.

D. Flash Crowd Scenario

In this experiment, we simulate a flash crowd scenario (topology 6c). Such scenarios are challenging for BitTorrent, since the upload bandwidth of a seeder is dominated by leechers' 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 IV, 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 V, 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 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 less 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, 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 VI, 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.

VI. nTorrent Design Extension To Scale NDN Routing

In the previous sections, we presented the nTorrent design assuming the native NDN approach of routing directly on application names. 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 to maintain the scalability of NDN forwarding. Afanasyev et al. [23] proposed the SNAMP scheme, which is briefly discussed below. We also discuss how nTorrent leverages SNAMP.

A. SNAMP Overview

SNAMP [23] proposes a secure 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. This association is called LINK, which is a piece of named data signed by this producer. A consumer application attaches the LINK to the expressed Interest, which will act as guidance (hint) for the Interest forwarding across the Internet. A LINK is published by the data producer in NDNS (DNS for NDN) [24] and the consumer will retrieve it from there. To reduce the danger of cache poisoning, in-network caches may store the LINK along with a data packet, so that only Interests carrying the same LINK retrieve the cached data.

SNAMP does not impose changes to local communication; LINK is needed by Internet backbone routers that do not know how to further forward an Interest based solely on the application name. When an Interest reaches a domain, where the attached routable prefix is available, the Interest forwarding is performed based on its name. The goal of SNAMP is to provide a framework to achieve NDN forwarding on Internet scale with minimal changes to the application logic.

B. nTorrent Leveraging SNAMP

The requirement that a LINK has to be signed by the original data producer has arisen from concerns that any router along the path could change the LINK content and forward an Interest towards compromised data sources. nTorrent uses full names to guarantee the integrity of received data, which can be verified both by the peers and the network, thus, the use of unsigned LINKs can be allowed. This property is critical for peer-to-peer environments, where peers come and go in an unpredictable fashion, therefore, a requirement for signed LINKs would introduce significant complexity. The implicit digest appended to the data name also allows for Interests to

TABLE IV: 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
nTorrent - Precise announcements (per file)	204 ± 2	197 ± 2	~176	225 ± 2	211 ± 1	~199	244 ± 2	224 ± 1	~214
nTorrent - Approximate announcements (per torrent)	216 ± 3	210 ± 2	~181	238 ± 2	222 ± 2	~202	259 ± 2	233 ± 1	~216
BitTorrent	225 ± 3	225 ± 3	225 ± 3	261 ± 3	261 ± 3	261 ± 3	319 ± 3	319 ± 3	319 ± 3

TABLE V: 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 %

TABLE VI: 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

be satisfied by a cached data packet no matter if the cached LINK matches the Interest LINK or not.

SNAMP preserves the desired nTorrent properties: 1) SNAMP requires applications to attach routable prefixes to the Interests to help the forwarding plane find the data across the Internet. However, this is fundamentally different from explicitly managing point-to-point connections as imposed by TCP/IP to BitTorrent, 2) the download speed maximization property holds, since the forwarding strategy logic does not change, 3) the cache hit/miss behavior is the same; cached data can be retrieved no matter if an Interest's LINK matches the cached LINK, so that peers do not have to satisfy every single Interest, and 4) data integrity verification based on full names is not affected by SNAMP.

We should note that a routable prefix is fundamentally different than a peer/publisher identification; it merely indicates a direction (hint) for Interest forwarding. Therefore, a single torrent may be available across the Internet under a number of routable prefixes, and under a single routable prefix a number of torrents may be available. For example, a torrent may be available under the "`/ucla`" and "`/att`" prefixes, and under the "`/ucla`" prefix, torrents for all the linux distributions may be available.

After downloading the .torrent file, a peer will query NDNS, where a LINK for the desired torrent is stored. This LINK will include a number of routable prefixes, under which the desired torrent is available. When the peer fetches the LINK from NDNS, it can: 1) start downloading file manifests and/or packets in the torrent; it creates an unsigned LINK containing a number of routable prefixes and attaches it to its Interests, and/or 2) acquire more routable prefixes; it sends Interests towards known routable prefixes to contact other peers in the swarm. The format of those Interests is shown in Figure 9. The first component refers to the nTorrent application and the second is the torrent name. The third is the sender peer's own routable prefix, under which the torrent data will be published when downloaded; the peer injects its routable prefix into the swarm, so that others can attach it to their Interests to

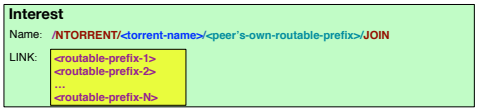


Fig. 9: Interest expressed by a peer to acquire routable prefixes

download data. The last component indicates that the peer wants to join torrent downloading and learn routable prefixes.

As peers come and go, some routable prefixes may become obsolete or even malicious locations may inject false routable prefixes into the system. To deal with these cases, peers attach diverse routable prefixes (i.e., names with no or small prefix matching) to their Interests, so that the forwarding plane can decide which prefix(es) to use to fetch data. If needed, peers can retrieve new routable prefixes, while obsolete ones can be evicted by maintaining a per routable prefix "soft state".

Overall, the state required by nTorrent to follow the SNAMP approach is maintained: 1) across the network (FIBs) in the form of globally routable prefixes (FIB entries), and 2) in NDNS (fully distributed database) in the form of LINKs containing routable prefixes to help newcomer peers bootstrap into the swarm.

VII. Related Work

BitTorrent [1] 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 [25], a scheme for peer-to-peer video streaming in Information Centric Networking (ICN) [26] cellular environments is presented. In [27], 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 [28]. 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 [29] - [31], 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.

VIII. Conclusions

This paper describes our design and experimentation with nTorrent. We have learned several lessons from this exercise, which seem worth sharing with the broader community. First, by letting the network directly use application names for data fetching, NDN enable applications to focus on what data to fetch, leaving to the network to decide from where to fetch data. This not only simplifies the design of nTorrent, but also leads to most efficient and resilient data dissemination as an NDN network is able to fetch data nearest available locations, make use of multiple paths, quickly adapting to failures as well as data source changes, and obeying routing policies.

At the same time, letting networks share application data names also brings up routing scalability concern. We propose to use the developed solution of NDN *LINK* to mitigate this issue, and notice that new work is needed to handle the binding between data names and the LINKs needed to reach them, to fully achieve the goal of "applications focusing on what data to fetch only."

Finally, compared to BitTorrent's handling of data authentication at application level, NDN enables data authenticity checking at network packet level. There for in disseminating static data, nTorrent can embed data digests in NDN packet names to enable routers to detect and drop bogus contents at very first router they cross.

Acknowledgment

This work is partially supported by the National Science Foundation under award CNS-1345318 and CNS-1629922.

References

[1] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson *et al.*, "Named data networking," *Comp. Comm. Review*, 2014.

[3] S. Tarkoma, *Overlay Networks: Toward Information Networking.* CRC Press, 2010.

[4] A. Loewenstern, "DHT protocol," *BitTorrent. org*, 2008.

[5] D. Wu, P. Dhungel, X. Hei, C. Zhang, and K. W. Ross, "Understanding peer exchange in bittorrent systems," in *Proc. of Peer-to-Peer Computing (P2P)*, 2010.

[6] B. Cohen, "The BitTorrent protocol specification."

[7] R. Bindal, P. Cao, W. Chan, J. Medved *et al.*, "Improving traffic locality in BitTorrent via biased neighbor selection," in *Proc. of International Conference on Distributed Computing Systems*, 2006.

[8] S. Le Blond, A. Legout, and W. Dabbous, "Pushing BitTorrent locality to the limit," *Computer Networks*, 2011.

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

[10] R. Perkins, "Zeroconf peer advertising and discovery," BEP 26, 2008.

[11] D. Harrison, S. Shalunov, and G. Hazel, "BitTorrent local tracker discovery protocol," October 2008.

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

[13] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: Named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*. ACM, 2013, pp. 15–20.

[14] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013.

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

[16] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-verifying names for read-only named data." in *INFOCOM Workshops*, vol. 12, 2012, pp. 274–279.

[17] C. Tschudin and C. Wood, "File-like icn collection (flic)," *Internet Engineering Task Force, Internet-Draft draft-tschudin-icnrg-flic-00*, 2016.

[18] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 4, pp. 200–208, 1990.

[19] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnsim 2: An updated ndn simulator for ns-3," Technical Report NDN-0028, Revision 2, NDN, Tech. Rep., 2016.

[20] "NS-3 Simulation Framework," http://www.nsnam.org/. [Online]. Available: http://www.nsnam.org/

[21] A. Afanasyev, J. Shi *et al.*, "NFD Developer's Guide," NDN, Tech. Rep. NDN-0021, 2015.

[22] "nTorrent Implementation." [Online]. Available: https://github.com/spirosmastorakis/nTorrent

[23] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "SNAMP: Secure namespace mapping to scale NDN forwarding," in *Proc. of IEEE Global Internet Symposium*, April 2015.

[24] A. Afanasyev, "Addressing operational challenges in Named Data Networking through NDNS distributed database," Ph.D. dissertation, UCLA, September 2013.

[25] A. Detti, B. Ricci, and N. 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*. IEEE, 2013, pp. 3583–3588.

[26] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, 2012.

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

[28] Y. Kim, I. Yeom, and Y. Kim, "Scalable and efficient file sharing in information-centric networking."

[29] H. Lee and A. Nakao, "User-assisted in-network caching in information-centric networking," *Computer Networks*, vol. 57, no. 16, pp. 3142–3153, 2013.

[30] F. M. Al-Turjman, A. E. Al-Fagih, and H. S. Hassanein, "A value-based cache replacement approach for information-centric networks," in *Local Computer Networks Workshops (LCN Workshops), 2013 IEEE 38th Conference on*. IEEE, 2013, pp. 874–881.

[31] H. Lee and A. Nakao, "Efficient user-assisted content distribution over information-centric network," in *NETWORKING 2012*. Springer, 2012, pp. 1–12.