UNIVERSITY OF CALIFORNIA
SANTA CRUZ

IMPROVING THE INTERNET ARCHITECTURE AT THE TRANSPORT LAYER

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

COMPUTER ENGINEERING

by

Abdulazaz Albalawi

March 2021

The Dissertation of Abdulazaz Albalawi
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Martine Schlag

Professor Brad Smith

Quentin Williams
Interim Vice Provost and Dean of Graduate Studies

Contents

List of Figures

Abstract

Improving The Internet Architecture at The Transport Layer

by

Abdulazaz Albalawi

Most existing reliable transport protocols in the current Internet architecture, like TCP, rely on an end-to-end connection as the main architectural construct to implement reliable transfer of data. This has become a big problem for content-oriented Internet applications because connections are brittle; in most protocols, the context exchange must be restarted if a connection is lost, a specific site must be selected to start the connection supporting content retrieval, and in-network caching cannot be used without relying on third parties applications such as CDN to replicate content-delivery functionality to only a subset of its users. As a result, several Information-Centric Networking (ICN) architectures have been developed that focus on information-centricity and eliminate connections between remote processes at the transport layer through modifications in the network infrastructure. However, this can only be achieved by requiring a far more complex network layer. In addition, transparent in-network caching is meant to be a vital benefit of these solutions, with the content being cached opportunistically throughout the network where it is needed. However, intermediate routers and caches have access to the content object being transferred to consumers, which poses privacy concerns.

To address these problems, this thesis proposes a new way of improving the Internet's architecture at the transport layer through a novel method of transporting data without using end-to-end connections and enables the use of transparent caching at the transport layer while preventing caches from decoding cached content.

The Internet Transport Protocol (ITP) is introduced as an alternative to the Transmission Control Protocol (TCP) for reliable end-to-end transport services in the IP Internet. The design of ITP is based on Walden's early work on host-host protocols, and the use of receiver-driven Interests and manifests advocated in several information-centric networking architectures. The design of ITP proves that connections or significant changes to the routing infrastructure are not needed to provide efficient and reliable data exchange between two remote processes on the Internet. Through minor extensions to the Domain Name System (DNS), ITP design can be leveraged to provide efficient content delivery by name over the existing IP Internet allowing it to achieve NDN advantages without the need for a new routing infrastructure. The use of manifests to describe contents globally at the transport layer eliminates the need for a new ICN like technology by allowing data to be cached at the transport layer and preventing these caches from understanding cached contents by enforcing decoding methods as part of the manifests.

To my hero, my number one fan, my father

Ali Awad Alhamodi

Acknowledgments

Writing this dissertation is the end of a journey. Throughout this journey, I have met many people; without their support, this thesis would not have existed.

I would first like to thank my advisor, Professor J. J. Garcia-Luna-Aceves, for his guidance throughout my grad school. I'm so grateful for your advice and the experience you gave me. I can't imagine accomplishing all this with a different advisor.

I also would like to extend my thanks to Prof. Brad for inspiring me and encouraging me to join the Computer Communication Research Group Lab, where I have met incredible friends, Chris, Ehsan, Nitin; many thanks to you.

Special thanks go to my managers and colleagues at Huawei, Cedric, Kiran, Hamed, and Asit. Thank you so much for the opportunity to work with you. I have learned so much working with you.

Finally, I cannot begin to express my thanks to my family for their support and encouraging words. Especially my loving parents, Munira and Ali, to whom I owe my success.

Chapter 1

Introduction

A transport protocol bridges the gap between the services available from a network infrastructure (e.g., a computer network or internetwork) and the facilities desired by application processes using the network infrastructure. As such, the design of any transport protocol is about interprocess communication [1, 2, 3]. The two main transport protocols used on the Internet today are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). UDP provides best-effort delivery between remote two processes, which is suitable for applications based on short transactions, and TCP provides reliable in-order delivery of data between two remote processes.

The current design of TCP dates back to the ground-breaking work by Cerf and Kahn on the *Transmission Control Program* [4]. In Cerf and Kahn's original design, Internet datagram delivery and end-to-end transport functionality were combined in a single protocol, and the use of end-to-end connections was not a necessity. Instead, *associations* between processes were discussed, which in principle could be instantiated with or without connections. Two processes are associated if they know of each other's existence and have a context to exchange data.

However, like most of the early work on reliable transport protocols [5, 2, 6], the subsequent design of the Transmission Control Program [7, 8, 9] used connections [10] to implement reliable in-order delivery of data between remote processes. Since then, even after the original design by Cerf and Kahn [4] was divided into the Internet Protocol (IP) [11, 12] and TCP [13], the conventional wisdom has been that end-to-end connections between remote processes are *necessary* to provide reliable and efficient communication between remote processes [2] without involving the underlying communication infrastructure for anything other than best-effort delivery of datagrams.

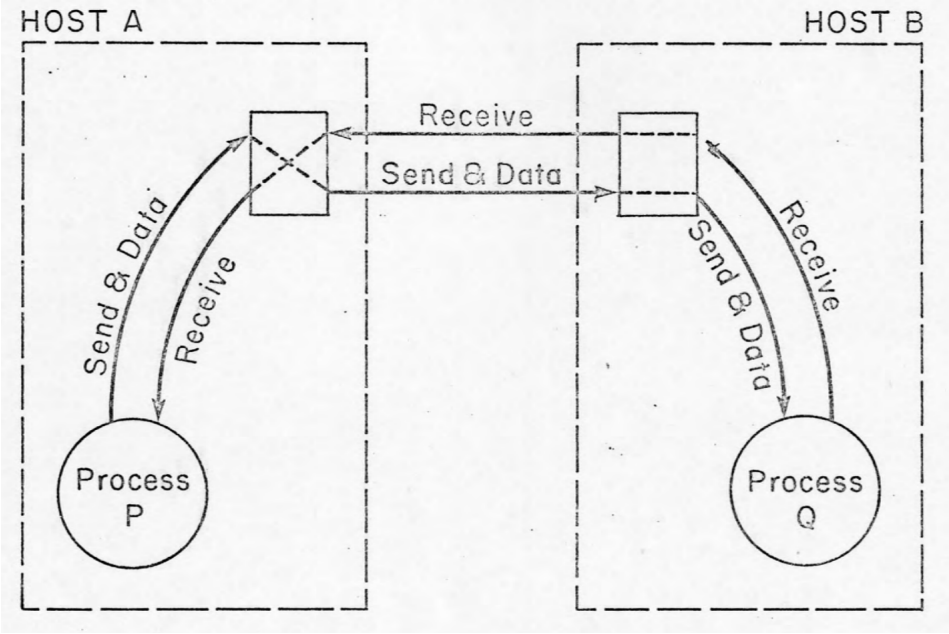


Figure 1.1: Walden's message-switching proposal for interprocess communication [15]

Interestingly, Walden [15], one of the designers of the host-to-host protocols of the ARPANET, proposed a message-switching alternative to the connection oriented ARPANET host-host protocol [5] based on his earlier work on a system for interprocess communication [3]. His approach was straightforward, as depicted

in Figure 1.1. In the figure process P has a message to send to process Q. Instead of setting up a connection, process Q sends a receive-message to process P, "the sender." When process P gets the receive-message from process Q and has data to send, it will send the data to process Q at Host B. When the message arrives at Host B, it will match a table entry left when the receive-message was sent by process Q. Walden already predicted the advantages of having a message-switching approach over a connection based one when it comes to interprocess communication. Such as minimal control messages needed to set up and terminate a connection. Even though Walden's proposal was discussed by Cerf and Kahn [4], connections became the norm for reliable communication between remote processes on the Internet, and message-oriented protocols like Walden's proposal have been viewed as inherently unreliable [10].

All current transport protocols on the Internet designed to provide reliable inter-process communication have been based on connections. This limits support for reliable and efficient access to content (e.g., videos and documents) and services (e.g., multiplayer games). Such limitation of the IP Internet communication protocols led to the development of dedicated systems aimed at content delivery. These systems can be categorized as content delivery networks (CDN) [16, 17] like Akamai [18] or peer-to-peer (P2P) applications [19] like BitTorrent [20]. While such content-delivery systems address many of the performance limitations of the basic Internet protocol stack, they require third parties to provide added functionality or applications to replicate content-delivery functionality to only a subset of its users.

Several Information-Centric Networking (ICN) architectures have been developed to address the limitations of the IP Internet and the inefficiencies of using CDN's and P2P applications. All of these architectures attempt to make Inter-

net content delivery more efficient, by allowing all Internet applications and users to obtain content and services by name, without requiring specific systems to serve different applications or forcing applications to determine where instances of content and services of interest are located on the Internet.

Name-based routing and forwarding have been considered to be necessary to attain an information-centric Internet. The most popular example of this approach today is the Named-Data Networking (NDN) architecture [23]. Packet forwarding in NDN is based on the names of content rather than the addresses where the content is stored, and transport-level connections are eliminated by increasing routers' forwarding state. Application processes ask for content by name using requests called Interests, and content is sent back over the reverse paths traversed by such Interests, which routers remember using Pending Interest Tables (PIT) in addition to the traditional forwarding tables.

Needless to say, NDN and other ICN architectures have produced valuable insight on what it means to provide information centricity at Internet scale. However, by requiring a far more complex network layer, these approaches have deviated from the end-to-end principle [21, 22] and the subsequent end-to-end arguments [14] that made systems built on datagram packet-switching networks so successful, including the Internet. Using name-based routing to attain an information-centric Internet is an architectural choice–not a requirement–that has profound implications. It means that efficient content delivery *must* come at the cost of redesigning the IP routing infrastructure and doing away with the DNS. Furthermore, servers and clients must also be modified to adopt the API's needed to use a name-based network layer, and widely successful application protocols like HTTP must be redone.

The motivation behind our thesis can be summarized in four points. First,

prior transport protocols leave applications in charge of reliable-communication functionality, rely on connections to provide end-to-end reliable communication, or require a network layer that provides much more than datagram delivery. Second, most ICN architectures attain efficient content distribution by: naming content and services rather than endpoints; using in-network caches to allow content to be delivered opportunistically from closer locations; and using either name-based routing and embedding name resolution in the routing infrastructure, or overlay networks of name resolvers or content routers. In addition, CDN and P2P approaches are insufficient to address the ever-increasing content-oriented usage of the Internet by all its users and applications. Third, prior ICN architectures do not provide a clear method of preventing caches or routers from accessing cached contents and ensuring consumer privacy. In addition, transparent caching on the Internet today is application-specific, and caches can access each request and response between the server and the client. Fourth, former end-to-end congestion-control algorithms proposed for ICN architectures are based on using round-trip time estimates as the primary indication for congestion in the network.

Chapter 2 provides a summary of related work in many different fields as well as a critique of prior work related to reliable communication between remote processes over computer networks as well as others. What is most notable is that no prior transport protocol has been proposed that provides reliable communication without end-to-end connections. Chapter 3, provides the design and implementation of ITP (Internet Transport Protocol), the first connection-free reliable transport protocol that operates over a datagram-based communication infrastructure. ITP is based on Walden's proposal [15], the content-centric approach to reliable data transfer first described Jacobson et al. [24], the use of manifest [25] and Interests advocated in a number of information-centric networking (ICN)

architectures.

Chapter 4, introduced the design of Named Data Transport (NDT), an alternative end-to-end approach to an information-centric Internet. NDT consists of the integration of three end-to-end architectural components: The first is a modification of the ITP protocol, we refer to as Named-Data Transport Protocol (NDTP); extensions to the Domain Name System (DNS) to include records containing manifests describing content; and transparent caches that track pending requests for content. NDT, TCP, and NDN's performance is compared using off-the-shelf implementations in the ns-3 simulator. The performance demonstrates that NDT outperforms TCP and is as efficient as NDN, but without making any changes to the existing Internet routing infrastructure.

Chapter 5, describes how to support transparent caching of content with privacy in ITP and NDT over the IP Internet using the manifest. This is done by relying on two methods (a) informational asymmetry; to prevent caches from accessing cached content and (b) computational asymmetry; to prevent clients from decoding unauthorized content.

Chapter 6, presents DCP, a delay based congestion protocol that can operate more efficiently than TCP based algorithms in architecture where end-to-end connections are replaced with Interest-based exchanges between a consumer and one or more other parties providing the requested content as in NDT and ICN. Finally, Chapter 7 provides our conclusions of this thesis.

Chapter 2

Related Work

In this chapter, we summarize prior work related to different fields of our thesis, including transport protocols, CDN and P2P approaches, ICN architectures, support for mobility and privacy, and end-to-end DNS-based schemes.

2.1 Reliable Transport Protocols

Considerable work has been reported over the years on transport protocols that provide reliable end-to-end communication [30, 31, 32, 33, 35, 36]. Although TCP [4, 7, 13] and most other reliable transport protocols use byte streams as the abstraction for retransmission and congestion-control, some notable transport protocols like the CYCLADES end-to-end protocol [6] have used packets rather than bytes for the same purpose.

A plethora of proposals have been made to improve the performance or functionality of TCP [31]. These proposals include more efficient flow control and retransmission strategies, the use of multiple connections, enabling multihoming of the remote processes exchanging data reliably [36, 35, 33, 32], and even the use of machine learning in TCP congestion control [37, 38, 39]. However, it is critical

to point out that all this prior work assumes that reliable communication between remote processes requires end-to-end connections.

What is so striking about all this prior work on reliable transport protocols operating over a datagram infrastructure is that all of them require the use of end-to-end connections to specific addresses. This has become a problem with the growing need to support location-independent Internet services and content. In addition, several results [40, 41, 42, 43] show that there is no connection-based protocol for reliable communication operating with a finite sequence-number space that works correctly under different conditions related to messages being deleted, duplicated, or reordered. It has also been shown that no correct connection-based protocol exists that provides reliable communication in the presence of nodes failing and losing their state [40, 41]. Hence, connection-based reliable transport protocols must operate with a relaxed definition of correctness that acknowledges that new connections are very likely to be initialized correctly.

Another important aspect regarding the evolution of transport protocols is their deployment [30]. Many novel solutions have not seen widespread deployment due to middleboxes (e.g., firewalls and NAT boxes) modifying or blocking any unfamiliar traffic [44, 49]. As a result, recent reliable transport protocols (e.g., QUIC [36]) adopt UDP for the naming of processes and framing of messages. Moreover, recent studies [45, 46] even suggest encapsulating TCP traffic inside UDP datagrams as it passes better through these middleboxes.

2.2 CDN and P2P Approaches

The limitations of using end-to-end connections between specific sites to support content delivery prompted the development of many CDN and P2P approaches over the years. Many surveys and taxonomies have been published for

CDN's and P2P applications and overlays (e.g., [16, 17, 19, 47]), and it should be clear that such systems provide much if not all of the functionalities and services present in *all* ICN architectures. For example, name-based content routing and redirection strategies were implemented in CDN's (e.g., [48, 50, 47, 51]) before they were advocated in ICN architectures.

The key difference between ICN architectures and CDN's or P2P applications is that they are applicable to *all* Internet users and applications, rather than just those users serviced by specific CDN third parties or specific P2P applications. It should be noted that the end-to-end nature of NDT enables the use of many CDN and P2P techniques to further improve its efficiency without requiring changes to the Internet routing infrastructure.

2.3 Prior ICN Architectures

TRIAD [52] and Content-Based Networking [26, 27, 28] were the earliest examples of ICN architectures advocating content-based routing by names or attributes and publish-subscribe operation. Many subsequent ICN architectures evolved over the years that either proposed to change the network layer of the Internet to implement location-independent name-based routing and forwarding, or adopted the use of overlays as in CDN's and P2P systems to provide name-based routing, resolve names to locations of content, and support publish-subscribe functionality. These ICN architectures include: COMET [53, 54], Content-Centric Networking (CCN) [24], CONVERGENCE [55], DONA [56], PSIRP and PURSUIT [57, 58, 61], MobilityFirst [62, 63], Named-Data Networking (NDN) [23], Nebula [29], 4WARD [60], SAIL [59, 65], and XIA [64].

The advantages and disadvantages of the ICN architectures are discussed in several surveys (e.g., [66, 67, 68, 70, 69]). Over the years, NDN has become the

most successful ICN architecture to date, and more recent work has even focused on embedding NDN routers as part of the IP Internet routing infrastructure [71, 72]. Accordingly, we use NDN as the leading example of prior ICN architectures.

The main benefits of NDN over the IP Internet architecture that have been stated in the past are: (a) Allowing all Internet applications to request content and services by name rather than locations, (b) eliminating performance issues associated with TCP connections, (c) providing added privacy to content consumers, (d) making efficient use of in-network caching for arbitrary content, and (e) enabling multicast services without the need for multicast routing protocols. The subsequent description of NDT and its comparison with NDN show that NDT provides all these benefits without the need to change the Internet routing infrastructure or establish new overlays on top of it.

In contrast to NDN and other ICN architectures, NDT does not require the deployment of overlays of name resolvers or content routers, or changes to the Internet routing infrastructure to operate. This approach is motivated by recent results on caching and ICN schemes. There is ample evidence that edge caching provides most of the benefits derived from in-network caching at every router [73, 74, 75]. On the other hand, the number of content name prefixes needed to provide name-based routing at Internet scale is many orders of magnitude larger than the number of Internet IP address ranges needed to support address-based routing. Hence, independently of the type of name-based content routing used [76, 77, 78, 79], the combined use of address-based routing and redirection schemes that map names to addresses can be done far more efficiently than the combined use of name-based routing and PIT's at each router. Performance results of recent ICN approaches based on addresses indicate that this is the case [80, 81, 82, 83, 84].

2.4 Privacy and Mobility Support

Transparent caching is meant to be a key solution to the rapid increase in network traffic in the Internet. Such a solution is meant to increase speed and reduce response time for end users. Prior approaches attempting to address the privacy concerns of transparent caching using end-to-end connections are not effective. Traffic carried over closed secure connections used in HTTPS cannot be cached, and this is a growing concern for content retrieval on the Internet because more than 50% of web traffic is served over HTTPS [85].

To overcome this issue, [86] [87] propose splitting a TLS(Transport Layer Security) session into two separate TLS sessions, one session between the consumer and the cache, on one side, and the cache and the producer on the other side. However, such an approach requires caches to be trusted using a certification authority. This introduces the issue where consumers cannot choose the level of security between the cache and the server. Other solutions [88] [89] , suggest altering the TLS layer to support trusted proxies operations on traffic flows. Although these solutions support features such as intrusion-detection or content-filtering they do not provide any support for transparent caching.

GroupSec [90] adapts HTTPS to support group memberships to allow transparent caching on the Internet without making any changes to caches or servers. Unfortunately, this approach does not provide privacy among members in the same group, and an adversary can infer whether multiple clients are in the same group.

Prior ICN architectures address privacy with different mechanisms [68], and Arianfar et al. [91] propose a privacy technique for NDN that leverages computational asymmetry to prevent caches from decoding cached contents in real-time; however, this method does not provide complete privacy for consumers.

The use of end-to-end connections between specific end-point addresses prompted the development of many approaches to cope with mobility in the context of the IP Internet [92, 93, 94, 95, 96, 97, 98, 99]. However, this prior work does not address eliminating connections for reliable end-to-end communication, and some can be used in NDT.

2.5 DNS-based Approaches

iDNS [100] and idICN [75] are arguably the first proposed approaches aiming to provide the ICN benefits in the current Internet by leveraging the DNS system. Instead of resolving a URL hostname to an IP address, iDNS resolves content names directly to metadata that contains the address of servers hosting the content, taking into account local caches; however, iDNS does not specify a specific protocol to retrieve contents using its method and instead leave it as part of their future work.

idICN also uses the DNS to resolve a content name to a nearby cache or a hosting server, and uses HTTP as the baseline for the transfer protocol. The limitations of this approach are that it is limited to HTTP, requires the connections used in TCP, and which must take place at the application level, causing different types of proxies to be used for caching. Similarly, DNS++ [101] and NEO [102] introduce an information-centric API combined with a DNS-based mechanism very similar to the one used in iDNS, and have the same limitation of requiring TCP and legacy application protocols to operate.

2.6 Congestion Control in ICN Architectures

Several congestion-control schemes have been proposed in the context of ICN architectures and NDN in particular. The novelty of the schemes is that the receiver is in charge of controlling retransmissions and managing congestion, given that their design is based on the exchange of Interests and responses sent to them. Some of these proposals depend on network assistance to detect and control congestion in the network, either by shaping Interests or using congestion signaling. Surprisingly, most of these schemes are based on using RTT estimates as the primary indication for congestion in the network.

ConTug [103] and ICP [104] are among the first proposed transport protocols for ICN's. They are window-based protocols that mimic the AIMD (additive increase multiplicative decrease) operation of TCP and assume that content chunks could be served from multiple sources. In ConTug, the retransmission timeout is set to the maximum RTT that is measured during a session. This approach can lead to a long timeout value for the Interest in a scenario in which a content chunk is served from the original producer or a remote cache, while the rest of the chunks are served from a nearby cache. If one of the chunks of the nearby caches is lost, the consumer has to wait for a long time before a retransmission timeout (RTO) is triggered and an Interest for the lost chunk is retransmitted. The same scenario can be applied to the way ICP sets the timeout value, because the timeout value is set to the mean of the maximum and minimum RTT that was measured throughout the session.

Most of the congestion control protocols proposed in the context of NDN follow a hop-by-hop approach taking advantage of NDN's stateful forwarding plane. Wang et al. [105] purpose a hop-by-hop Interest shaper for congestion control. Each router shape Interests going upstream in order to control the rate of re-

turning data thus avoiding congestion in downstream links. However, the author assumes that data and Interest size are fixed and link capacity is known. This approach is improved in [106] by incorporating congestion signaling using NACK packets. Recently, Schneider et al. [107] proposed PCON a congestion- control scheme for ICN's that performs congestion signaling to consumers by marking returning data packets. The protocol does not require routers to have knowledge about the link capacity and does not require Interest and data packet sizes to be fixed. The protocol detects congestion in the network by measuring packets queuing time over an interval and compares it to a target using CoDel AQM [108]. One of the main differences between PCON and other hop-by-hop protocols (e.g., the one in [105]) is that PCON does not reject Interests when it detects congestion but instead signals congestion back to consumers by explicitly marking returning data packets.

Chapter 3

ITP: Internet Transport Protocol

Based on the results discussed in Chapter 2, we designed the Internet Transport Protocol (ITP), a new transport protocol that is able to accommodate the rapid change in the current Internet usage. ITP demonstrates by example that, as Walden predicted back in 1975 [15], reliable and efficient communication between remote processes is possible without connections over a datagram-based communication infrastructure. That being said, the design of ITP is inspired by Walden's proposal on a message switching host-to-host protocol; and the introduction of Interests and manifests [25] in some ICN architectures, where a manifest is a data packet that describes the structure of content.

The use of manifest in ITP eliminates the end-to-end connection used in TCP. It allows the content consumer and content producer processes to share a common description of the structure of the content object that needs to be exchanged. It can also be leveraged to enable transparent caching of content with privacy as well. The idea of using manifests in content distribution is not new, and we are confident that manifests can be used effectively for content dissemination in a network. In fact, both CCNx and NDN have described the use of manifests in their designs. In a nutshell, a consumer application sending Interests to a content

provider first obtains a manifest of the content object being requested and uses the manifest to submit subsequent Interests regarding the content object. However, neither NDN nor CCNx take advantage of the fact that a manifest can be used to reduce or eliminate the ability of intermediate caches and routers to decode content objects being cached or disseminated, by stating a decoding method as part of the manifest as explained in Chapter 5.

3.1 Protocol Overview

The association between two remote processes using a manifest is established with a two-way handshake in which a process A requests some data from a remote process B by sending a data packet, and process B returns a manifest informing process A what Interests (requests) to send to obtain the requested data. Once process A has a manifest, it sends Interests to process B making reference to the manifest and informing B of what A needs. Process B simply responds to Interests from process A, without maintaining any state regarding process A. There is no need for signaling to start or terminate the association; process A can end the association when it obtains all it needs according to the manifest, and process B can simply end the association after a timeout.

3.1.1 Nexus: Implementing Associations without Connections

For two remote processes to communicate with one another reliably (i.e., be associated with each other), one must be able to address the other *and* they must understand what they communicate to each other. We can view these two requirements of an *association* as addressing and context. The current use of pairs

of IP addresses and ports satisfies the addressing requirement of an association. But what about the context?

If by necessity the communicating processes have no means of discerning the structure of the data they are exchanging, then they must agree on a method whereby they agree synchronously on the state of their exchange using a common representation for that state.

In this light, the evolution of TCP into a connection-based transport protocol makes sense given the state of computing technology back in the 1970s and 1980s when TCP was developed. Establishing a *connection* implements an association using a minimum of computing resources by establishing an agreement for the reliable exchange of a byte stream with an initial sequence number and provides enough context for processes to agree on how many bytes have been successfully delivered.

However, the price paid for such efficiency in resource utilization is big in terms of functionality. The context of the reliable communication between the remote processes must be established in real time, because there is no information regarding the structure of the data to be exchanged, and hence must also be updated and terminated jointly. Furthermore, a connection is ephemeral and the context of the association is lost if the connection is broken.

The cost of computing resources today is such that the basic premise of processes not knowing the structure of the data they exchange must be revisited. This is where the integration of manifests with a message-switching approach to reliable interprocess communication becomes so timely.

ITP overcomes the severe functionality limitations resulting from using connections to maintain the context of associations by using a *manifest* to maintain the context of associations. A manifest constitutes a common description of the

structure of the content to be shared between a content consumer and a producer. The description frees both processes from having to create and maintain an additional structure in real time to ensure that all the portions of the content are received correctly.

The manifest enables an "asynchronous" type of association between processes that is not lost if physical connectivity is lost. This is done based on two insights: (a) no correct connection-based protocol can be defined if delays are unbounded and nodes may lose connection state; and (b) all ICN architectures require the names of data objects to be permanent and uniquely associated with the objects they denote.

ITP takes advantage of the uniqueness and immutability of data objects' names to establish the needed context of an association independently of network delays or the occurrence of node failures, out-of-order datagram delivery, or packet replication. The manifest of a data object is itself a data object and specifies: (a) The unique immutable name of the data object, (b) the structure of the data object consisting of object chunks (OC) that can be sent in messages, and (c) the procedure that should be used to decode the data object from a set of OC's in case the data object is encoded. Additional metadata can be made part of the manifest of a data object, such as the names and size of object chunks and a list of IP addresses to contact to request them.

Provided that communicating processes can refer to the same manifest, they can exchange elements of the content described in the manifest on a transactional basis, and a consumer process is free to contact multiple processes using the same nexus provided by the manifest. In addition, a producer does not have to maintain state for each consumer by having the consumers inform the producer of their status in the nexus using the Interests.

We use the term **nexus** to denote the establishment of the context of an association between processes by means of manifests and references to them. Figure 3.1 illustrates how a producer transmits a data object D to a consumer in ITP based on a nexus. As in other modern transport protocols, ITP messages use UDP headers stating the address and port of the consumer and producer; this is indicated in the figure with "UDP." The *manifest* of a data object D is denoted by $M(D)$.

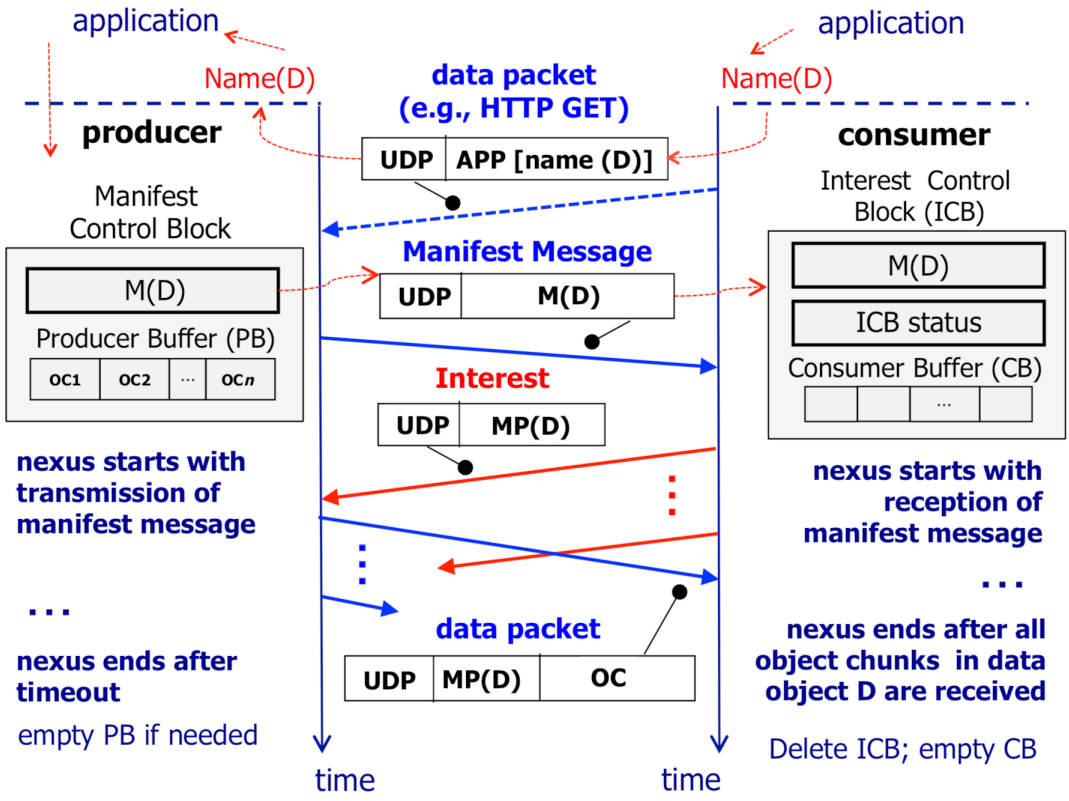


Figure 3.1: Nexus association in ITP

The application running at the consumer site asks ITP to send a data packet, which requests a data object (e.g., an HTTP GET). In response to that request, the producer specifies a number of parameters through a specific system call; the content to send, the IP and port address of the client, and additional control

information depending on the system call it used. Rather than setting up a connection to send the data, the ITP process constructs a manifest and sends it to the client as a data packet called *Manifest Message* in Figure 3.1, and creates a *Manifest Control Block* (MCB) that specifies the manifest $M(D)$, points to the memory location for D, and includes a *Producer Buffer* if additional memory space must be allocated for efficiency. The OC's of D are stored in a Content Store (CS) at the producer, and can be copied to the producer buffer to reduce latencies.

The nexus for D at the producer starts when it creates the MCB for D and ends after a *nexus timeout* that must be long enough to allow consumer(s) to obtain the OC's in D, without having to reallocate memory for the Producer Buffer. The nexus timeout is restarted each time the producer sends a data packet with an OC of D.

The nexus for D at the consumer starts when it receives the manifest $M(D)$. After receiving $M(D)$, a consumer creates an *Interest Control Block* (ICB) for D and allocates memory for a *Consumer Buffer* (CB) to store OC's of D. The ICB includes $M(D)$ and the ICB status indicating the OC's that have been received and those that are missing. The nexus at the consumer ends when it has all the OC's needed for D, at which point it deletes the ICB for D. Once the consumer has a nexus for D, it obtains the data in D by sending *Interests* for D. An Interest for D is denoted by $I(D)$ and states: (a) the names of the consumer and producer using their IP address and port number; and (b) a manifest pointer ($MP(D)$) that references $M(D)$ using the content object immutable name. The $MP(D)$ also states implicitly or explicitly the OC's that the consumer is missing and those that it has obtained by stating the names of the OC's it needs. The $MP(D)$ can be thought of as a transport-level cookie since it frees the producer from maintaining a per-consumer state. This is similar to the use of cookies at the

application layer in order to eliminate the need for servers to maintain a per-client state. Finally, the producer responds to an Interest $I(D)$ with one or multiple data packets. Each data packet contains the manifest pointer $MP(D)$ of the Interest that prompted it and OC's that are part of D.

3.2 Protocol Specification

3.2.1 Software Architecture

As Figure 3.2 shows, the ITP layer consists of five data structures: Producer, Consumer, Manifest Control Block (MCB) list, Interest control block (ICB) list, and Content Store. All entities in ITP will have the same structure, making a bi-directional messaging protocol.

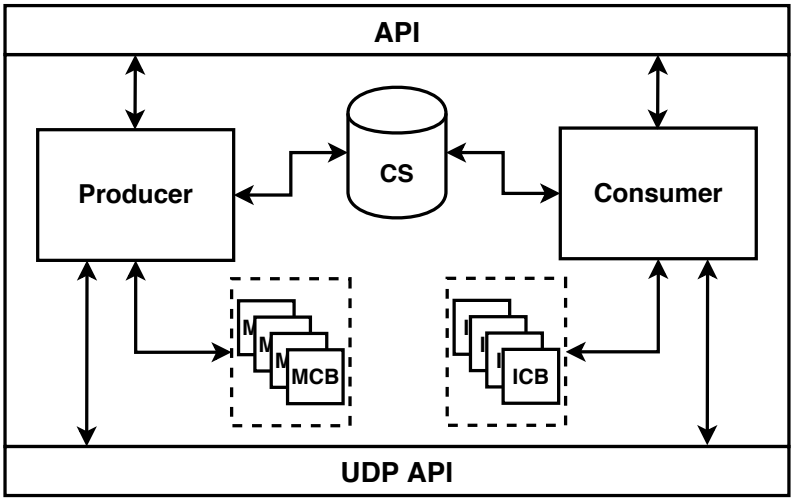


Figure 3.2: Software architecture of ITP

ITP messages are encapsulated in UDP packets, which are further encapsulated in IP datagrams. Encapsulating packets in UDP ensures that no alterations are needed for the network stack, and minimal changes are required for the application, since most network devices widely support processing UDP packets. This

is similar to existing transport protocols, such as [34],[36]. As a result, ITP can operate completely in user-space, allowing us to rapidly develop and experiment with the protocol without any change to the transport layer. This implementation makes ITP interfaces on one side to user or application processes and the other side to the UDP interface. The UDP socket is created once the application process creates an ITP socket. Eventually, the UDP calls on the Internet module to transmit each segment passed by the ITP layer.

Producer & Consumer

An ITP producer is responsible for sending a content object, and an ITP consumer is responsible for consuming the object. Going back to our previous example in Figure 3.1, When the server application sends out its reply, it is the ITP producer's responsibility to construct the manifest for this data and send it to the ITP consumer at the other end. It is also its responsibility to send requested data packets in response to received Interests. The ITP consumer is responsible for retrieving the data using Interests based on the information provided by the manifest. It is important to highlight that the same occurs when the client application sends out a request (e.g., an HTTP GET). However, applications that consume data are naturally different from applications that produce data. Therefore, we specified different system calls to send the data over ITP that fits the application need. For example, when the client sends a request for a specific content, it is done through a different system call than the one used by the server application to send a content to a client. This system call triggers the ITP producer at the client side to deliver the data packet that encapsulates the content request instead of constructing a manifest and sending it to the ITP consumer at the server end.

Manifest Control Block

The ITP producer needs to remember several variables for each manifest it creates. The producer remembers these variables in a data structure known as the *Manifest Control Block* or MCB, where some of these variables represent the manifest itself such as manifest timeout and list of authorized consumers to retrieve the content. All MCB's are stored in a list that can be accessed using the manifest pointer carried in ITP Interests and data packets. The MCB is similar to the Transmission Control Block (TCB) used in TCP to maintain information about a connection. However, the MCB is only used to maintain a minimal state about each manifest constructed while the ITP consumer carries most of the heavy lifting by remembering many variables about the connection and exchanging them back with the ITP producer using the manifest pointer.

Interest Control Block

The ITP consumer remembers more variables for each content object that needs to retrieve. These variables are also stored in a data structure known as *Interest Control Block* or ICB. As with the MCB, all these structures are stored in a list. A consumer creates an ICB for each new manifest it receives. The ICB includes variables such as manifest name, list of ITP producers to contact, Interest timeout and so forth. Once all the Interests for a content object are satisfied, the ICB tigers the ITP consumer to deliver the content to the application.

Content Store

The Content Store (CS) in ITP resembles the Content Store used in NDN and the send buffer used in TCP; where unacknowledged TCP segments are maintained in case of retransmission timeouts. However, TCP cannot reuse a packet

after it has been acknowledged by the other end. While an ITP producer can make use of object chunks (OC) in the CS to satisfy Interests from different consumers, depending on the application need. The CS in ITP is also used to buffer OC's waiting to be requested by an ITP consumer similar to a regular receive buffer in other transport protocols. A content object that is being retrieved is buffered at the CS until all its OC's are received and then it is delivered to the application by the ITP consumer. The decision of when to deliver the data to the application is issued by its ICB as mentioned before. The CS stores the OC's based on their name in the data packet which is specified in the manifest.

3.2.2 Data Naming

Each data object in ITP carries a name, including the manifest. The manifest's name is mapped to a specification of content to be sent. Similar to NDN and CCNx, ITP assumes a hierarchically structured name that is based on URI formatting[109]. NDN and CCN use hierarchical names to allow for a scalable name-based routing, but ITP can simply rely on the IP forwarding plane as it does not require any routing infrastructure changes. However, using a hierarchical structure when naming content can have other advantages by allowing applications to represent relationships between data pieces. For example, when segmenting a content into multiple object chunks (OC's) or representing nested manifests, such as when a single manifest is not enough to represent a large content.

There are multiple approaches to name OC's, one simple approach to name OC's in ITP is by using sequencing with the content name from the manifest. Using this method, an ITP consumer appends a chunk number to the outgoing Interest's content name and keeps incriminating it until it receives all the data packets corresponding to the content object. The same method can also be used

when retrieving dynamically generated data. Other deterministic algorithms can also be used by sharing them with the consumer as part of the manifest.

An alternative way to name OC's in ITP is by using a cryptographically secure hash function of their content, similar to the one proposed for NDN [110]. The hash digest for OC's in a manifest can be thought of as the sequence number in TCP used to identify bytes' streams. This means that a manifest is a content object with a hash digest as its name that describes an ordered collection of OC's and their corresponding hash digest. The combination of the OC's stated in the manifest and carried out according to the method indicated in the manifest renders the original CO. Large CO's can be organized into a hierarchy of decoding manifests, such as hierarchical manifests proposed in CCNx [25]. To allow transparent caching for data objects in ITP, all data objects names are prefixed with the ITP producer's IP address and application's port number. This way, caches can simply use the IP address and port number in the name when forwarding Interests in a cache miss without relying on information outside the ITP packet, such as the UDP and IP header, as this information can change when packets are intercepted.

3.2.3 Packet Types and Formats

Because ITP identifies its data using names, as in most ICN architectures, not all its header fields are predefined and static like in most IP protocols such as TCP and UDP. Like other proposed packet formats in CCNx & NDN [111] [23], some ITP header fields are encoded using the Type-Length-Value fields (TLV) format. This allows ITP to provide a flexible packet structure that can support ITP's main functionality. Figure 3.3 illustrates the general ITP packet format containing fixed-size fields followed by a manifest pointer field that includes a variable-length

content object's name and name chunk offset value, and then optional TLV fields. The TLV format used in ITP is based on a 2Bytes Type and 2Bytes Length encoding, and the Length field represents the size of the Value field in bytes.

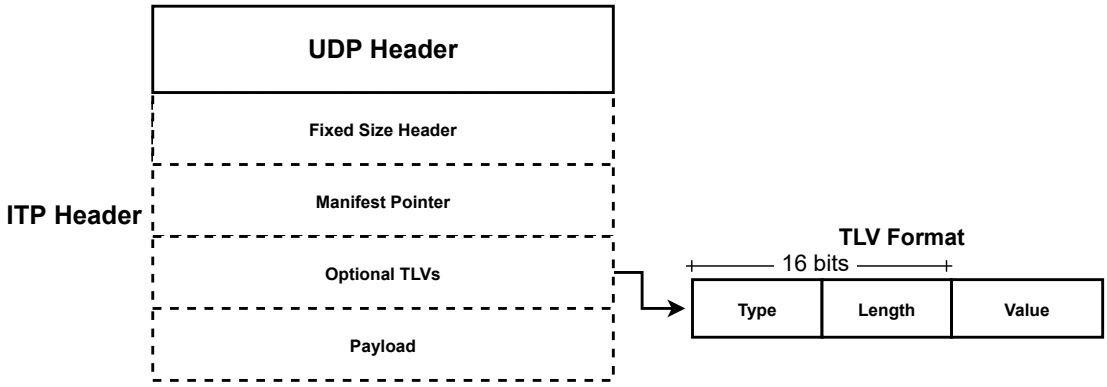


Figure 3.3: ITP packet general format

ITP packet's format follows a similar format used for CCNx packets[111], considering both share some of the design principles when it comes to data naming. ITP has three types of packets; manifest, Interest, and data packets. All three are distinguished using a 1Byte control field, as shown in the packet header in Figure 3.4. The version number allows for different versions for ITP to be used, and it is specified by 1Byte. The packet length field is determined by 2Bytes, including the packet header size.

The structure of the packet in ITP varies based on the packet type. For example, data packets will usually carry a payload in them, making them relatively larger than Interests and manifests as they do not hold any payload. However, every packet in ITP must have a name that addresses the content object of interest. Like the CCNx packet header format, the content object names in ITP are placed in a fixed location in the packet header. The 2Byte Content Name Length field specifies the variable length of the content object name. Right after this field comes the Name Chunk Offset TLV which is used to specify the chunk name

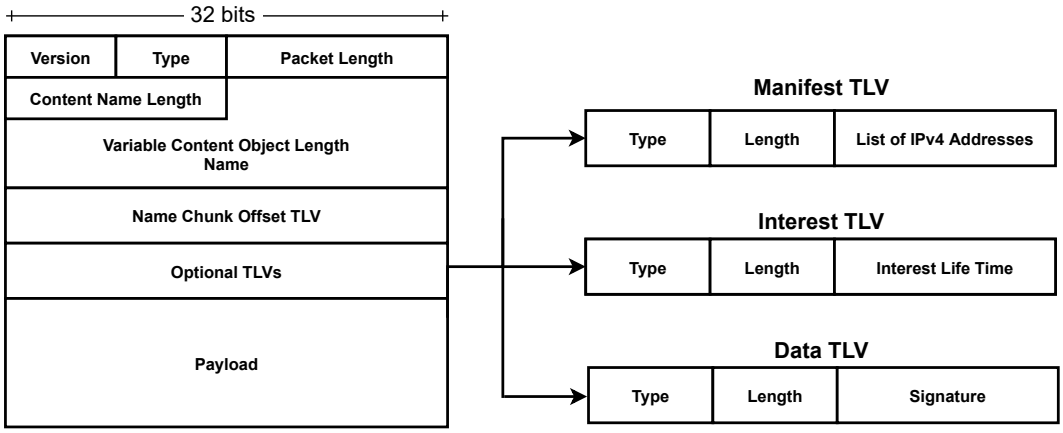


Figure 3.4: ITP packet layout

offset that is then applied to the content object name to retrieve the chunk names to be requested. The length field of this TLV specifies the size of the offset value field. Each 2Bytes of the value field indicates the offset of the chunk names' starting point inside the content object name field.

The consumer uses the manifest pointer in ITP packets to reference the manifest used to describe the content object of Interest to the producer. It informs the producer of the consumer nexus status without maintaining a per consumer state. However, there is no single field that specifies the manifest pointer in the ITP packet; instead, it is represented by multiple fields, including the content object name field and the chunk name offset TLV field. Using the content object name in the Interest allows the producer to traverse the Manifest Control Block list to find the corresponding MCB for that content. Using the chunk name offset TLV field, the producer knows which chunks are missing from the consumer side on which then it will send it to the consumer. The same goes for when a consumer receives a data packet. It will find the corresponding ICB for this content based on the content object name field on which then it will process the data packet accordingly.

Table 3.1 shows a list of different TLV types that can be used in ITP packets. Different packet types can only use certain TLV types. For example, a manifest may have an additional TLV that lists multiple servers to contact other than the original. Also, data packets can carry a signature by using the TLV signature. Other optional TLVs can also be defined and used in ITP packets to support more future features.

Type	**Description**
Name Chunk Offset	Offset of the chunk name in the content name field
Selector	List of IP addresses of other producers to contact
Signature	Producer's signature for the data payload
Timestamp	Timestamp of when a packet is sent
⋮	⋮

Table 3.1: ITP TLV types

3.2.4 Reliability

The ITP consumer side enforces reliability for most of the packets exchanged in a nexus, except for the manifest, which is enforced by the ITP producer. The ITP producer achieves reliability for the manifest by keeping a list of authorized ITP consumers to receive a particular manifest. An ITP producer acknowledges the arrival of a specific manifest to an ITP consumer by receiving an Interest with a name that maps to the manifest name. Certain system calls can also enforce the ITP producer to deliver specific data packets reliably without constructing a manifest. Such as the one used by the client's application to request content from a server.

Once the ITP consumer receives a manifest, the reliability to retrieve the data object is shifted to the consumer side. The ITP consumer keeps a state of

unsatisfied manifests in the ICB list. A manifest is satisfied and removed with its ICB from this list once a consumer satisfies all the Interests related to this manifest. Once that is done, the consumer invokes the application to deliver the data, as explained previously. An Interest is acknowledged by the arrival of a data packet that corresponds to the Interest.

3.2.5 Application Programming Interface

The interaction between transport protocols and applications can differ from one protocol to another. For example, the TCP API calls `bind()`, `listen()` and `accept()` are specific for server sockets and `connect()` is specific for client socket, while `send()` and `recv()` are common for both types. The communication loop between the `send()` and `recv()` calls in the TCP server, and the TCP client specifies the dialog between the client application and the server application. It depends on the application developer on how the dialog goes. For example, the default HTTP/1.0 application dialog will receive a request and then send a response for that request. After this, the HTTP server will close the client socket, known as Short-lived connections in the HTTP world. Another application would be to transfer a large file and the dialog for example would be to send the file into chunks. After all the chunks are sent, the server will close the client socket.

Figure 3.5 illustrates an example of a client/server relationship using the primary socket API functions and methods in ITP. The naming of these system calls is inspired by the early work by Walden on host-host protocols [3]. We can see these calls are somewhat similar to the TCP/UDP socket API, but they differ significantly. The current TCP/IP socket API dates back to the 1980s with the release of what is called Berkeley sockets. Most of these socket calls prevent the programmers from understanding what goes on at the transport layer; instead,

these calls just produce numeric error codes that usually have a generic meaning. Therefore, a future goal in the design of ITP API is to provide a platform that simplifies the programming of today's applications while hiding the complexity of the communication calls — allowing application developers to customize their content distribution and have full control over deployment decisions.

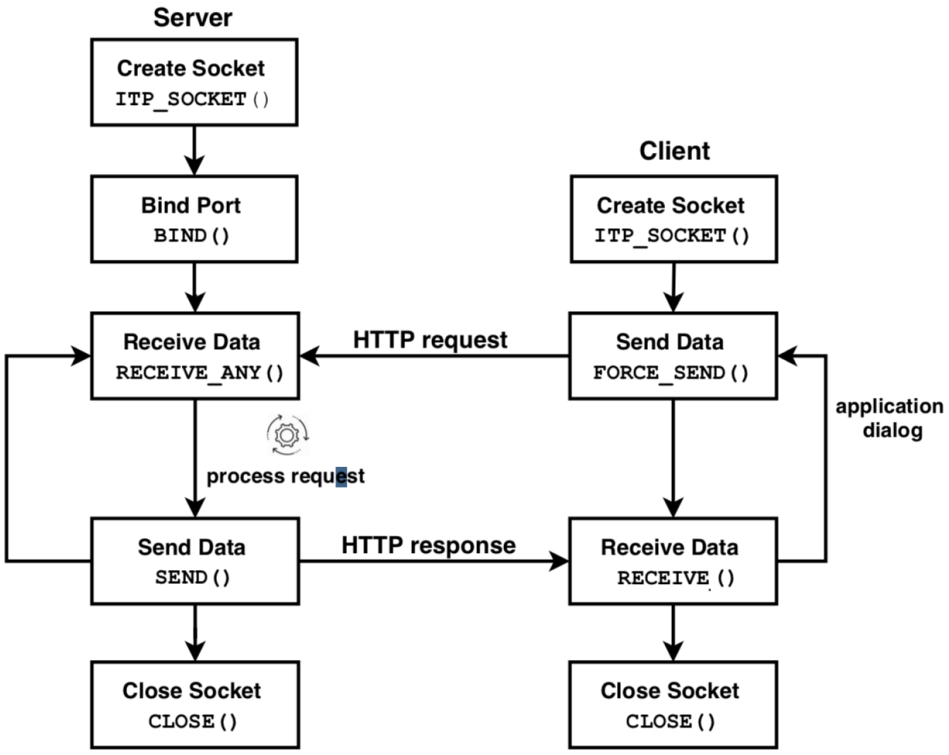


Figure 3.5: client and server application communicating using ITP socket

As Figure 3.5 illustrates, no connection is established from the client to the server like in TCP, and instead, the client just sends messages to the server using the `FORCE_SEND()` call. This is the same system call that was used by the client to send its HTTP GET request in Figure 3.1. This system call will force the ITP producer at the client-side to send the message directly to the server without constructing a manifest. Also, the server in ITP does not need to accept a connection; and instead, it just waits for messages to arrive. When a message

arrives at the server, it contains the address (IP, Port) of the sender, which then the server can use to reply back to the client through the system call `SEND()`. It is up to the application developer to decide if clearing the buffer is done through timeout or by the recipient of a specific control message from the consumer after retrieving the content. ITP is considered bi-directional, but it is not fully duplex the same way as in TCP since there is no notion of connection. However, it is up to the **application dialog** as highlighted in Figure 3.5 to handle this. Because sockets by themselves are fully duplexed, an application can simply send back to the port of origin, as we mentioned before. Like TCP, an ITP server application can close its socket after the dialog ends. However, since there is no notion of a connection between the two ends, a server can only close its socket.

3.3 Retransmission Strategy and Congestion Control

ITP uses a receiver-driven selective-repeat retransmission strategy inspired by the work by Jacobson et al. [24] and based on the fact that the consumer is in charge of the management of the nexus with the producer. The consumer in ITP controls the flow of data traffic by controlling its Interests' sending rate and allows for a window of data packets to flow to the receiver in response to its Interests. The manifest pointer included in each Interest tells the producer what portions of the content have been received and which ones are missing, which serves as a selective acknowledgment used in many TCP variants. The rest of the section explains the consumer's action and the producer in ITP when dealing with retransmission of loss packets and congestion in the network.

3.3.1 Retransmission Strategy

Consumer Steps: The consumer keeps track of the nexus status in its Interest Control Block (ICB) to ensure that a data object is passed to the application correctly. The initial status of the ICB is set with all OC's missing when the consumer sends its first Interest to the producer for a data object D, and the ICB status is updated with each OC received correctly from the producer. Depending on the application using ITP, the consumer may have to obtain all the OC's of a data object before it passes them to the application process and then delete its ICB. Alternatively, the consumer may be allowed to pass to the application OC's that are in order and without any missing OC's and delete those OC's from its ICB.

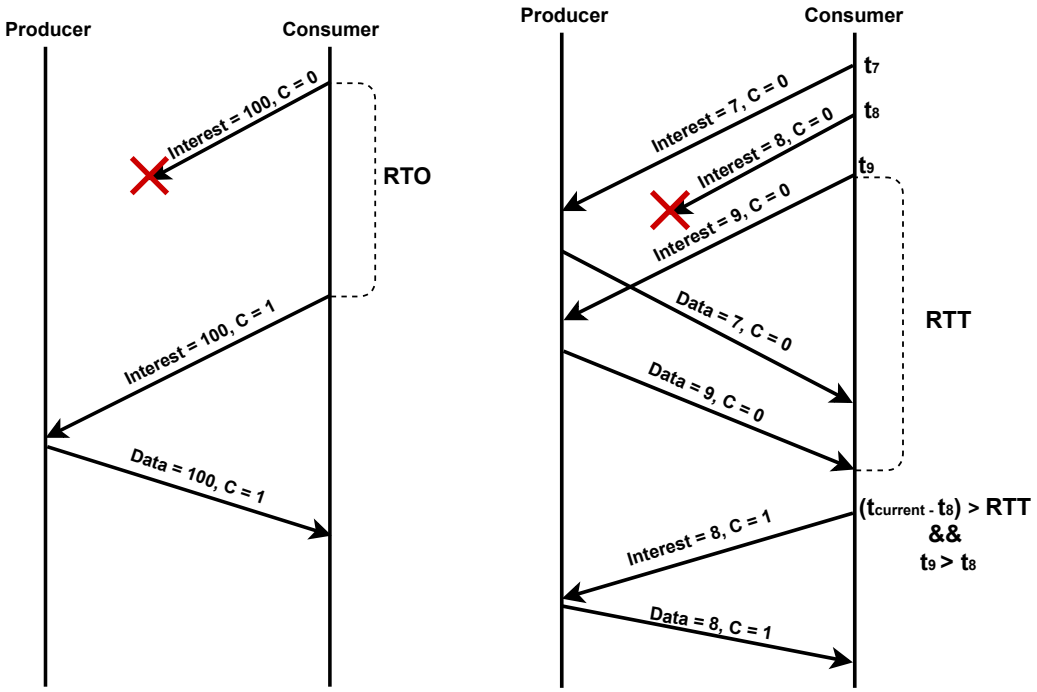
The consumer in ITP is in charge of retransmissions and simply persists in sending Interests to the producer, asking for the OC's needed to decode the data object. Each Interest includes a manifest pointer that reflects the ICB status's latest snapshot for this particular content. As such, each Interest can be viewed as including a vector of acknowledgments informing the producer about the OC's that have been received correctly and those that are missing. The consumer maintains a list of Interests based on the local times when they were transmitted. Each Interest's manifest pointer includes the local transmission time or a sequence number that differentiates it from any other Interest, even when multiple Interests carry manifest pointers stating the same data object. Hence, each data packet received can be matched uniquely with a transmitted Interest and the consumer can accurately update its round-trip time (RTT) estimate with every data packet it receives, even when data packets and Interests are lost or delivered out of order. To facilitate efficient retransmissions, the consumer applies a retransmission timeout (RTO) for each Interest it sends to the producer; the RTO is updated

based on Jacobson's algorithm [112]. This allows ITP to provide better RTO estimates by measuring the correct RTT with each data packet. The consumer retransmits an Interest i originally sent at time t_i in two cases: (a) after the RTO for Interest i expires, or (b) when a data packet for an Interest j sent at time $t_j > t_i$ arrives at time t_d and $t_d - t_i >$ RTT. The latter allows for fast retransmits without incurring unnecessary Interest retransmissions. Figure 3.6 shows the retransmission strategy for each case.

It's important to highlight that ITP does not need to rely on such mechanisms as Fast Recovery in TCP NewReno to detect multiple packet losses within a single window. It's well known that TCP Reno suffers under scenarios with more than one packet loss within a single window, because it can only detect that using a timeout. However, TCP NewReno recovers from this issue by using partial ACK's (ACK received during Fast Recovery that advance the Acknowledgment Number field of the TCP header, but do not take the sender out of Fast Recovery) as evidence to retransmit later lost packets, and also continue using the Fast Recovery process. As a result, TCP NewReno is constrained to retransmit at most one dropped packet per round-trip time. In addition, the use of the SACK option in TCP allows the sender to know which packets were received during Fast Recovery instead of relying on duplicate ACK. However, if a retransmitted packet is lost, the SACK will only detect the packet drop using the retransmission timeout, which is not the case in ITP since it is driven by the receiver.

Producer Steps: A producer simply responds to Interests from any consumer regarding a data object D using the information about D stored in its MCB and the manifest pointer included in each Interest. The manifest pointer carried in an Interest identifies the consumer and the OC's that it needs. The order in which a producer receives Interests is not critical, and the occurrence of Interest

losses or Interest duplicates does not confuse a producer, because each Interest carries a manifest pointer.



(a) Retransmission due RTO

(b) Retransmission due to out-of-order arrival of Interest

Figure 3.6: ITP retransmission strategy in action

3.3.2 Congestion Control

We describe a congestion-control strategy for ITP that attempts to mimic the approach used in TCP Reno, but takes advantage of the receiver-driven retransmission strategy using manifest pointers in Interests and data packets. Using a congestion-control strategy that mimics TCP Reno allows us to illustrate the inherent benefits of the connection-free receiver-initiated approach used in ITP compared to the connection-based sender-initiated approach used in TCP.

We adopt a simple Interest-based approach for congestion control in which

one Interest from the consumer elicits one data packet from the producer. This is the original approach advocated in CCN [24] and NDN [23]. However, more sophisticated approaches could and should be implemented in which a window of packets or packets with multiple OC's are sent in response to Interests.

Producer Steps: Data packets are the main cause of congestion in ITP, because the size of an Interest is relatively small compared to a data packet. The use of receiver-driven Interests allows the data traffic in ITP to be controlled by controlling the rate at which the consumer issues Interests. This frees the producer from having to maintain any per-consumer congestion state. Accordingly, the producer's role in congestion control is minimum and consists of simply submitting requested OC's upon reception of Interests from consumers.

Consumer Steps: The consumer controls the flow of data traffic with the producer by controlling its Interests' sending rate using a congestion window. Algorithm 1 shows the consumer's congestion control algorithm in ITP. The Interest congestion window, Icwnd, defines the maximum number of outstanding Interests allowed to send without receiving their data packets. The window size is adjusted based on the AIMD (Additive Increase Multiplicative Decrease) mechanism commonly used in TCP for the congestion window. Like TCP Reno, the consumer in ITP starts in slow-start with a congestion window of size one. The congestion window size value is increased for each data packet received. This continues until either a loss is detected or the congestion window reaches the slow-start threshold, ssthresh. Once the consumer exceeds the ssthresh, it enters the congestion avoidance state as in TCP [112]. The consumer increases its congestion window by one Interest every round-trip time during this state.

When an Interest times out, the consumer retransmits the Interest, reduces its congestion window to one, and sets the ssthresh to half the congestion window size

before the timeout; it then goes into slow-start. If it detects a packet loss using fast retransmit, the consumer reduces its congestion window by one half and sets the ssthresh to the new window size causing it to go into congestion avoidance.

Algorithm 1 ITP Consumer: Congestion Control Algorithm

```
1: Initially:
2:     Icwnd = 1
3:     ssthresh = INFINITY
4:
5: On Data:
6:     if (Icwnd < ssthresh)
7:         /*slow start*/
8:         Icwnd = Icwnd + 1
9:     else
10:        /*congestion avoidance*/
11:        Icwnd = Icwnd + 1/Icwnd
12:
13: Fast Retransmit:
14:        Icwnd = Icwnd/2
15:        ssthresh = Icwnd
16:
17: Timeout:
18:        ssthresh = Icwnd/2
19:        Icwnd = 1
```

3.3.3 Content Multihoming

To understand how the congestion-control mechanism in ITP can be used when data are retrieved from multiple sources, we need to address three cases in which data can be retrieved from various sources. When a consumer retrieves OC's for the first time, some of the OC's may be dropped due to congestion. If these chunks are cached on the way before they get dropped, then they will be retrieved from the cache instead of the producer for retransmitted Interests. If two consumers with different RTT share the same path to the producer, then some OC's for one of the consumers may be retrieved from the cache instead of the producer. Lastly, as Chapter 5, an ITP producer may obfuscate data by encoding the data with old OC's, and some of these old OC's may be cached and retrieved from caching proxies while other OC's are retrieved from the producer.

Relying on measurements of a single round-trip time (RTT) in NDN as it is done in TCP can give the wrong indications of congestion in the network. As a

result, many of the congestion control protocols proposed for NDN [113] argue against using a single RTO (retransmission timeout) to detect packet loss because it does not span across multiple data sources. This is because NDN cannot see when data is being retrieved from different sources. ITP uses a single RTO in both single-source and multi-source cases along a path. ITP detects when data is being retrieved from a close cache or the original producer based on the packets' source address. The consumer maintains a list of the sources used to satisfy an Interest, only the source with the most satisfied Interests is flagged as the primary source for the consumer. Then, the consumer will only use the RTT estimate for the primary source in the RTO estimate. Whenever a new primary source is detected, the consumer resets the RTO estimate based on the new primary source's RTT values. ITP keeps an RTO for each Interest. When an RTO event is triggered in ITP, the consumer reduces its congestion window to 1 just like in TCP New Reno.

3.4 Transparent Caching

Transparent caching in the current Internet requires caches to be aware they are receiving applications specific requests transparent to the clients. These caches are required to receive redirected client requests possibly by intercepting TCP connections destined for specific ports or for a specific set of destination addresses. One example of a widely used transparent caching in today's Internet, is transparent web caching, where web traffic is intercepted and redirected toward a web cache server without requiring any configuration at the client. This is usually done by using a layer 4 switch on the route between the client and the origin server. Such an approach is widely used in different US cell carriers to enhance mobile network performance [114].

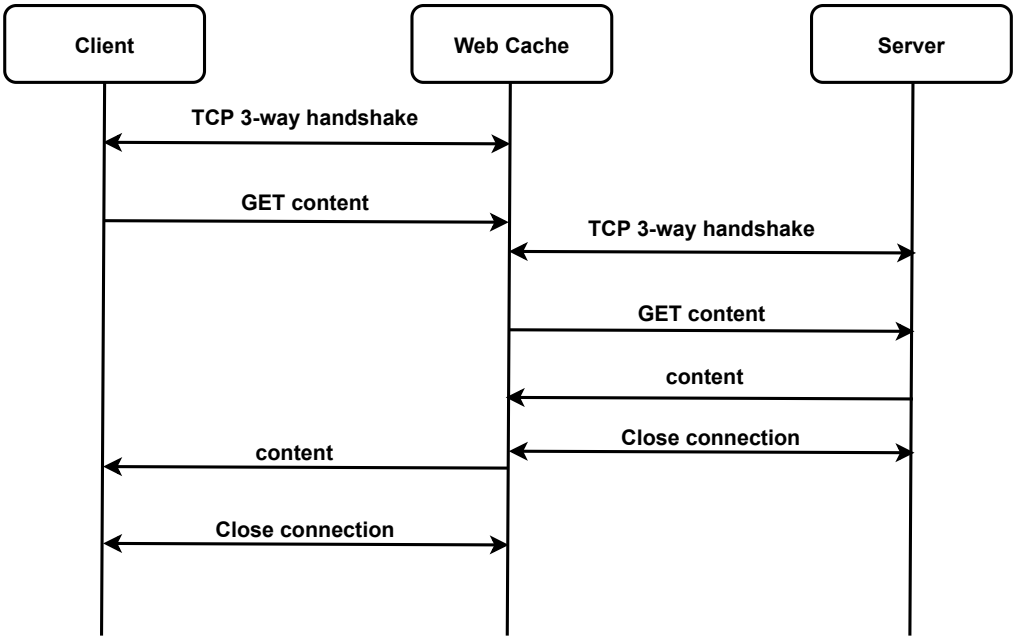


Figure 3.7: Web caching in action

Figure 3.7 shows an example of transparent web caching, where a user sends a request for a web page to an HTTP server. One of the web caches on the way intercepts the request, and checks if it has the requested web page. If the web cache does not have the requested web page, it will establish a separate connection to the server and resend the client's request to the original web server. Once it receives the response for this request it will send it back to the client after caching the response for future requests assuming it is cacheable. This could be repeated along the path to the original server, where different web caches intercept an earlier cache miss.

By necessity, web caches serve clients' requests by transparently splitting the TCP connection between the client and the original server. Specifically, the web cache splits a TCP connection into two connections, one between the client and the web cache and the other one between the web cache and the web server. The web cache terminates the client's TCP connection by spoofing as the web server

— also, the web cache spoofs as the client when connecting with the server. By splitting the TCP connection, the web cache can have access to each request and response between the server and the client. Since all this happens transparently to both of them, this poses a significant privacy concern. The only way for the client and the server to impose privacy is to set up a private connection using a protocol like TLS. However, such an approach precludes caching in middleboxes.

3.4.1 Transparent Caching in ITP

NDN eliminates connections between remote processes at the transport layer through modifications in the network infrastructure. However, it is important to note that NDN does not avoid the use of virtual connections altogether, because routers must maintain per-Interest state in their PITs in order to forward data packets in response to Interests. The per-Interest state maintained by the routers between two communicating processes (a client and a content provider) constitute a per-Interest virtual circuit. Even though application traffic running over ITP is cached at the transport layer, they do not require their caches to maintain a per-Interest state as in NDN. In addition, ITP allows application traffic running over ITP to be cached at the transport layer without the caching logic leaking to the application layer. This means that network administrators can simply install a single ITP proxy cache in their network and configure a layer-four switch to redirect ITP traffic to ITP caches. Future work entails extending the API for this caching entity to support network features such as load balancing, filtering, or QoS policy for different applications.

The following sequence of ITP Proxy steps for transparent caching of ITP traffic corresponds to the numbers shown in Figure 3.8.

1. A client process sends a data packet, which requests a data object (e.g., an

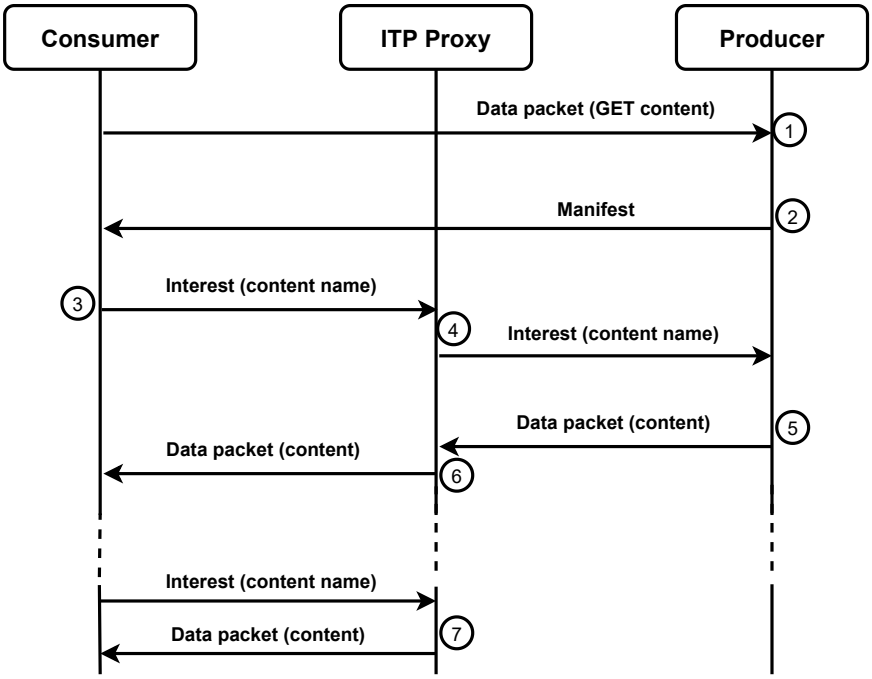


Figure 3.8: Transparent caching for ITP traffic

HTTP GET). This request is sent over the system call `FORCE_SEND()` which causes it to be sent as a data packet without the need for a manifest. The middle ITP Proxy on the way is configured to only intercept Interests destined to a list of ITP producers and data packets from these ITP producers. It is up to the administrator to cache manifests sent by the ITP producer but there is no meaning for it.

2. After the ITP producer at the server processes the incoming data packet it will deliver it to the server application along with the client IP address and port number. Once the application processes the message it will trigger a reply back to the client with the proper response using the system call `SEND()`. This causes the ITP producer to construct a manifest and send it to the ITP consumer at the client side.

3. After processing the manifest, the ITP consumer sends an Interest to the

ITP producer at the other end.

4. The ITP Proxy on the way intercepts the Interest then checks if it has the requested data packet in its content store. Since it does not, it forwards it to the ITP producer using the Interest name. [1]

5. After processing the Interest, the ITP producer responds with the requested data packet.

6. The ITP Proxy intercepts the data packet and caches it in its Content Store if it does not exist to satisfy incoming future Interests. It then forwards it to the ITP consumer, which after processing it, will deliver it to the application layer.

7. Finally, the Interests from a new ITP consumer (who has the same manifest) will be satisfied by the ITP Proxy instead of going all the way to the ITP producer.

As illustrated in the example, every Interest in ITP is an atomic operation. When an ITP cache cannot satisfy an Interest, it simply forwards it to the ITP producer based on the IP addresses appended in the Interest using the TLV fields in the packet as explained previously. As a result, ITP can provide transparent caching at the transport layer for all data carried over it. This is different from today's caches, where they rely on the application in order to specify the server, by using fields like complete URL as in HTTP.

Even though ITP provides transparent caching at the transport layer, forwarding Interests in ITP will simply rely on datagram routing without requiring any changes to the routing infrastructure as in most ICN architectures. In addition,

[1]For simplicity we assume these switches preserve the original packet header which can be done by using a protocol data unit (PDUs) to provide metadata for these packets to the ITP proxy just how some of the HTTP proxy providers define their standards.

ITP caches do not need to keep track of pending Interests as it is done in NDN. However, keeping track of Interests in a table as in NDN can have multiple advantages such as preventing sending duplicate Interests over the same path. However, one of ITP's main objectives is to minimize the state kept at the servers and them same should also be applied to caching proxies in ITP. In the next chapter, we present NDT, an end-to-end approach that leverages the design of ITP to provide efficient content delivery by name over the existing IP Internet through minor extensions to the Domain Name System (DNS). Similar to ITP, NDT is able to provide transparent caching at the transport layer. However, NDT proxies use Pending Interest Table (PIT) to track pending Interests at the transport layer. The aggregation of pending Interests at NDT proxies prevent caches from sending duplicate packets over the same path, allowing it to provide native multicast support but is based on IP addresses.

3.5 Evaluation and Performance Comparison

We evaluate the performance of ITP, TCP and NDN using the ns3 [115] and ndnSIM [116] simulators, and consider the efficacy of congestion control methods, the efficiency of transparent caching, fairness, and TCP friendliness.

3.5.1 Efficacy of Congestion Control

This experiment illustrates the inherent benefits of using a receiver-driven connection-free reliable transport protocol based on manifests instead of a connection-based transport protocol like TCP. We compared the congestion-control and retransmission mechanisms of ITP and TCP assuming a simple network scenario with a single source and a single sink as depicted in Figure 3.9. The consumer

in ITP can only issue Interests for the content after requesting the manifest from the producer, where the client in TCP consumes traffic generated by the server after establishing a connection with it using the TCP three-way handshake. The size of the object chunks in ITP is equal to the segment size in TCP and is fixed at 1500 bytes, and both ITP and TCP share the same fixed-header size.

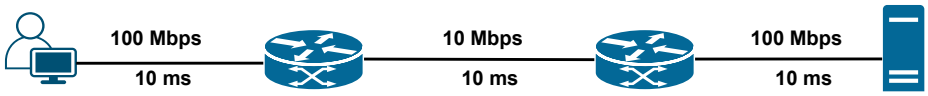
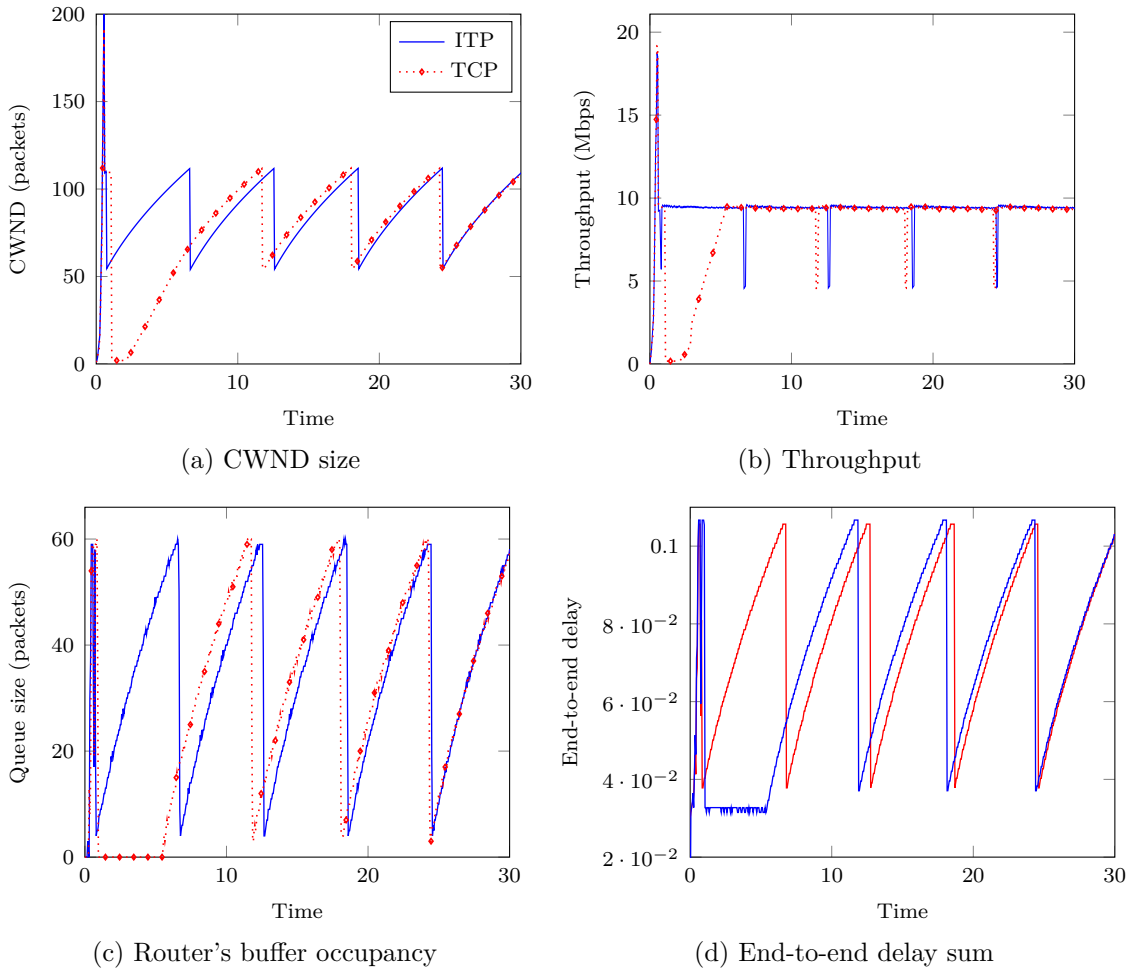


Figure 3.9: Baseline topology

Figure 3.10a shows the evolution of the congestion window for both protocols during the first $30s$ of downloading a content of $3.5Mbyte$ in size, a total of 24632 chunks/segments. The retransmission policy in ITP allows receivers to detect and recover from a packet loss faster than in TCP, where it took the client a total of around $34s$ to download the file compared to the total download time of a total around $31s$ in ITP. This is due to the fact that ITP does not use connections and applies a fast retransmission strategy enabled by manifests. A consumer in ITP has a complete picture of which OC's were received correctly and which were lost and does not rely on partial ACK's like TCP does. Accordingly, the consumer immediately goes into congestion avoidance state instead of fast recovery. As a result, ITP continues increasing its congestion window normally. This allows ITP to use the bottleneck's buffer more efficiently compared to TCP, which is forced into fast recovery, during which the sender can only transmit new data for every duplicate ACK received.

Figure 3.10c shows the bottleneck's buffer's queue size. It can be seen from the figure that TCP has a longer idle period compared to ITP due to packet timeout. As a result, ITP achieved higher average throughput due to better utilization of the link's capacity.



(a) CWND size

(b) Throughput

(c) Router's buffer occupancy

(d) End-to-end delay sum

Figure 3.10: Single-flow scenario for TCP and ITP

Table 3.2 shows the average simulation results for TCP and ITP for the same scenario with few changes to the topology. The client requests/receives data for the simulation duration, which is 30s. The client's session's start times are randomly chosen between 1s and 5s. The simulation was repeated 10 times, and results were measured at the client-side. The simulation time does not take into account the TCP handshake used to close a connection. As shown from the table, the consumer in ITP achieves higher throughput than TCP, resulting in more bytes received. This is mainly due to ITP's connectionless nature, which gives it

the ability to detect and react to congestion faster than TCP, resulting in fewer packet losses.

Protocol	Throughput (Mbps)	Bytes (Mbyte)	Packet Loss
ITP	9.020282 ±0.01438	31.85985 ±1.84591	115.5 ±0.52705
TCP	8.110141 ±0.05913	28.83405 ±1.75277	220 ±0.00000

Table 3.2: Single-flow results

3.5.2 Efficiency of Transparent Caching

We compared the total time taken to retrieve multiple copies of a large data file using ITP, TCP, and NDN. The experiment assumes a network consisting of a source node connected over a 10 Mbps shared link to a cluster of 10 sink nodes, all interconnected via 100 Mbps links, where a middle node in this scenario is acting as a caching proxy for ITP traffic. The same topology was used for NDN as well. For a fair comparison between NDN and the other protocols, we used the same transport protocol highlighted in the previous scenario. We ran ten scenarios; with each scenario, we increased the number of sinks in the network, bringing the total to 10 sinks. Each sink starts pulling a 6MB data file from the source at random start time based on a Poisson distribution with an average arrival of 5 minutes. We ran each scenario ten times, each one with a different random arrival time.

The total elapsed time for all the sinks to complete the task was recorded and displayed in Figure 3.11. With only a single sink, both TCP, ITP, and NDN perform the same, given that most of ITP's and NDN requests were retrieved from the source, and all three approaches use a similar algorithm for congestion

control. As the number of sinks increases, the completion time in ITP and NDN remains fairly constant for all ten scenarios, while the completion time in TCP increases linearly because all the data have to be retrieved from the source. NDN outperforms ITP when two or more downloads start before data are available at the nearby cache. In ITP this results in those Interests being sent to the producer while in NDN only the first Interest is sent.

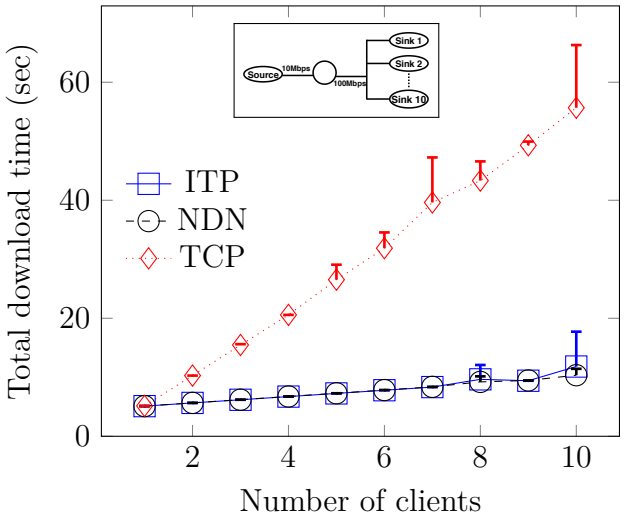


Figure 3.11: Total transfer time vs. number of sinks

This scenario highlights the ability of ITP to take advantage of transparent caching without requiring any changes to the communication infrastructure.

3.5.3 Fairness

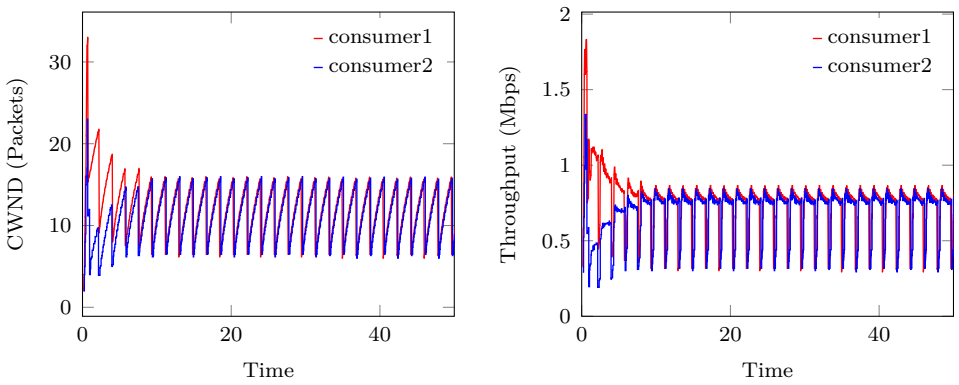
We evaluate ITP's fairness under a multiple-flow scenario. The topology consists of two consumers and two producers connected via a bottleneck link with 1Mbps capacity. The queue size is set to 20 packets and the file size is 10 MB. Both producers transmit the manifests for the file at the beginning of the simulation at the same time. Both consumers start issuing Interests to retrieve the data from the producer after receiving the manifests. We used Jain's fairness index F

as our performance metric for this scenario which is defined as

$$F = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{3.1}$$

where x_i is the throughput for the ith connection, and n is the number of users sharing the same bottleneck resource. The fairness index F is bounded between $1/N$ and 1, where 1 corresponds to the case in which all N flows have a fair allocation of the bandwidth (best case), and $1/N$ refers to the case in which all the bandwidth is given to only one user (worst case).

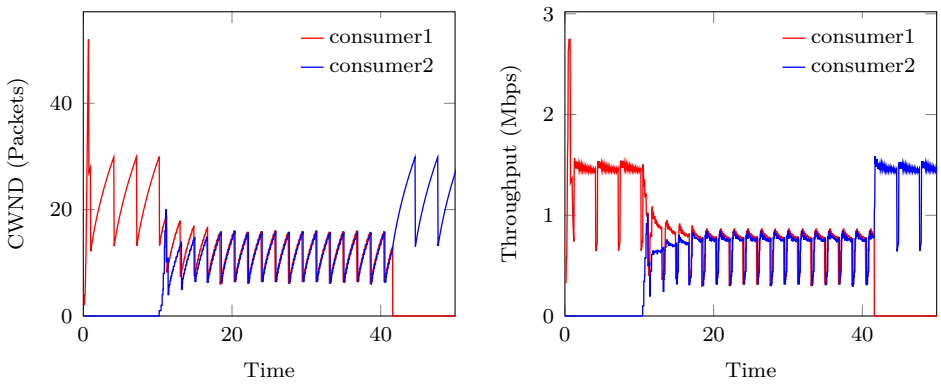
Figure 3.12a shows the evolution of CWND for both consumers. As seen from the figure, ITP CWND behavior follows the usual TCP sawtooth behavior since both are based on AIMD congestion control algorithms. The fairness between the two flows is $F = 0.99$; this can also be seen from Figure 3.12b, where both consumers achieve a similar average throughput.



(a) CWND Size (b) Throughput

Figure 3.12: Multiple-flows with homogeneous Start

We also, evaluated the case of multiple ITP flows under different initial start times. Figure 3.13a shows the evolution of the CWND for both consumers. consumer 1 starts requesting the data at the beginning of the simulation, and once



(a) CWND Size (b) Throughput

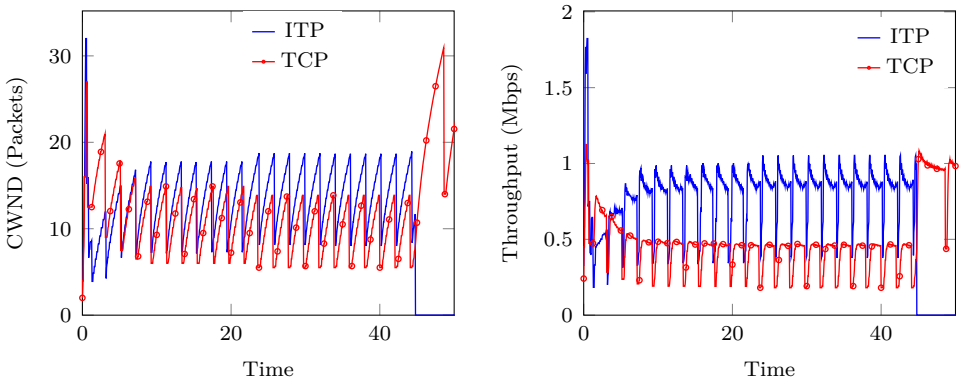
Figure 3.13: Multiple flows with heterogeneous start

it fully utilizes the link capacity, consumer 2 joins the network and starts sending Interest to its associated producer at the other end. This is approximately 10 seconds after the beginning of the simulation. As seen from the figure, once consumer 2 joins the network, it causes a buffer overflow, which results in a packet loss for both consumers, who decrease their sending rate by halving their congestion window and then going into congestion avoidance. Both consumers achieve a similar sending rate until consumer 1 leaves the network, and consumer 2 utilizes the remaining link capacity, as Figure 3.13b shows.

3.5.4 TCP Friendliness

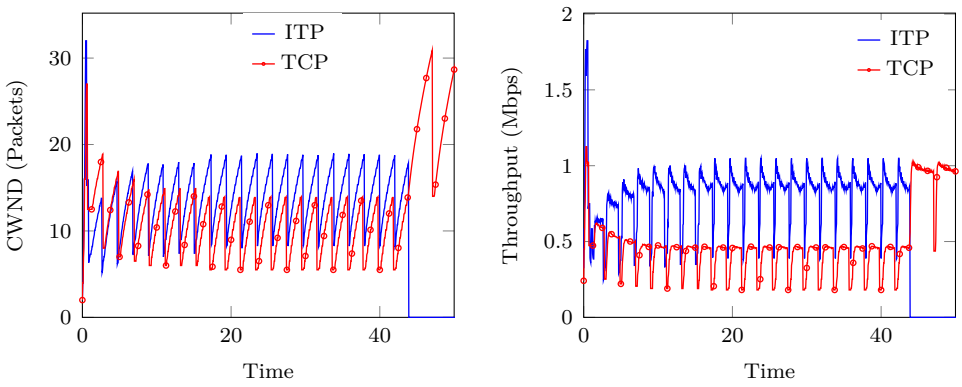
To examine TCP friendliness in ITP, we considered a scenario in which caching takes place and one without caching.

The topology of the network consists of a bottleneck link of capacity 1 Mbps and a buffer size of 20 packets. For the sake of simplicity, the chunk size of ITP is fixed at 1000 bytes, and the same goes for the segment size of TCP. TCP operates with the SACK option enabled, and ACK's are not delayed. Both TCP and ITP have the same round-trip-time delay, and they are retrieving the same file of



(a) CWND Size (b) Throughput

Figure 3.14: TCP friendliness without caching



(a) CWND Size (b) Throughput

Figure 3.15: TCP friendliness with caching

around 3MB. For the scenario with cashing, ITP caching proxies are configured in the topology before the bottleneck. This allows the ITP caching proxy to serve the consumer's Interests for dropped data packets due to congestion along the bottleneck. Initially, the ITP caching proxy is empty and caches any data packets that pass through it. A router is configured to interrupt ITP packets based on the protocol number and redirect them to the ITP caching proxy.

Figures 3.14a-3.14b show the results without caching. Using Jain's fairness index, the fairness between the two flows is equal $F = 0.9988$, which is under-

standable since ITP also follows an AIMD congestion control algorithm like TCP. The same goes for the scenario where caching takes place, as seen in Figures 3.15a-3.15b. The results illustrate that caching does not cause a high decrease in fairness; in fact, fairness between the two flows was equal to $F = 0.9967$. This is due to the ability of ITP to detect that most packets were retrieved from the primary source, and therefore control its sending rate accordingly. Even though ITP achieved less fairness than two competing TCP flows under the same scenario (where fairness equal $F = 0.999996$), the total download time for TCP flows retrieving the same file size as TCP and ITP was higher by 8.5%.

3.6 Conclusion

In this chapter we introduced ITP, the first connection-free reliable transport protocol. Its design consists of the integration of a message-switching approach first discussed by Walden [3, 15] with the use of manifests [25] and receiver-driven Interests [23]. ITP eliminates the need for servers and caching proxies to maintain per-client state and allows for all application data to be cached on the way to consumers using ITP caching proxies that can act as middle boxes. To prevent these middle boxes from accessing cached content, Chapter 6 explores some of the ways to protect the data using the manifest and only consumers who have access to the manifest will have access to the data. We outlined what a manifest is and how it is used to establish a nexus between remote processes, and it is clear to us that connections can be eliminated as the way to instantiate the context in associations. The next chapter explains how the connection-less nature of ITP can be improved to support location-independent Internet services and content resulting in a content-centric IP Internet.

Chapter 4

NDT: Named Data Transport

In the current TCP/IP-based network architecture, data resources are transmitted based on end-to-end connections. Many variants of TCP have been proposed over the years, and several transport protocols have been proposed to improve on its performance [32, 33, 36, 35]. However, all transport protocols designed to provide reliable inter-process communication have been based on connections as we showed in Chapter 2. This has become a problem with the growing need to support location-independent Internet services and content. In most protocols the context exchange must be restarted if a connection is lost, a specific site must be selected to start the connection supporting content retrieval, and in-network caching cannot be used without compromising privacy. To some extent, this reliance on connections prompted the recent development of several information-centric networking (ICN) architectures [69, 70, 69]. All of these architectures attempt to make Internet content delivery more efficient, and the most prominent of them has been Named Data Networking (NDN).

In Chapter 3, we introduced ITP, the first connection-free reliable transport protocol that enables the use of transparent caching at the transport layer for all applications carried over ITP. In this chapter, we introduce Named-Data Trans-

port (NDT), an alternative end-to-end approach to an information-centric Internet that leverages the novel design of ITP. NDT supports the emerging Internet content-oriented applications of the current IP Internet architecture that is free of connections and without modifications to the network architecture as in prior ICN architectures.

Conceptually, NDT amounts to an end-to-end implementation of the Interest-based approach first introduced by Jacobson et al. [24] that does not require any changes to the Internet routing infrastructure and takes advantage of the DNS. NDT allows applications to ask for content and services by name, and makes the transport layer responsible for: (a) mapping content names to one or multiple locations where the content is offered, (b) delivering the content reliably to consumers from content servers or transparent caches without using end-to-end connections, and (c) enforcing the privacy of consumers. Like other ICN architectures, NDT can provide the benefits attained with CDN's and P2P applications, but without requiring the services of specific vendors or replicating functionalities at the application layer. Furthermore, NDT can be deployed incrementally and much faster than prior ICN architectures.

4.1 Overview

NDT consists of the integration of three end-to-end architectural components: the first one is Named Data Transport Protocol, or NDTP, which is a transport protocol that is inspired by the design of ITP. However, NDTP is designed with an extended API and more features to support the key goals of ICN without requiring a modification to the network infrastructure as most ICN architectures. Similar to ITP, NDTP eliminates the need to maintain the ephemeral type of context provided using connections by allowing a consumer and producer to share

a common description of the structure of the content being exchanged, and such that both can refer to that description to deliver specific portions of content reliably. The manifest of a content object frees the NDTP consumer and producer of the object from having to create and maintain the context for their reliable exchange of the object in real-time.

The second architectural component is the manifest-yielding DNS (my-DNS), which mapps content names into manifests describing their content. The my-DNS augments the DNS to maintain manifest records describing the locations and structures of content objects. Given that NDTP consumers obtain the location of the nearest copy of a content object given its name, my-DNS serves as a de facto name-based routing overlay with redirection operating on a publish-subscribe basis. Using my-DNS redirection eliminates the need for name-based routing protocols while incurring very small additional delays.

The final NDT architectural component is the NDT Proxies (NP), which provide transparent caches that track pending requests for content. NDT Proxies support multicasting services without the need for IP multicast routing by maintaining a Pending Interest Tables (PIT) similar to those used in NDN and other ICN architectures, but without making any changes to the IP Internet routing infrastructure. In the next chapter we will describe how ITP and NDT proxies can be used to: (a) support transparent caching of content with privacy over the IP Internet for arbitrary applications, and (b) secure content relying on informational asymmetry to prevent caches from accessing cached content and on computational asymmetry to prevent clients from decoding unauthorized content.

4.2 Named Data Transport Protocol

NDTP provides reliable end-to-end communication over datagram communication without establishing connections, it uses the manifest as ITP does to describe the structure of the content. The association between processes in NDTP is built around content names instead of IP addresses and ports numbers as in ITP. As stated previously, the association between any two processes requires two requirements; addressing and context. In most protocols including ITP the addressing part can be satisfied using pairs of IP addresses and ports numbers. With the help of DNS, the association between processes in NDTP is built around content names. In NDT, applications have access to content names in the same way clients have access to Uniform Resource Locators (URLs) today. Applications request content by name and NDTP provides reliable name-based transport services by obtaining the structure of content objects from the DNS without the need for end-to-end connections and in a way that is transparent to end-user applications.

Provided that an NDTP consumer and a producer can refer to the same manifest, they can exchange any portion of the content object described in the manifest on a transnational basis, and an NDTP consumer process is also free to contact multiple parties hosting the content using the same manifest published by the NDTP producer of that content. More specifically, the NDTP process managing application content publishes manifests accessible on the IP Internet that describe how the content can be retrieved. The server application servicing the content notifies its NDTP producer to publish a Manifest Record (MR) that maps to a unique global name and describes how the content object is structured. Before the NDTP process servicing the client application starts retrieving the content object, it queries its local my-DNS (manifest-yielding DNS) to resolve the object name to its MR, which contains the manifest of a content object and additional

information to manage the content on the IP Internet. Once the consumer obtains the MR for the content object, it proceeds with a window-based sequence of Interests requesting the chunks that are needed to decode the content object, as stated in the MR.

Similar to ITP, NDTP is implemented over UDP, as a result, NDTP operates completely in user-space, and its packets are encapsulated in UDP datagrams, which are encapsulated further in IP datagrams. NDTP is implemented using two main data structures: A producer that publishes content and a consumer that retrieves content. This design is inspired by the NDN API design [117].

4.2.1 Publishing Content

Figure 4.1 shows an overview of the NDTP producer functional structure. When a server application needs to publish content, it is the responsibility of the NDTP producer to publish the content on the Internet. This consists of three main parts: (1) saving the content object and its name into its content store, (2) sending requested data packets in response to received Interests, and (3) publishing a MR for the content object by registering it along with its name with its authoritative my-DNS server.

Before publishing content on the Internet, the producer segments the content into multiple chunks and encodes them in a specific way to ensure the privacy of cached content. An example of an encoding method is explained in Chapter 5. Chunks can also be signed and encrypted to ensure security at this stage. Once the content is segmented into multiple chunks, it is cached at the content store. The content store can be viewed as the sender buffer in TCP and other connection-based transport protocols. The producer then appends the names of the chunks into the manifest along with the encoding method and other security

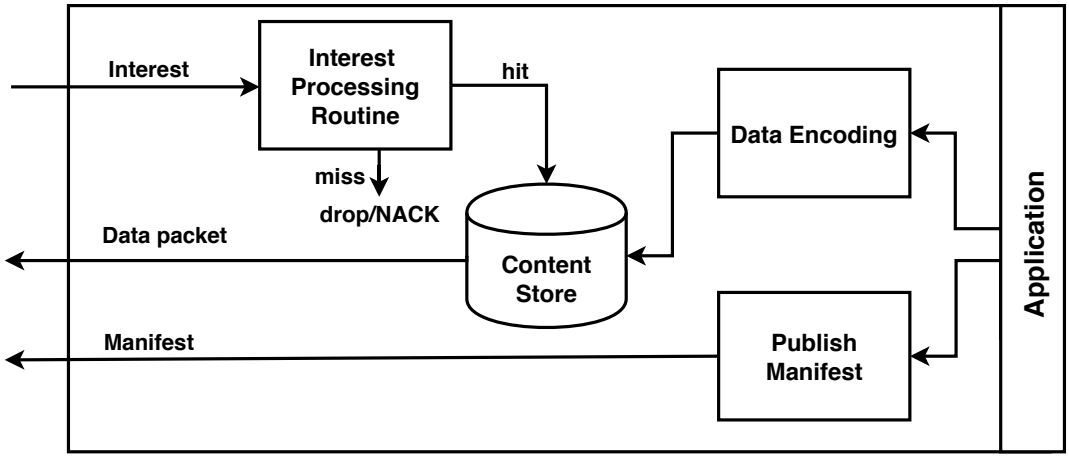


Figure 4.1: Processing Interests and data packets at an NDTP producer

parameters. The final stage of publishing content on the Internet is by publishing its manifest. This is done by constructing the MR using the manifest itself and meta-information about the content (e.g., a time to live, list of servers to contact, etc.). The producer then registers the MR along with the content name with a my-DNS authoritative server. Registering and updating MRs is done using regular DNS standards [118], and it is up to the content provider to determine which sites to use to host content objects.

An Interest from a consumer goes through the Interest processing routine, which is used to perform a cache lookup at the content store. The Interest is simply dropped if it's a miss, and a data packet is sent back if it succeeds. A NACK can be sent back to the consumer if necessary. The design of NDTP allows application developers to customize their content distribution and have full control over deployment decisions.

4.2.2 Retrieving Content

Figure 4.2 shows the functional structure of an NDTP consumer. Client applications running over NDTP retrieve data objects using their names (URL). It

used to be that applications that rely on URLs in their interaction will need to query the DNS server first to translate the hostname part of the URL into its IP address before they start their interaction. However, in NDTP, the client only needs to provide the content-object name to the NDTP layer. The NDTP consumer does all the work in retrieving the content, which involves contacting the local my-DNS to retrieve the MR, and requesting the actual content and decoding it. Using the NDTP API, clients access the content directly through the function `GetContentByName()`, which takes the content name as its parameter. Calling this function invokes the consumer side of the NDTP layer.

The NDTP consumer responsibility can be broken into two main tasks: resolving content names to their MR, and retrieving the content using information from the MR. As Figure 4.2 shows, the consumer in NDTP remembers a set of variables for each content that needs to be retrieved. These variables are stored in a data structure called the Content Control Block or CCB, which is used to control such things as Interest timeouts, window size, the decoding method, and list of IP addresses to contact. All Interests go through the Interest crypto routine. The routine signs these Interests based on information from the manifest. Arriving data packets will go through the Data verification routine for authentication including manifest as well as checking if packets are corrupted.

4.2.3 Retransmission and Congestion Control

Retransmission and congestion control algorithms in NDTP are receiver-driven, just as in NDN and ITP [24]. These algorithms are based on the inclusion of a manifest pointer in each Interest and data packet. Clearly, many receiver-driven strategies can be implemented. For simplicity, however, we chose to use the ITP congestion-control and retransmission algorithms explained in Chapter 3

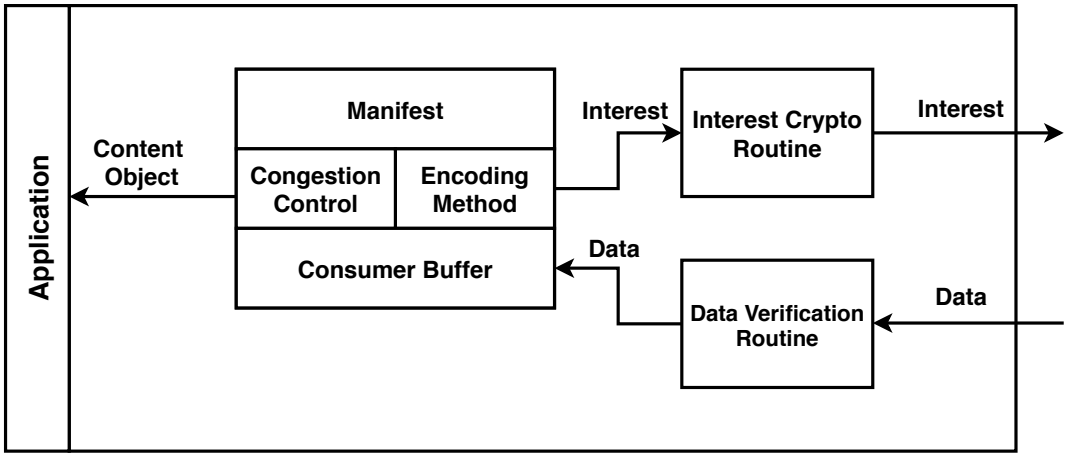


Figure 4.2: Processing Interests and data packets at an NDTP consumer

in NDTP. Given the similarities of the mechanisms used in NDTP and ITP, the rest of this section provides only an outline of the retransmission and congestion-control mechanisms in NDTP.

Congestion Control

The NDTP consumer controls the flow of data traffic by controlling the sending rate of its Interests using a congestion window. The congestion window specifies the number of outstanding Interests allowed to be sent before receiving their data packets. The window size is adjusted based on the AIMD (Additive Increase Multiplicative Decrease) mechanism commonly used in TCP for the congestion window. The NDTP consumer increases its congestion window based on slow-start, starting with transmitting one Interest and increasing the congestion window by one for each newly received Data packet. The slow-start continues until the window reaches the slow-start threshold. The consumer operates under the congestion-avoidance state as in TCP [112] When the slow-start threshold is exceeded, and increases its window by one Interest every round-trip time.

Fast Retransmit

NDTP uses a receiver-driven selective-repeat retransmission strategy similar to the one used in ITP. An NDTP consumer retransmits a lost Interest once an out-of-order data packet is received based on the order in the transmitted list if a time constraint is met. A lost Interest y initially transmitted at time t_i is retransmitted once the following constraint is met: As soon as a data packet arrives for any Interest transmitted at t_x where (t_x>t_i), and ($t_{current}$-t_i)> RTT, where $t_{current}$ is the current time and RTT is the time it takes to send an Interest and receive the Data packet for it. Once NDTP detects a packet loss using fast retransmit, the consumer reduces its congestion window by one half and sets the slow-start threshold to the new window size causing the consumer to go into congestion-avoidance mode.

RTO Estimate

Because of transparent caching, congestion detection based on retransmission timeouts (RTO) is not reliable in NDN when data are retrieved from multiple sources. Many of the congestion-control protocols proposed for NDN [113] argue against the use of a single RTO to detect a packet loss because consumers cannot detect when data is being retrieved from different locations. Unlike NDN, a consumer in NDTP relies on IP addressing to identify the source of each data packet, even when data is being retrieved from multiple sources. This allows NDTP to provide accurate RTO estimates by measuring the correct round-trip time for every data packet, while many NDN congestion-control algorithms (e.g., [119, 120] must guess the sources of data packets. In the case of a timeout event, the NDTP consumer retransmits the Interest that caused the timeout, reduces its congestion window to one Interest, sets the threshold half the congestion window size before

the timeout, and then goes into slow-start mode.

4.3 Manifest-yielding DNS

Applications such as HTTP that rely on URLs can be viewed as information-centric, given that URLs name a piece of content stored on the Internet. However, they are not location-independent because a client must first establish a TCP connection with the HTTP server specified in the hostname part of the URL in order to resolve a URL to the corresponding data object. Only then will the client be able to request the data named in the URL.

NDT attains location-independent content naming through the integration of name resolution with the transport protocol used to carry content reliably. A new resource record type, referred to as *Manifest Record* (MR) is added to the DNS, resulting in the **manifest-yielding DNS** (**my-DNS**). Instead of creating a new type of DNS resource record for the manifests, it is possible to encode the manifest using a TXT record instead. An MR describes the content structure by carrying the manifest generated by the NDTP producer, lists the IP addresses of the different locations of the content on the Internet, and other information, such as fields specifying the freshness of the content and fields specifying security parameters.

Content naming in NDT is inspired by the iDNS approach [100] to separate the content name from its location on the Internet. Content names in NDT are based on DNS domain names, allowing them to be persistent and unique through the hierarchical nature of my-DNS. For example, the content name contentA.ucsc.edu represents contentA hosted by the DNS domain ucsc.edu. With NDTP help, each content name on the Internet is mapped to an individual MR generated by its producer, as explained in the previous section. Having a single authority on

manifests allows consumers to authenticate the origin of content on the Internet easily. To achieve near-replica routing of content, my-DNS is used to map the name of a content object to the MR that describes the locations and structure of the content object to the consumers on the Internet. In turn, an MR maps the manifest of a content object to a list of IP addresses hosting a replica of the content. Each one of these addresses is added to the list as an individual DNS type A record. my-DNS updates this list as needed. This includes sorting the list by the nearest replica based on the consumer's geographical location issuing the DNS query for the content name. Such an approach is already being used to enhance domain-name lookup on the Internet by many vendors. Because NDT uses standard DNS procedures to resolve content names to MRs, it can rely on standard DNS procedures to dynamically register content names with its corresponding MRs. This is similar to a website adding DNS records to its authoritative DNS server. Content servers can dynamically register the content name with their MR using dynamic-update DNS mechanisms [118].

4.3.1 my-DNS Operation

Figure 4.3 shows the steps used in NDT to resolve a content name into a Manifest Record using my-DNS. In Step 1, an NDTP consumer issues a manifest query with the content name passed by the client application to the local my-DNS server. The manifest query is merely a DNS query with the name field (QNAME) set as the content name and the type field (QTYP) set as the MR type, in addition to the standard DNS message fields. Assuming that a specific my-DNS zone manages the MRs under its zone, the local my-DNS queries iteratively the global my-DNS for the location of the authoritative my-DNS server of the my-DNS zone specified in the URL. After the local my-DNS server obtains the IP

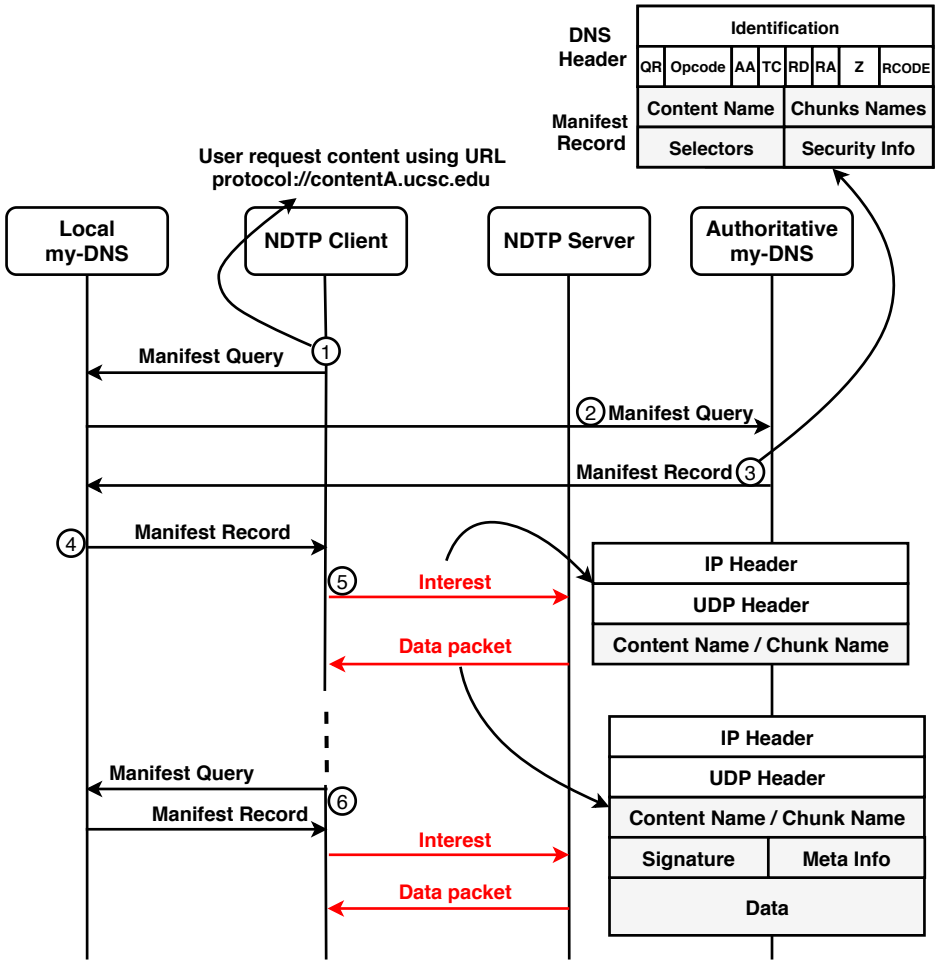


Figure 4.3: Mapping content names to Manifest Records

address of the authoritative my-DNS server, it sends a query of an MR type along with the content name, as shown in Step 2. In response to the query from the local my-DNS, the authoritative my-DNS server returns the MR associated with this content name, as shown in Step 3. After receiving the MR from the local my-DNS server, the NDTP consumer can start issuing Interests to retrieve the content, as shown in Step 4. When another NDTP consumer tries to retrieve the same content, the local my-DNS simply returns the MR that has been cached, as shown in Step 5.

NDT ensures the security of the content itself, rather than relying on closed private connections. To protect the authenticity and integrity of the content objects, the MR must also be secured. This could be done by relying on digital signatures based on public-key cryptography as in DNSSEC [123]. However, DNSSEC is not widely deployed, is expensive to operate, and is not viewed as a complete solution [122]. New techniques are clearly needed to secure and protect MRs, and they are the subject of future work.

4.3.2 Scalability

Without proper care, adding Manifest Records to the DNS could lead to scaling problems resulting from IP address changes for content servers and mirroring sites hosting large numbers of content objects, each with an MR that must be stored. Fortunately, adding a layer of indirection prevents this problem, and the DNS design already provides the means to add indirection via the CNAME resource records. Using the CNAME records instead of the A records of the content server inside the MR, DNS updates messages to the server are avoided. Whenever a content server changes its IP address, only the A record stated in the CNAME record of the content server needs to be updated. To ensure consumers keep up with the changes of the IP address of the content server, the TTL can be set low for these records. This action does not increase the load on the authoritative DNS server, as has been discussed in the past in the context of mobile networks [124]. In addition, notification mechanisms can be used to update the local DNS with the new IP address proactively using known consistency mechanisms proposed for the DNS [125].

Mapping a domain name for every content object requires several orders of magnitude additional storage capacity in the authoritative and local DNS server.

This may appear too onerous at first glance; however, as has been noted before [100], most of today's HTTP servers can handle such a load (by hosting an entire directory tree), and a dedicated DNS server can be used to host and manage MRs for content objects under a separate domain. For example, the DNS resolution for the hierarchical content name "ContentA.Contents.example.com" would involve a maximum of four requests, with the final one to the authoritative DNS server for (Contents.example) for contentA.

We note that the possibility of incurring additional redirection delays to reduce storage requirements in my-DNS servers is a better trade off than requiring all routers to maintain FIB's and PIT's that are several orders of magnitude larger than the FIB's of IP routers today.

In terms of the size of the Manifest Records that are handled by the DNS, RFC 1035 [126] already defines mechanisms on how to handle large DNS responses. This is done by relying on TCP instead of UDP to handle such a response. However, another approach is to use a layer of indirection by having an MR pointing to other manifests that can be retrieved from the content server responsible for publishing the content objects and its Manifest Record instead of using the authoritative DNS server.

4.4 NDT Proxies

The use of manifests in ITP and NDT eliminates the limitations of transparent caching in the IP Internet by describing the structure of content objects globally at the transport layer. Unlike ITP proxies, NDT Proxies (NP) track pending Interest from NDTP consumers using a Pending Interest Table (PIT) that serves the same purpose as in NDN. Adding PIT structures to NDT proxies allow it to support the main goals of ICN without requiring a pure ICN architecture. Specifically,

forwarding Interests in NDT proxies does not need a new routing infrastructure like NDN and similar ICN architectures and instead it will use the IP forwarding plane. In addition to the PIT structure, NDT proxies will also contain a Content Store (CS) that is used to cache NDT data packets to satisfy future Interests just as in NDN.

4.4.1 NP Operation

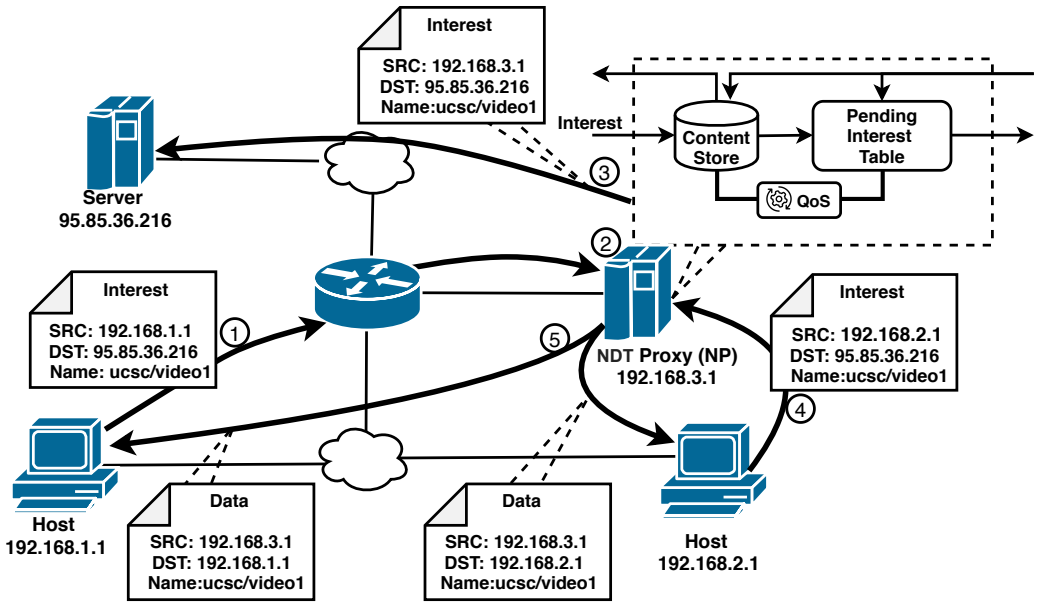


Figure 4.4: Transparent caching and Interest aggregation

An NDT proxy forwards an Interest only once towards the address of a content producer or mirroring sites using its own IP Address as the Interest source, and aggregates subsequent Interests in the same content. This is different from how transparent caching operates in ITP, where the goal was to create stateless servers and proxies. The PIT in NDT proxies is kept outside the routing infrastructure and does not impose additional overhead compared to today's web caches, which must keep track of TCP connections between the client and the origin server in a

data structure referred to as a connection tracker.

Figure 4.4 shows an example of transparent caching in NDT. After an NDTP consumer retrieves the manifest record of a content object, it proceeds with a window-based sequence of Interests requesting all the chunks that are needed to decode the requested content as stated in the manifest. An NDTP consumer at the client starts issuing Interests destined to the NDTP producer at the server (Step 1 in Figure 4.4). A layer-four switch on the way is configured to intercept Interests intended for the content server and forward them to a nearby NDT proxy cache ((Step 2 in Figure 4.4). The NDT proxy then checks whether the requested data packet is stored in its content store, and forwards the Interest towards the NDTP producer using its own IP address as the origin of the Interest (Step 3 in Figure 4.4). Once the NDT proxy receives a data packet, it uses it to satisfy any pending Interest received from other NDTP consumers as indicated in Step 5 of Figure 4.

4.4.2 Multicast Support at the Transport Layer

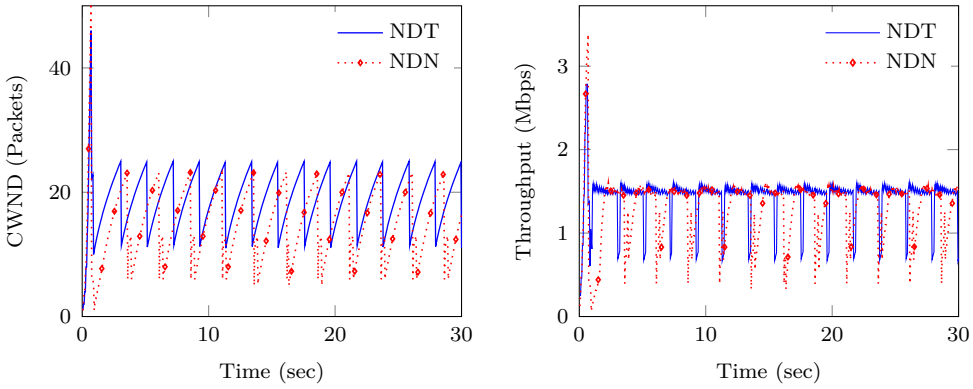
The IP multicast architecture in place today is based on the approach introduced by Deering and Cheriton [127]. The limitations of this approach have been discussed multiple times in the past [128], and among them are the need for global agreements on group addresses and the use of multicast routing protocols. CCN and NDN are able to provide “native multicast support," (i.e., supporting multicast delivery without multicast addresses and multicast routing protocols) by using PIT’s to track pending Interests for multicast content denoted by name.

As we have explained, NDT proxies use PIT’s to track pending Interests at the transport layer, and the aggregation of pending Interests at NDT proxies is very similar to that of NDN routers near consumers, but is based on IP addresses.

However, this still leaves multiple copies of Interests for the same object flowing through the routers along the paths between NDT proxies at the edge and content sites. To reduce this traffic, network providers may choose to deploy layer-four switches to intercept and redirect NDT traffic to caching proxies at different network locations between popular mirrored content producers and customers.

The combined use of NDT proxies and manifest records with which consumers can be redirected to nearest mirroring sites results in similar functionality as a CDN, but without the need for overlays.

4.5 Performance



(a) CWND Size (b) Throughput

Figure 4.5: Single-flow scenario for NDT and NDN

We compared the performance of NDT, TCP, and NDN using our NDT implementation in ns-3 [115] and off-the-shelf implementations of TCP, DNS, and NDN in ns-3 and ndnSIM [116]. The ns-3 implementation of NDT is publicly available to the research community [121] to facilitate reproducibility of results and future NDT improvements.

4.5.1 Efficiency of Congestion Control and Retransmission Mechanisms in NDT and NDN

We first compare the congestion control algorithm and retransmission policies of consumers on NDT and NDN. We assume that NDN uses an end-to-end protocol that behaves in the same way as TCP to provide a fair comparison. Accordingly, consumers in NDN can only infer congestion via a retransmission timeout and use AIMD window control to avoid congestion, such a mechanism is used by most end-to-end protocols in ICN. The size of the object chunks in NDN and NDT are equal to 1500 Byte. The topology of the network is a single path of four nodes with a single consumer at one end and a producer at the other end. Both ends share a common bottleneck of 1.5 Mbps and no in-network caching takes place. The propagation delay between the two ends is set to 40ms. The consumer in NDT issues Interests for the content served at the other end after requesting the manifest for this content from the producer.

	NDT	**NDN**
Total time (sec)	33.2694	35.9408
Avg. throughput (Mbps)	1.41298	1.2736
Packet loss	39	56
Jitter sum (sec)	3.3151	4.9951

Table 4.1: NDT and NDN single-flow results

Figure 4.5 shows the evolution of the congestion window and throughput for both protocols during the first 30s. The propagation delay between the two ends is set to 40ms. The growth of the congestion windows for both protocols matches the expected behavior of the AIMD algorithm. However, a consumer in NDN cannot detect the data source, which prevents the use of out-of-order delivery methods to detect packet losses. Accordingly, consumers must rely on methods that depend on retransmission timeouts [103, 104]. This degrades the overall completion time

for NDN compared to NDT as Table 4.1 shows. Unlike NDN, a consumer in NDT is able to detect where packets are coming from and use this information to detect out-of-order packets as a sign of packet loss (similar to duplicate ACKs in TCP). This allows the consumer in NDT to respond to congestion faster than relying on retransmission timeout as most end-to-end protocols in ICN.

4.5.2 Efficiency of Transparent Caching in NDT

This experiment highlights the ability of NDT to take advantage of in-network caching like NDN does, but without requiring any changes to the IP routing infrastructure. We compared the total average time taken to retrieve multiple copies of a large data file using NDT, TCP, and NDN. The experiment consists of a source node connected over a 10 Mbps shared link to a cluster of six consumers, all interconnected via 100 Mbps links. An intermediate router is configured to forward NDT traffic to a caching proxy for NDT traffic. The same topology was used for NDN as well. Both NDN and NDT use the same congestion control algorithm, which mimics the TCP congestion control algorithm to provide a fair comparison with TCP. The scenario was run six times, and each time we increased the number of consumers in the network. All consumers start pulling a 6MB data file from the source simultaneously, and the total download time for every consumer is displayed in Figure 4.6.

As can be observed from Figure 4.6, TCP, NDT, and NDN perform very much the same when a single consumer is involved. This is to be expected, given that most of the NDT and NDN data packets are retrieved from the source in this case, and all three approaches use similar algorithms for congestion control. As the number of consumers increases, the completion times in NDT and NDN remain constant for all six scenarios. In contrast, the completion time in TCP increases

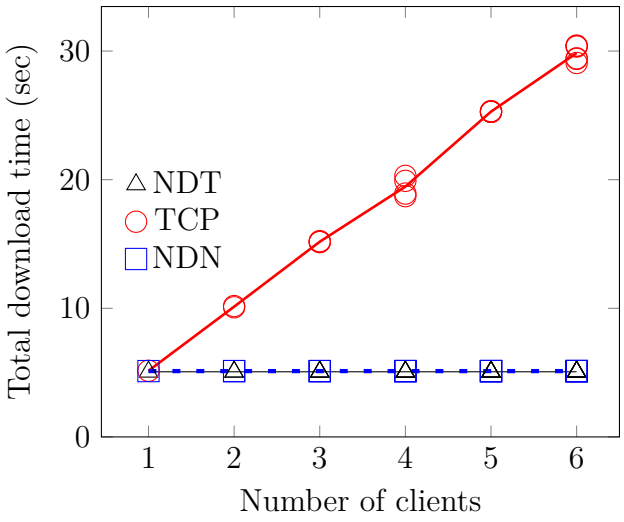


Figure 4.6: Transfer time vs. number of consumers

linearly because all the data must be retrieved from the source. The use of PIT's in NDT and NDN results in only the first consumer Interests traversing the path to the producer, while the rest of the Interests are simply added to the PIT of the first router in NDN and the caching proxy in NDT.

4.5.3 Impact of Manifest Records and Mirroring

This experiment highlights how NDT's architectural components work together to provide efficient name-based content delivery over the existing IP Internet. We compared the total average time taken to retrieve a large data file in three cases, namely: Using only the transparent caching enabled by NDT proxies (NP), using redirection to nearest mirroring sites based on my-DNS without transparent caching at NP's, and using NP's together with with redirection to nearest mirroring sits based on my-DNS.

Figure 4.7 shows the topology used in this scenario, which consists of multiple edge networks connected by a cluster of multiple consumers and mirroring sites located between the edge and the cloud network where the content server is lo-

cated. Each edge router is connected to NDT proxies that provide transparent caching for NDT traffic passing through them. When my-DNS is enabled, Interests from consumers are routed to the nearest mirroring site for the content. Each experiment was run five times and the number of consumers in each cluster was increased. Each consumer starts pulling a 6MB data file from the producer at a random start time based on a Poisson distribution with a short average arrival time.

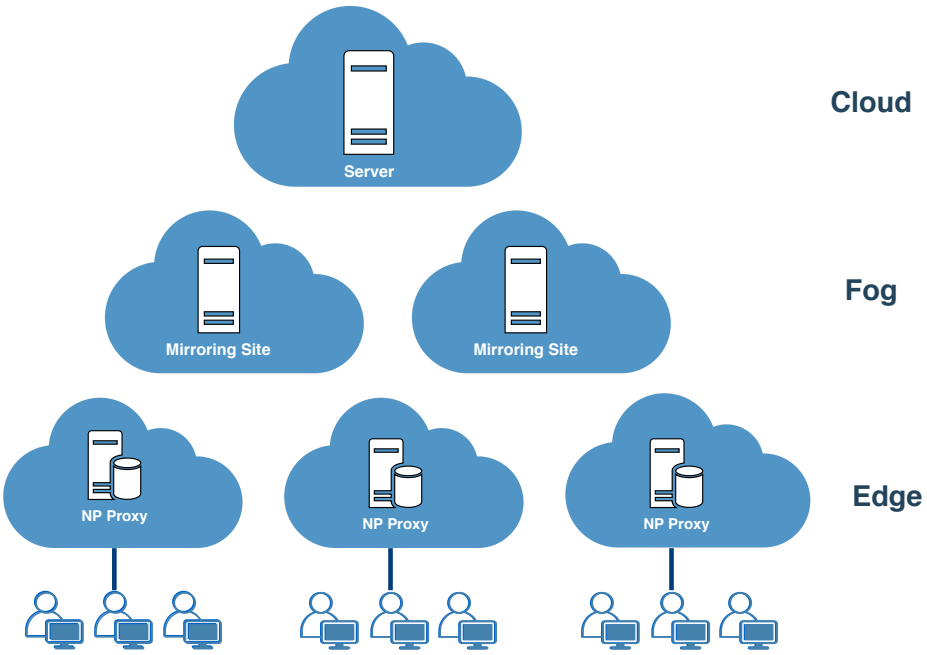


Figure 4.7: Network topology

Figure 4.8 shows the average latency incurred in retrieving the content object for each scenario, along with the variance. As expected, NDT performs its best when NDT proxies and nearest-replica routing through the my-DNS are used. As Figure 4.8 indicates, when only NP's are used, Interests from consumers have to reach the producer site, and the benefits of using NP's come from aggregating Interests and caching content. However, because of the short inter-arrival time of

Interests, only aggregation is useful. In our scenario with ten consumers, only two Interests for the same data object traverse the path to the producer. Using my-DNS without NP's results in a shorter retrieval time for consumers, but duplicate packets are sent along links. When both my-DNS and NP's are used, Interests from consumers are routed to the nearest mirroring site and aggregated at the NP's.

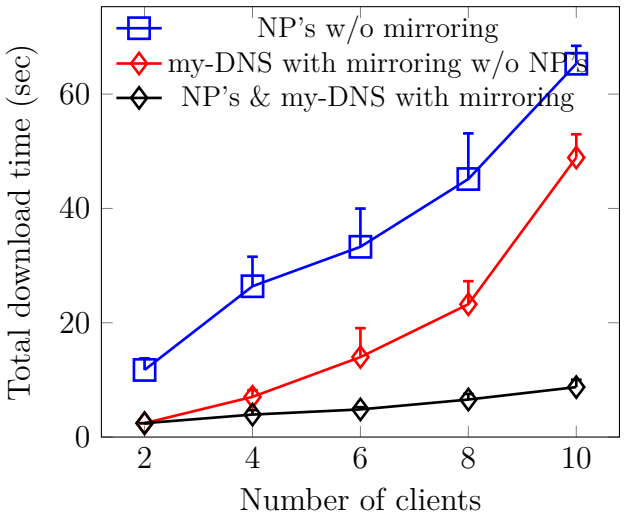


Figure 4.8: Total download time for different approaches to the use of mirroring and caching in NDT

4.5.4 Overhead of URL to Manifest Mapping

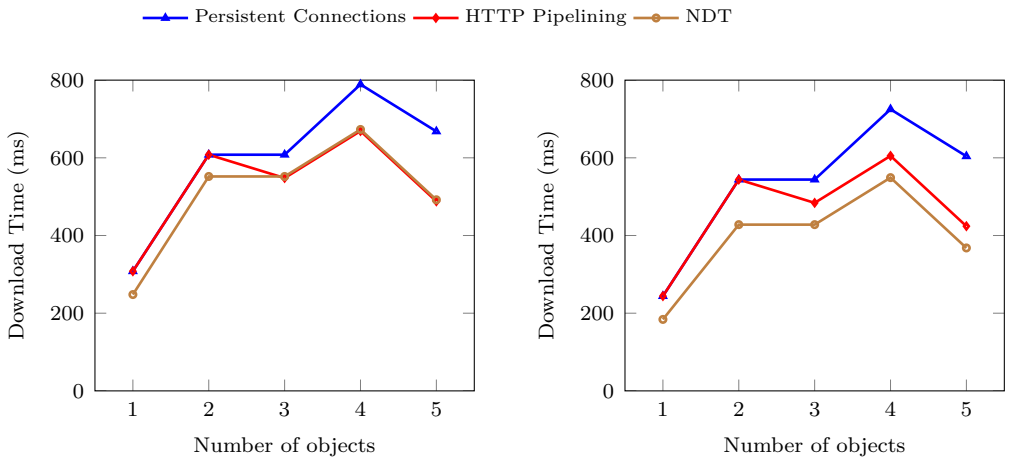
This experiment illustrates the overhead of mapping URLs to manifest records using my-DNS. The scenario is based on the HTTP application, where clients start requesting the main web page and then start requesting the embedded inline objects based on their URL in the web page. The main object size and the size of the embedded inline objects are based on the top one million visited web pages indicated in [129]. We used two HTTP applications for our comparison, one based on persistent connections, in which a new HTTP request cannot be sent

until the response to the current request is received. The other application is based on HTTP pipelining, where multiple HTTP requests can be sent together over a single TCP connection. For the case of NDT, each URL mapped to a single manifest record. The NDTP consumer starts querying the my-DNS for the manifest record of the main object, and it queries for the inline objects records after retrieving the main object from the server.

The topology of the network is a single path of four nodes with a single client connected to its local my-DNS server, and a content server at the other end that is connected to its authoritative my-DNS server. For the sake of simplicity, the IP address of the authoritative my-DNS server is cached at the local my-DNS server at the start of the simulation. Both resource and manifest records need to be retrieved from the authoritative my-DNS server at the other end if they are not cached. For a fair comparison between NDT and HTTP over TCP, NDT and TCP have the same fixed-header size, the same chunk and segment size, and the same congestion control algorithm.

As Figure 4.9a shows, NDT performs at least as well as HTTP pipelining. Both HTTP applications require a connection to be established using TCP three-way handshakes before a client can start sending and receiving data. By contrast, NDTP allows clients to retrieve data without the need to establish a connection, which reduces the number of RTTs by at least one compared to TCP. Multiplexing is easily supported in NDTP because objects in NDTP are globally named and pointed by their own manifest, allowing consumers to pipeline and multiplex multiple objects together. As seen in Figure 4.9b, when manifest records are cached, NDT outperformed both types of HTTP. This proves that using my-DNS to translate URLs as structured in applications like HTTP do not impose significant overhead in NDT.

Using my-DNS in NDT does not impose significant overhead compared to NDN. A consumer in NDN has to retrieve the manifest from the producer before issuing Interests to retrieve the content. Retrieving the manifest record using my-DNS adds only the additional delay incurred in redirecting the consumer to the site with the manifest.



(a) No cached DNS records

(b) With cached DNS records

Figure 4.9: Overhead of URL-to-manifest mapping

4.6 Conclusion

In this chapter, we introduced Named-Data Transport, the first ICN approach that attains efficient content dissemination without end-to-end connections or modifications to the IP routing infrastructure. The design of NDT provides the same benefits of NDN and similar ICN architectures through the integration of a new connectionless reliable transport protocol with name resolution and NDT proxies that support on-line caching and native multicasting.

NDT allows applications to ask for content or services by name, and makes the transport layer responsible for: (a) mapping content names to one or multiple

locations where the content is offered, (b) delivering the content reliably to consumers from producers or transparent caches without end-to-end connections. In the next chapter we will see how the transport layer can also enforce the privacy of consumers by using the Manifest.

The results of simulation experiments in ns-3 show that: (a) NDT is inherently more efficient than TCP, (b) the performance of NDT and NDN is very similar, and (c) NDT outperforms HTTP over TCP while being able to provide privacy.

We focused on static content in our discussion of NDT; however, the approaches that have been described for the support of real-time voice and video-conferencing in NDN and CCN [131, 130] are equally applicable to the end-to-end information-centric approach in NDT.

Chapter 5

Secure Transparent Caching

Transparent in-network caching is meant to be a key benefit of ICN architectures, with content being cached opportunistically throughout the network where it is needed. However, intermediate routers and caches have access to the content object being transferred to consumers, which poses privacy concerns. The same goes for transparent caching in the current Internet design, where it leaves cached content open to intermediate routers and caches. On the other hand, applications that employed TLS (Transport Layer Security) to ensure complete security precludes caching by establishing a closed connection between consumers and content providers. Thus, losing tremendous performance benefit over privacy.

Access to caches in ICN architectures can expose consumers' privacy as an attacker can use these caches to learn about consumers' incoming and outgoing content traffic. Many kinds of literature have already introduced different types of attacks that can be used to expose consumers' and producers' privacy in NDN. For example, in a timing attack, an adversary can leverage round trip time measurements to learn whether a nearby consumer recently requested certain content by distinguishing cache hits and cache misses. In addition, NDN names cannot be protected directly by encryption, as doing so will prevent content objects from

being routed. This allows an adversary to easily detect cache hits or misses to learn about consumers' requests. The adversary can be a normal consumer. It will just compile a list of the names of privacy-sensitive content objects and periodically just test them, or the adversary can be a privileged stakeholder such as government authorities ISPs in that case, they can have direct access to content names.

Chapter 3 and 4, outlined what a manifest is and how it is used in ITP and NDT to establish an association between remote processes without the need for end-to-end connections. We also outlined how manifests and manifest records eliminate the limitations of transparent caching in the IP Internet by describing the structure of content objects globally at the transport layer. This chapter presents a design of how transparent caching with privacy can be enforced in NDT and ITP by using the manifest.

5.1 Overview

From a security standpoint, content carried over ITP and NDT can be public or private. Protecting public content privacy is not a concern; however, authentication and integrity are necessary. Ensuring the integrity and authenticity of the manifest and manifest record allows ITP and NDT to ensure the security of the content as well. Part of a manifest is the name of the chunks that need to be requested using Interests in order to construct the content object. By using a hashing function, consumers in NDT and ITP use each chunk hash digest as the name in their request. As a result, the manifest contains the content name, the digest of each chunk composing the content, and the hashing function used by the server to name these chunks, similar to the one proposed for NDN[110]. Finally, by computing the hash digest of these chunks, a consumer can verify the

authenticity and integrity of a received data packet.

Multiple security methods can be leveraged to ensure the protection of private contents in NDT and ITP. The goal is to ensure privacy while also enabling transparent caching of content. To achieve this, the following must be ensured: (a) Only the producer and the consumer of a content object should be able to access the content object to preserve privacy, (b) producers and consumers must be able to authenticate each other, (c) consumers must be able to detect the integrity of received content objects, and (d) both protocols should at least provide the same level of anonymity as HTTPS.

TLS and other secure methods based on end-to-end connections rely on symmetric cryptography per each connection to ensure privacy. Given that the keys they use are generated uniquely for each connection, it is apparent that using similar keys to secure content in ITP and NDT would negate the benefits of transparent caching when using the manifest.

One simple way to benefit from transparent caching with symmetric cryptography is to use group keys. Such an approach certainly downgrades the privacy of those consumers retrieving the same content objects, but also ensures that caches are unable to access the content cached for the group. To increase the level of privacy between content consumers, computational asymmetry is used by encoding the data using a specific method known only to one particular consumer. This is done by combining a content object with useless data (or old chunks in the content store) intended solely for obfuscation.

5.2 Group Key

Using group keys should be enough to prevent an adversary from accessing cached content. However, compared to TLS in which an encryption key is used

for each connection, this might raise two security issues: (1) a member inside the list may infer that other members in the same list are retrieving the same content object by observing the chunks names; and (2) members that are not part of the content list may infer that multiple consumers are requesting the same content by observing the names of requested chunk names. ITP and NDT overcome this problem by encoding part of the content for each consumer in a different way using computational asymmetry. This can be done by mainly relying on old chunks to do this, which increases the chance of these chunks being in a cache, reducing the latency to retrieve them. As a result, consumers retrieving the same content will be requesting different sets of chunks.

5.3 Encoding Data

The way how a manifest can be used to encode the data in ITP and NDT to ensure computational asymmetry is by combining the content of an object with old data or “useless" data intended solely for obfuscation to produce a set of coded fragments. For example, multiple chunks of the original content can be encoded with other chunks from the content store, resulting in new coded chunks. The way in which these chunks are encoded is only shared between the content consumer and the content producer. Even though such a method does not provide complete privacy, it should make it computationally expensive for caches to access cached contents.

One way to encode the data of a content object in ITP and NDT is by leveraging previous work on censorship at storage systems [132]. A similar method was also used in [91] to prevent real-time censorship at nodes in ICN networks. Algorithm 2 shows how a producer in ITP or NDT can encode the content for data obfuscation.

The return manifest of Algorithm 2 is simply a list of tuples, each describing what chunks to request and how to decode the original chunk as shown in Eq. (5.1). After receiving its manifest, the consumer simply sends Interest for the coded chunks corresponding to the original chunk in the manifest as shown in Eq. (5.2).

$$\left[\left(h(B_{1,1}), ..., h(B_{c,1}), h(X_1)\right), ..., \left(h(B_{1,n}), ..., h(B_{c,n}), h(X_n)\right)\right] \tag{5.1}$$

$$f_i += (X_i \oplus_{j=1,...,c} B_{j,i}) \ for \ i = 1, ..., n \tag{5.2}$$

Algorithm 2 Data Obfuscation

```
1:  procedure ENCODE(content)
2:      f ← SEGMENT(content)                          ▷ return list of chunks
3:      for i ∈ {1, ..., n − 1} do
4:          X_i = f_i
5:          ChunkName ← NULL
6:          for j ∈ {1, ..., c} do
7:              X_i = (X_i ⊕ b_j)
8:              ChunkName ← h(b_j)
9:          ChunkName ← h(X_i)
10:         Manifest ← ChunkName
11:     return Manifest
12: ──────────────────────────────────────────────────
13: c: number of random old chunks
14: n: number of content's fragments
15: h: hash function
16: ChunkName: tuple of hash digest
17: Manifest: list of tuples
```

5.4 Partial Encryption

Data encoding alone is not sufficient to provide complete confidentiality, but it could be enforced by using group keys to **partially encrypt** some of the encoded chunks. Figure 5.1 shows a high-level view of the encryption operation. In the example, two consumers retrieving the same content have two different encoding

methods but only one group key. Having a group key ensures that caches are not able to decrypt cached chunks. The group key also allows for overlapping requests to benefit from transparent caching.

The basic approach serves two goals. First, it provides privacy among consumers retrieving the same content by encoding part of the content differently for each consumer. Second, given that each consumer has its own secret on how to decode part of the content, an adversary cannot easily infer if consumers are retrieving the same content. The reason why consumers with the same group key cannot be sure about other members of the same group is that chunks used in the data obfuscation might have been used to encode other content as well. This should be sufficient to prevent consumers from decoding these chunks, mainly when they have limited resources. Besides, by using the group key, we are guaranteeing **partial caching** for all the chunks that are encrypted using the group key.

To increase the level of obfuscation, the producer can apply an all-or-nothing transform, where caches or consumers in the same content list cannot decode the content unless all chunks are known. The trade-off with obfuscation and using group keys is that more coded chunks that combine the original content are needed to decode it. However, the older the chunks used to obfuscate the content, the higher the chance that it will be retrieved from a closed cache. Also, this method can be used to populate caches with chunks from other content objects.

5.5 Manifest Privacy

To provide partial encryption while also leveraging caching, only the manifest has to be secured using public key encryption. By using a specific URL format, a consumer knows that its request is for private content (like in HTTPS). The

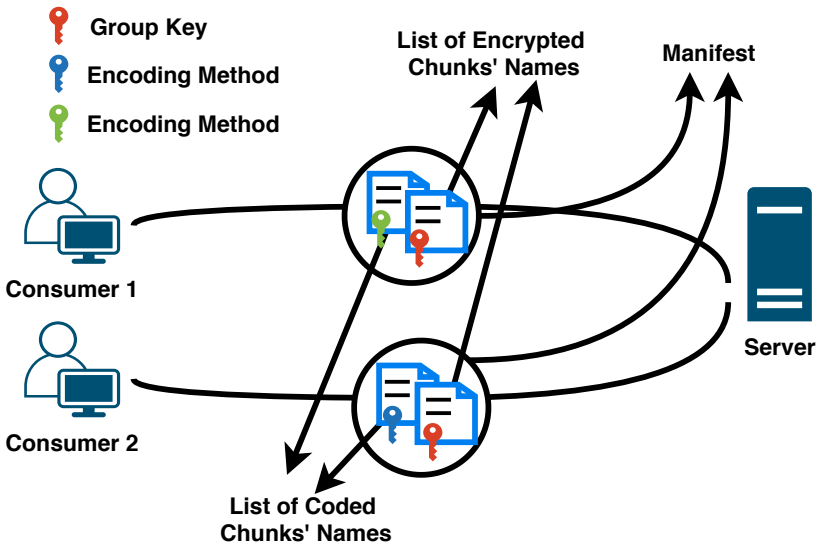


Figure 5.1: Partial encryption using manifests

consumer then sends its request to the IP address of the content provider, just as in HTTPS. Consumers can use the DNS to retrieve the necessary keys of the content producer (server). This can be done by using techniques like DNS-based Authentication of Named Entities (DANE) [133], which allows the publication of Transport Layer Security (TLS) keys in zones for applications to use. Other methods can also be leveraged to exchanging and validating public key encryption such as the one used in TLS.

After the consumer retrieves the IP address and the server key, it can issue a *Manifest Interest* encrypted using both the client and the server public and private key to ensure its authentication, integrity, and encryption. Once the server proves the client is authorized to request the content, it will then send back the manifest. While this is happening, caching proxies on the way will not be able to intercept and understand either the Manifest Interest or the manifest itself since they are encrypted. To allow transparent caching while also ensuring privacy, the manifest contains a specific encoding method that is unique for each consumer and a group

key that is unique for each content as we explained earlier.

5.6 Name Privacy and Access Control

Because Manifest Interests and manifests are encrypted using public-key encryption, no intermediate node is able to access any information inside them, including the name of the content requested. This means that consumers that are seeking the same content will not know who else is requesting it since both the Manifest Interest and the manifest are encrypted using the consumer and the producer keys. The same applies to members that are not part of the content group as well.

As we explained before, manifests uses chunk hash digest as its name. To prevent an adversary from reversing these chunk names to their original content, a one-way hash can be used. In generating these hash digests, a producer uses a randomly generated number (salt) to control access to chunk names from consumers who have their access revoked. This salt is added to the manifest along with a timer on when this manifest and salt will expire. A consumer who was at some point authorized to receive a certain content but not anymore is unable to infer whether a particular chunk name will be related to a specific content object without the current salt value. Besides, consumers who have their access revoked will also be unable to request the content.

5.7 Conclusion

This chapter presented a design of how transparent caching with privacy can be provided using a manifest. This is done by mainly relying on two methods;

- Symmetric cryptography using group keys, which are used to prevent an

adversary from accessing cached content.

- Computational asymmetry by encoding part of the content for each consumer in a different way to prevent consumers from inferring about other members in the same group.

Thanks to the connection-free nature of ITP and NDT by using a manifest, the same method can be used in both protocols with no need for any modifications.

Chapter 6

DCP: Delay-based Congestion Control Protocol

In the previous chapters, we used an AIMD congestion control algorithm in ITP and NDT design. This chapter explores other methods to control congestion that does not rely on packet loss and is suitable for most Interest-based architectures, including NDN and NDT. One of NDT's advantages is its implementation at the userspace, where it defines most of its functionalities, including congestion control. This allows application developers to alter the congestion control algorithm based on their application needs. The LEDBAT (Low Extra Delay Background Transfer) protocol [134] was recently introduced as the new transport protocol operating on UDP for BitTorrent. It has the distinctive feature based on a delay-based congestion-control algorithm that uses queuing delay estimates over the forward path between senders and receivers of packets. This chapter focuses on designing a delay-based congestion control algorithm based on LEDBAT in the context of information-centric network (ICN) architectures, where contents disseminated using Interests issued by consumers and producers caching sites respond to them sent the content requested.

This chapter introduces the delay-based Congestion-control Protocol (DCP), a protocol designed to detect and control congestion at the consumer end of a network based on an Information-Centric Networking (ICN) architecture without the use of round-trip time measurements or additional congestion signaling. DCP is based on a receiver-driven, window-based approach to congestion control using measurements of delays along the forwarding path from a producer or caching site to a consumer. DCP was implemented using the ndnSim simulator and compared to two other protocols, namely: an end-to-end protocol that behaves in the same way as TCP and a pure implementation of the Low Extra Delay Background Transport (LEDBAT) algorithm as a congestion-control protocol for ICN's. Even though DCP was implemented in the NDN environment, both NDT and ITP can benefit from the same algorithm considering both protocols replace end-to-end connections with Interest-based architecture similar to NDN.

6.1 Overview

A major part of the Delay-based Congestion Protocol (DCP) design is retransmission and congestion control algorithm that avoid replicating the same design and performance limitations in TCP. An ideal congestion control protocol for DCP should be able to achieve the following goals: (a) To utilize end-to-end available bandwidth and knowledge of whether the content is being served from a close cache or a distance producer; (b) to maintain low queuing delays for data packets in a way that does not affect the application performance, and (c) to use the bottleneck link fairly by yielding to other network flows that share the same link.

Estimates regarding the increase or decrease of RTT in NDN/CCNx should not be used as primary indications of congestion in the network. Using RTT

estimates is not efficient to determine whether congestion increases or decreases along the interest-path or data-path. Figure 6.1 shows an example of two Interests sent by the same consumer and their corresponding data packets. Using only RTT measurements could lead to an incorrect conclusion of congestion developing in the data-path for the second packet. However, the true cause for the increased RTT for the second data packet is congestion along the interest-path, not the data-path.

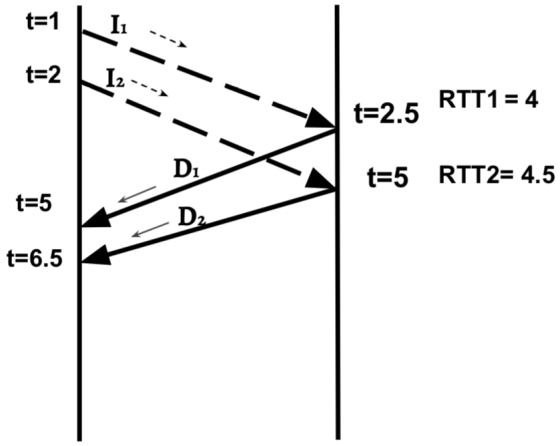


Figure 6.1: Example of RTT ambiguity

To cope with the above complexities facing the design of a delay-based congestion-control mechanism, we used prior work on algorithms for congestion control and retransmission in TCP Santa Cruz (TCP-SC)[135] and LEDBAT as the baseline of our design. Both protocols based their congestion control algorithm on estimating the queuing delay along the forward path to the receiver, rather than using RTT. Besides, both protocols implicitly attempt to establish a critically-damped control system in which a sender attempts to quickly fill the capacity of the bottleneck receiver and uses estimates forward queuing delay to force the transmitter to slow down or speed up transmission attempting to maintain minimum queuing delay.

The major differences between ICN-based architectures and LEDBAT or TCP-SC are that: (a) there is no need for connection establishment ICN, (b) ICN is receiver-initiated based on Interests and data packets sent back, and (c) the basic units of data transfer in ICN are chunks rather than bytes. However, these differences are what makes ITP and NDT common with NDN and CCNx since all four are viewed as an Interest-based architecture. Therefore, implementing a delay-based congestion control algorithm in ICN architecture like NDN is beneficial for ITP and NDT as well.

6.2 Protocol Design

6.2.1 Measuring Packets Delay

In ICN, consumers can calculate the one-way delays incurred by data packets in their own system time by using timestamps returned from the producer or middle nodes (caches) in the data packet headers. A producer or intermediate nodes timestamps the transmission time of the data packet in its header so that the consumer to calculate the one-way delay of a data packet. The consumer then keeps both the transmission time and arrival time of a data packet. Based on these timestamps, the consumer can calculate the one-way delay for the data packet as follows: $delay_{\mathrm{d}} = A_{\mathrm{d}} - T_{\mathrm{d}}$, where A is the arrival time for a packet and T is the transmission time of that packet.

6.2.2 Congestion Control

As we mentioned earlier, DCP uses delay measurements along the data-path to estimate the queuing delay. Whenever the estimated queuing delay is less than a predetermined threshold, the consumer infers that the network is not yet con-

gested and increases its sending rate by increasing the congestion window (interest window) to utilize any bandwidth capacity of the network. When the estimated queuing delay becomes greater than the predetermined threshold, the consumer decreases its sending rate as a response to potential congestion in the network. The algorithm used for congestion detection by this protocol is shown in Algorithm 3. The algorithm is similar to the one used in LEDBAT [136] with slight changes. In this algorithm, we also used the same/similar terminology as in LEDBAT, where TARGET is the maximum queuing delay that the consumer may introduce in the network. To ensure fairness between multiple DCP flows, the TARGET value must be the same. off-target is a normalized value representing the difference between the current measured queuing delay and the predetermined TARGET. Notice that off-target can be positive or negative; a negative off-target will reduce the congestion window as the queuing delay is exceeding the application delay threshold.

Algorithm 3 DCP: Estimating Queuing Delay

```
1: function on initialization()
2:       base_delay= +INFINITY
3:
4: function ONDATA(dataPkt)
5:       dataPkt ← current_delay
6:       base_delay=min(base_delay, current_delay)
7:       if (current_delay > base_delay)
8:             queuing_delay=(current_delay-base_delay)
9:             off_target=(Target-queuing_delay)/TARGET
10:       else
11:             queuing_delay=(base_delay - current_delay)
12:             off_target=(Target+queuing_delay)/TARGET
13:
14: function updateCWND(off_target)
15: end function
```

Algorithm 4 DCP: Window Update

```
1: function on initialization()
2:       flightsize= 0
3:       CWND= INIT_CWND
4:
5: function updateCWND(off_target)
6:       CWND+=(off_target * GAIN)/CWND
7:       max_allowed_CWND=flightsize +1
8:       CWND=min(CWND, max_allowed_CWND)
9:       CWND=max(CWND, INIT_CWND)
10: end function
```

6.2.3 Measuring Base Delay

A consumer using DCP can detect congestion in the network by measuring data packets queuing delay and compare it to the TARGET. However, it is important to distinguish the queuing delay from the rest of the end-to-end delay components, as stated by the LEDBAT protocol. The end-to-end delay is the time taken for a packet to be transmitted across a network from source to destination. It can be decomposed into nodal processing delay, transmission delay, propagation delay, and queuing delay. Except for the queuing delay, the rest of the end-to-end delay components are considered constant. Separating the queuing delay component from the rest of the end-to-end delay enables the consumer to detect any change in the network's queuing delay. The minimum delay that packets experience along either the data-path or interest-path is referred to as the *base delay*. The base delay is considered to be the sum of all of the constant delay components since queuing delay is always an additive element of the end-to-end delay. Calculating the base delay in DCP is done with a similar approach to that used in LEDBAT. It works by maintaining a list of one-way delays minimal over a time interval. Then only the minimum value of the minimal delays in the list is used as the approximate base delay for this path. A longer time interval results in a higher likelihood the true base delay is detected. However, a shorter time interval allows the base delay to respond quickly to route changes. We set the time interval to be 60 seconds as in LEDBAT's original algorithm and the base delay list length to be 6.

6.2.4 Measuring Current Delay

The consumer calculates the current delay for every data packet. It is clear that the current delay value can fluctuate from one data packet to another, which

affects extracting the actual delay estimate. To eliminate any outliers, LEDBAT suggests the use of filters. One suggestion was to use the Exponentially Weighted Moving Average (EWMA) of the current delay. Another suggestion was to use a NULL filter that does not filter at all. However, in our implementation of DCP, the consumer maintains a fixed-size list of the current delay that gets updated for every new value it calculates. The length of the current delay list is 5, and only the minimum value is used to estimate the queuing delay.

6.2.5 Measuring Queuing Delay

The queuing delay estimation is simply the difference between an end-to-end delay measurement and the current base delay estimate. The consumers measure the queue delay for every data packet received after updating the base delay and the current delay using Algorithm 4. If the queue delay of each packet exceeds the TARGET delay, the consumer will consider the network is congested and reduce the congestion window size according to Algorithm 4. However, if the queue delay is less than or equal to the delay target, the consumer will increase the congestion window's size. DCP uses a linear controller to increase and decrease the consumer congestion window, allowing the protocol to recover quickly from bad delay samples. The increase and decrease of the congestion window are proportional to the difference between the current queuing delay estimate and the TARGET delay. The congestion window in DCP, refers to the number of Interests a consumer is allowed to send.

As the queuing delay gets closer to the TARGET delay, the window growth gets slower. To compete with concurrent TCP flows fairly, the LEDBAT algorithm maximizes the congestion window growth by 1 per RTT. This happens when the queuing delay estimate is zero. The same approach is used in DCP

during congestion avoidance state where the maximum window growth is only 1 per RTT as seen in line 7 in Algorithm 4. Also, in LEDBAT, the sender may maintain its congestion window (cwnd) in bytes or packets. The decrease in the congestion window in DCP happens when the queuing delay estimate exceeds the application's TARGET delay. The window's decrease is proportional to the difference between the TARGET delay and the current queuing delay. This allows the consumer to decrease its sending rate as a response to potential congestion in the network, unlike loss based congestion control protocol. However, data loss can still occur in DCP. A data loss is considered an indication of strong congestion in the network.

The TARGET delay determines how much queuing delay is tolerated in the network. For example, if the target's value is set to 0, it means that no delay is tolerated in the network. This can lead to underutilization of the network's available capacity because the bottleneck queue will always be empty. A high TARGET delay might increase congestion in the bottleneck and cause packets to be dropped. Therefore, the TARGET delay choice is a trade-off between these two cases. The LEDBAT algorithm uses a TARGET value of 100 ms, or less [136]. In our current version of DCP, the default TARGET value is set to 50 ms. However, applications have the option to set their TARGET value based on their delay requirement.

6.2.6 Timeout

If the timeout interval for Interests is too small, unnecessary retransmissions of Interests occur. On the other hand, a timeout interval that is too large results in the consumer retransmitting Interests too late when a data packet or an Interest is dropped. LEDBAT uses a congestion timeout (CTO) as a strong indication of

heavy congestion in the bottleneck when a packet gets dropped. The LEDBAT CTO approach is based on retransmission timeout (RTO) in TCP. Even though consumers in DCP are able to indicate congestion in the network before a data packet is dropped at the bottleneck queue, we still use CTO as an indication of strong congestion in the network. When an Interest or data packet gets dropped, a CTO event is triggered. This results in the consumer reducing its congestion window to 1. The CTO calculation is based on the RTT estimation, which is the time it takes to send an Interest and receive the data packet for it.

6.3 Fairness Among Consumers

Using a delay-based congestion control algorithm in an Interest-based architecture like ICN and NDT raises several challenges. For example, LEDBAT suffers from the fairness issues known as the later-comer advantage, which also occurs between consumers in DCP. In addition, fairness issues can arise among consumers in ICN, ITP, and NDT when caching takes place. Another issue is that synchronization in clock rate can affect the delay measurements. In this section, we will go over each of these challenges.

6.3.1 Latecomer's Advantage

Both LEDBAT and TC-SC suffer from fairness issue, known as the "latecomer advantage", in LEDBAT. For example, in LEDBAT, late flows arriving at a non-empty bottleneck increase their sending window more aggressively than existing flows due to wrong estimates of the base delay. For example, If one flow arrives at an empty bottleneck, it will measure the correct base delay of the network, leading to a correct estimate of the queuing delay. However, if a new flow arrives

at the bottleneck, it will calculate the first flow's queue delay as part of its base delay, leading to a wrong estimate of the queuing delay, thus a higher target delay than the first flow. This will result in an aggressive increase of the second flow's window size, causing the first flow to back off as it sees the true queuing delay of the bottleneck increase. The second flow will eventually start using the entire bottleneck's capacity while shutting the first flow out. There have been several solutions to the latecomer advantage in LEDBAT [137]. DCP solution to the latecomer advantage is based on LEDBAT's original approach, by using TCP like slow-start.

A solution to the latecomer advantage is introducing TCP slow-start at the beginning of DCP flows. The goal is to drain the queue empty by filling it, which will likely induce losses on existing flows, allowing the newcomer to measure the correct base delay. Such a mechanism cannot ensure fairness without causing packets to drop off other flows, but such a drop will be limited. However, the above consideration suggests that a more suitable solution is needed. Thus, we leave the topic of evaluating different fairness solutions in DCP for our future work. DCP slow-start is similar to TCP slow-start mechanism; it uses it for new consumers joining the network to ensure correct measurement of the base delay. During the slow-start, consumers double their sending window for every round-trip time until a timeout event is triggered. After a timeout event is triggered, consumers enter the congestion avoidance state. Consumers in the congestion avoidance state follow the original algorithm in LEDBAT, where the maximum increase of the window size is 1 per RTT.

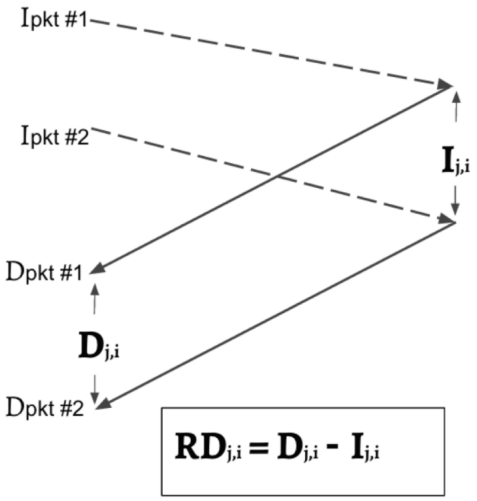


Figure 6.2: Transmission of two data packets and the corresponding relative-delay measurements

6.3.2 Latecomer's Disadvantage

Because of ITP and ICN's design nature, data might be served from different sources. This can lead to underutilization of the link's capacity to some consumers when caching takes place. For example, if a consumer's initial requests were satisfied by a close cache that was generated by an old consumer. After the new consumer exhausts the cache, it will join the first consumer in retrieving the data from the producer. The new consumer Interests will no longer be satisfied by the close cache, and instead, it has to traverse all the way to the producer. This scenario will cause underutilization of the link's capacity. This is because the new consumer base delay measurements were affected by the close caching site. We identify this as the "latecomer's disadvantage". Consumers in DCP overcome this issue by resetting their base delay list and calculating a new base delay whenever they identify contents retrieved from a new site. In ITP and NDT, this can be easily done by observing the data packets' IP address. However, in other ICN architectures, a new method is needed since consumers cannot infer

where chunks are coming from. An immediate calculation of the relative delay can help determine whenever contents are being retrieved by a new site in ICN architectures. To determine the change in the relative delay value of data packets, we use $RD_{\mathrm{j,i}} = \Delta D_{\mathrm{j,i}} - \Delta I_{\mathrm{j,i}}$, where RD$j, i$ represents the delay experienced by packet j with respect to packet i.

Figure 6.2 shows the arrival of two Interests 1 and 2 at the provider and the arrival of two data packets corresponding to Interests 1 and 2 at the consumer. The relative delay value fluctuates from one data packet to another due to congestion in the network. Therefore, it is important to distinguish between a change in the relative delay of data packets due to congestion in the network rather than being retrieved from a new site. Because of this fluctuation, any specific relative-delay value could be atypical. Hence, an average of the relative delay gives a better estimate of the delay between packets in the network. The mean absolute deviation (MAD) is the average of the absolute deviations between each relative delay value and the average delay. Calculating MAD helps deal with relative delays with a negative value. The MAD value is measured for the last 10 relative delays samples and is updated with every new value. Upon obtaining a new value of the relative delay, DCP estimates how far the new relative delay is from the average relative delay. If it is higher than the MAD value, then the consumer infers that content is being received from a new site. To ensure fairness between this consumer and existing DCP flows, a consumer should resume a slow start whenever the base delay history is reset, as we mentioned earlier. Also, since the relative delay value is affected by packet dropout at the bottleneck queue, the relative delay is not measured during slow start.

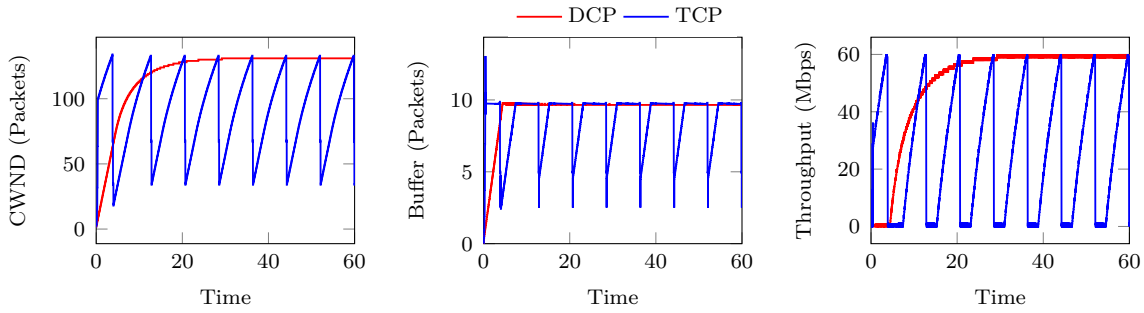
6.3.3 Synchronization

No synchronization between the consumer and the producer or middle nodes is required in DCP if data are retrieved from the same source. This is because measurement errors in delays due to clock offset are canceled out by their difference in the queuing delay estimate. Even though clock offset errors have no impact on LEDBAT, clock offset could be critical in an ICN architecture. This is because an ICN allows Interests to be resolved from caching sites along the paths to the nodes with best-match names for the Interests. For example, suppose two consumers connected to the same router with caching want to retrieve the same data, and consumer b starts a few seconds after consumer a. In that case, consumer b initial requests will be satisfied by the cache at the nearby router. If consumer b receives all the cached content, it joins consumer a in retrieving the data from the producer; this could lead to an inconsistent calculation of the one-way delays of data packets if the cache and the producer have different clock offsets. Thus, causing clock offset to show up as a linearly changing error in a time estimate instead of a fixed error. Such inconsistency is limited to the amount of delay history maintained in the base delay estimator used by DCP for measuring the one-way delays for data packets.

Measurement errors in delays in an ICN can be caused by different clock offsets of different caching sites. This translates into a difference in the clock rate from its true rate and can be seen as a clock skew problem similar to the skew of LEDBAT's one-way delay estimate. One possible consequence of this issue in DCP occurs when data are retrieved from a new data site with a greater clock offset than the previous site clock offset. This results in a low estimate of the queuing delay; however, it also leads to a lower base delay measurement, so it does not lead to unfairness for the consumer. Another possible consequence of

the issue happens when data is being retrieved from a new data site with a clock offset less than the previous site clock offset. This results in a high estimate of the queuing delay. This increase can reduce the throughput of a flow because the base delay will only keep the minimal samples over a time interval. A high change in the relative delay of data packets could be used by the consumer to infer that such change in the delay is due to clock drift in the clock offset, which means data is being retrieved from a new source. The sender then can compensate by recalculating its base delay estimate.

LEDBAT suggested a correction mechanism for the synchronization problem and related to clock skew. Even though clock skew is usually very small, we believe more research is needed to define the synchronization issue in an ICN architecture. We leave the topic of researching and testing different solutions to the issue as part of our future work.



(a) CWND Size (b) Router's Buffer Occupancy (c) Throughput

Figure 6.3: Single-flow scenario

6.4 Performance Results

We evaluate the performance of DCP under different scenarios using simulation using the ndnSIM [116] network simulator .

6.4.1 Basic Bottleneck Configuration

We compare DCP against an end-to-end protocol that behaves in the same way as TCP, where consumers can only infer congestion via a retransmission timeout and use AIMD window control to avoid congestion. Such a mechanism is used by most end-to-end protocols in ICN. The simulations last for the 60s, where the consumer will be issuing Interests nonstop as long as its congestion control protocol remits it. The network's topology is a single path of four nodes with a single consumer at one end sending Interests for content served at the other end. The size of the data packet of the content is fixed at 1024 Byte with an Interest size of 24 Byte. The router's queue is set to be equal to 60 packets. The bandwidth-delay product of the network is around 133 packets, including the queue's size. Since ndnSim ignores the processing delay per packet, the only constant end-to-end delay in the network would be the propagation delay and the transmission delay. Our test uses a TARGET delay of 50 ms, based on the network's parameters; this will allow enough packets to be built up in the queue without being dropped.

Figure 6.3b shows the size of the bottleneck's buffer for the whole duration of the simulation. For the TCP-based protocol, packet dropout is the only way to detect congestion in the network. Therefore, the consumer will keep increasing its sending rate until it causes a buffer overflow, which will result in a timeout event being triggered. Thus, the consumer reduces its sending rate by halving its congestion window. This process drains the queue empty for few seconds until it is filled again and the same cycle repeats. However, DCP(slow-start disabled) allows the consumer to increase its congestion window as long as the queue delay is less than the TARGET delay. As soon as the queue builds, the consumer slows down its sending rate. Once the queue delay reaches the TARGET delay

in both protocols, the consumer halts the window growth. This demonstrates the main strength of DCP as a congestion control for ICN by transmitting Interests at the bandwidth of the connection, without congesting the network, and without overflowing the bottleneck's queue. Finally, the congestion window reaches a steady-state value of 133 Interests as it can be seen in Figure 6.3a. The algorithm maintains this steady-state value for the duration of the connection with no timeout occurrence since slow-start was disabled as we are interested in only seeing. Fig. 6.3c shows the throughput for both protocols for the duration of the simulation. With a bottleneck link of speed 10 Mbps, the consumer in DCP was able to almost utilize the link's capacity with an average throughput of around 9.3 Mbps. As a result of the multiple packets dropout in the TCP based protocol, the consumer was only able to achieve an average throughput of 8.7 Mbps.

6.4.2 Multi-Flow Scenario

We evaluated LEDBAT's "latecomer advantage" in NDN architecture, which is similar to ITP, and showed how DCP with slow-start overcome this issue. The topology used in the scenario consists of two consumers (Consumer 1 and Consumer 2) and two producers (Producer 1 and Producer 2). The two consumers request different content files of a fixed data packet size of 1024 Bytes; each one is hosted by one of the producers. The queue size is set to 120 to allow packets not to be dropped due to buffer overflow. To simulate the latecomer advantage in ICN, the second consumer should start sending Interests when the queue starts to build up, and the first consumer begins to experience an increase in current delay. Based on our network parameter, that should be when the first consumer's window size is greater than 73, equal to the bandwidth-delay product of the network. This way, a later consumer accounts for the queuing delay of the first consumer

as its base delay measurement, resulting in a higher target delay. This leads to unfairness as the first consumer will have to back off as it can sense a true increase in the queue delay of the bottleneck. The simulation for this scenario runs for 60 seconds. Figure 6.4 shows the result of DCP with slow-start enabled. Once consumer 2 joins the network, it will go into the slow-start state. Consumer 2 doubles its sending window for every round-trip time until a timeout event is triggered during this state. Such a mechanism will drain the queue empty. Thus, allow consumer 2 to measure the correct base delay. Once consumer 2 measures the correct delay, fairness between the two consumers is achieved.

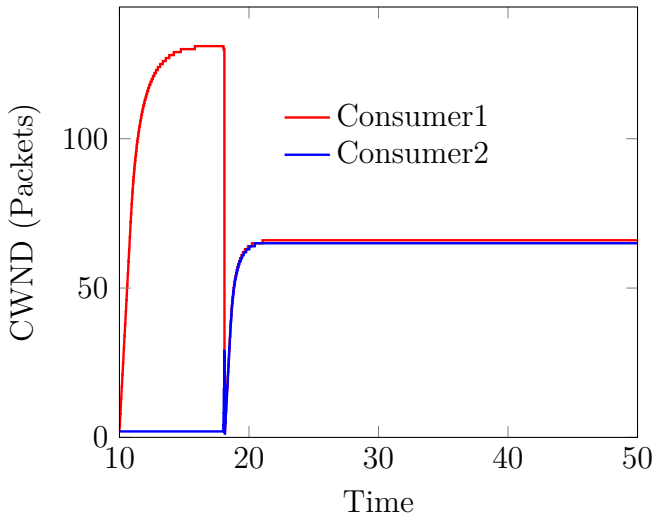


Figure 6.4: DCP performance under latecomer advantage

6.4.3 Latecomer's Disadvantage

We evaluated LEDBAT's "latecomer disadvantage" in NDN architecture. The topology of the network used to evaluate performance with caching consists of two consumers retrieving the same data. However, consumer 1 stops after retrieving around 100 data packets. Since caching is used, a consumer's 2 initial requests are

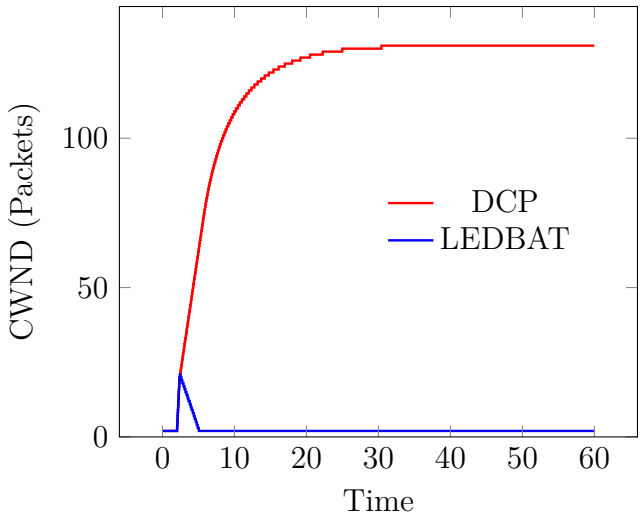


Figure 6.5: DCP and LEDBAT performance under caching

satisfied by the closest cache. The cache starts retrieving the data from the producer after consumer 2 exhausts its capacity. Figure 6.5 shows the increase of the congestion window for both LEDBAT and DCP for consumer2. It is clear that a pure implementation of LEDBAT in ICN architecture suffers from the "latecomer disadvantage". This is because consumer 2 initial requests were satisfied by a close cache; this resulted in a small base delay. Therefore, once consumer 2 starts retrieving the data from the producer, it will experience a high queuing delay even though the network is empty. However, DCP (slow-start disabled) overcomes this issue by using relative delay measurements of data packets. Consumer 2 concludes that an increase in the relative delay value is due to data being retrieved from a new distance caching site or producer, resulting in consumer 2 resetting its base delay history.

6.5 Conclusion

Many recent proposed solutions for congestion control in ICN suggest using RTT measurements as the primary indication of congestion in the network. Other solutions suggest a hop-by-hop approach either by shaping Interests or congestion signaling. This chapter introduced DCP, which adopts the congestion estimation algorithm of LEDBAT in the context of Interest-based architecture. DCP is based on a receiver driven, window-based approach to congestion control using measurements of delays along the forwarding path. Because of ICN's design nature compared to NDT and ITP, both protocols can benefit from the DCP congestion control algorithm in their design.

In this chapter, we analyzed the known "latecomer advantage" problem that exists in LEDBAT for ICN and how DCP slow-start can enable fairness. We also analyzed a new issue raised by the LEDBAT algorithm in ICN due to caching we call it "latecomer disadvantage." We showed how the use of relative measurement delay in DCP allows consumers in ICN architectures to overcome this problem.

Chapter 7

Conclusion

The original use of the Internet was to share expensive, limited resources by addressing endpoints using IP addresses and relying on end-to-end connections using TCP as the main architectural construct to implement reliable transfer of data. Such a design trace back to the 1960s and 70s. However, a connection is necessary for reliable data transfer only if the communicating processes have no means of discerning the structures of the data they are exchanging. In addition, the current Internet usage today has been shifted to access content rather than to share resources, but we still end up addressing endpoints in order to access distributed content.

Several Information Centric Networking (ICN) architectures have been proposed to cope with the current pattern usage of the Internet. The main approach of ICN architecture is moving the Internet toward a content distribution architecture by naming contents and services rather than endpoints. This enables packet forwarding based on the names of content or the service rather than the address of where they are hosted. These ICN architectures have produced important insight on how to provide content centricity at Internet scale. However, this comes at the cost of redesigning IP routing infrastructure and requiring a far more complex

network layer. In addition, transparent in-network caching is meant to be a key solution for these architectures. However, the solutions pose privacy concerns of consumers as intermediate routers and caches have access to the content being transferred to consumers.

In this dissertation, we proposed a new approach to improve the Internet at the transport layer. Our proposal supports reliable and secure content dissemination and caching on the Internet based on the use of decoding manifests that enable transparent caching of content with privacy and eliminate the need for end-to-end connections for reliable transport of content.

In Chapter 3, we outlined what a manifest is and how it is used to establish a nexus between remote processes in ITP without the need for connections. The manifest allows the content consumer and producer to share a common description of the structure of the content object that needs to be exchanged. The use of manifest in ITP, eliminates the need to maintain any transport state at content servers, intermediate routers, or caches and allows for all application data to be cached on the way to consumers using ITP caching proxies that can act as middleboxes.

In Chapter 4, we introduced Named-Data Transport, an approach that provides the same benefits of NDN and similar ICN architectures using the DNS and without the need for end-to-end connections or modifications to the IP routing infrastructure. In Chapter 5, we showed how a manifest could be used to prevent caching proxies in ITP and NDT from accessing cached contents by leveraging informational and computational asymmetry methods. And in Chapter 6, we introduced a new delay-based congestion-control algorithm for Interest-based architectures, as in NDN and NDT, that is based on forwarding delay measurements instead of round-trip time measurements. Such an algorithm allows consumers to

control the queuing delay in the network based on their application needs.

In closing, much has been written about the current IP Internet architecture's inability to support the emerging Internet content-oriented applications. Our description and analysis of the approaches we have proposed in this thesis demonstrate that the IP Internet can provide efficient support of such applications by means of connection-free reliable transport services. In addition to the benefits and results of a reliable connection, transport protocols can help address the many limitations of the IP multicast that have been pointed out over the years, which include the need to agree on multicast addresses of global scope and the use of multicast routing protocols, this is clearly an area that deserves future work.

Bibliography

[1] D.W. Davis, D.L.A. Barber, W.L. Price, and C.M. Solomonides, *Computer Networks and Their Protocols*, Wiley & Sons, 1979.

[2] C.A. Sunshine, "Interprocess Communication Protocols for Computer Networks," Tech. Report 105, Stanford University, Dec. 1975

[3] D.C. Walden, "A System for Interprocess Communication in a Resource Sharing Computer Network," *CACM*, April 1972.

[4] V.G. Cerf and R.E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Trans. Commun.*, May 1974.

[5] S. Carr, S.D. Crocker, and V.G. Cerf, "HOST-HOST Communication Protocol in The ARPA Network," *Proc. Spring Joint Computer Conference '70*, 1970.

[6] H. Zimmermann, "The CYCLADES End-to-End Protocol," *Proc. ACM Fourth Data Commun. Symposium*, Oct. 1975.

[7] V.G. Cerf, Y.K. Dalal, and C.A. Sunshine, "Specification of Internet Transmission Control Program," INWG Note 72, revised Dec.1974.

[8] V.G. Cerf, "Specification of Internet Transmission Control Program, TCP (Version 2)" March 1977.

[9] V.G. Cerf and J. Postel, “Specification of Internet Transmission Control Program, TCP (Version 3), Jan. 1978.

[10] C.A. Sunshine and Y.K. Dalal, “Connection Management in Transport Protocols," *Computer Networks*, 1978.

[11] J. Postel, “DoD Standard Internet Protocol," IEN 128, RFC 760, Jan. 1980.

[12] J. Postel, C.A. Sunshine, and D. Cohen, “The ARPA Internet Protocol," *Computer Networks*, 1981.

[13] J. Postel, “DoD Standard Transmission Control Protocol," IEN 129, RFC 671, Jan. 1980.

[14] M.S. Blumenthal and D.D. Clark, “Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World," *ACM Trans. on Internet Technology*, Aug. 2001.

[15] D.C. Walden, “Host-To-Host Protocols," in *Tutorial: A Practical View of Computer Communications Protocols* (J.M McQuillan and V.G. Cerf, Ed.s), pp. 172-204, IEEE, 1978.

[16] N. Bartolini, E. Casalicchio, and S. Tucc, “A Walk through Content Delivery Networks," *Proc. IEEE MASCOTS ‘03*, 2003.

[17] B. Zolfaghari et al., “Content Delivery Networks: State of the Art, Trends, and Future Roadmap," *ACM Computing Surveys*, April 2020.

[18] E. Nygren, R.K. Sitaraman, and J. Sun, “The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS Operating Systems Review*, Aug. 2010.

[19] J. Li, “On peer-to-peer (P2P) content delivery," *Peer-to-Peer Netw. Appl.*, 2008.

[20] BitTorrent. http://bittorrent.com

[21] P. Baran, “On Distributed Communications Networks," *IEEE Transactions on Communications Systems*, March 1964.

[22] J.H. Saltzer, “End-to-End Arguments in System Design," RFC 185, 1980.

[23] NSF Named Data Networking project. [Online].
Available: http://www.named-data.net/

[24] V. Jacobson et al., “Networking Named Content," *Proc. ACM CoNEXT ‘09*, Dec. 2009.

[25] Mosko âCCNx Manifest Specificationâ 2016

[26] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, “ Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service," *Proc. ACM PODC ‘20*, 2000.

[27] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, “Content-based Addressing and Routing: A General Model and Its Application," Tech. Report CU-CS-902-00, Univ. of Colorado, Jan. 2000.

[28] A. Carzaniga and A.L. Wolf, “Content-Based Networking: A New Communication Infrastructure," *Proc. Workshop on Infrastructure for Mobile and Wireless Systems*, 2002.

[29] T. Anderson et al., “A Brief Overview of the NEBULA Future Internet Architecture," *ACM SIGCOMM CCR*, 2014.

[30] G. Papastergiou et al., “De-ossifying the Internet Transport Layer: A Survey and Future Perspectives," *IEEE Communications Surveys and Tutorials*, Nov. 2016.

[31] M. Polese et al., “A Survey on Recent Advances in Transport Layer Protocols," *IEEE Communications Surveys and Tutorials*, Aug. 2019.

[32] B. Ford, “Structured Streams: A New Transport Abstraction, *Proc. ACM SIGCOMM ‘07*, Aug. 2007.

[33] A. Ford et al., “Architectural guidelines for multipath TCP development," IETF, RFC 6182, Mar.2011.

[34] Y. Gu and R. Grossman, “ UDT: UDP-Based Data Transfer for High-Speed Wide Area Networks," *Computer Networks*, Elsevier, Volume 51, Issue 7, 2007.

[35] R. Stewart, “RFC 4960: Stream Control Transmission Protocol (SCTP)," IETF, 2007.

[36] A. Langley et al., “The QUIC Transport Protocol: Design and Internet-Scale Deployment," *Proc. ACM SIGCOMM ‘17*, Aug. 2017.

[37] A. Agrawal, “Xavier: A Reinforcement-Learning Approach to TCP Congestion Control," Technical Report, Stanford University, 2016.

[38] N. Jay et al., “A Deep Reinforcement Learning Perspective on Internet Congestion Control," *Proc. 36th Int’ Conf. on Machine Learning*, 2019.

[39] K. Winstein and H. Balakrishnan, “TCP ex Machina: Computer-Generated Congestion Control," *Proc. ACM SIGCOMM ‘13*, 2013.

[40] N.A. Lynch, Y. Mansour, and A. Fekete, “Data link layer: Two Impossibility Results," *Proc. ACM PODC ‘88*, 1988.

[41] J.M. Spinelli, “Reliable Communication on Data Links," LIDS-P-1844, MIT, Dec. 1988.

[42] D. Wand and L.D. Zuck, “Tight Bounds for the Sequence Transmission Problem," *Proc. ACM PODC ‘89*, Aug. 1989.

[43] Y. Afek et al., “Reliable Communication over Unreliable Channels," *JACM*, Nov. 1994.

[44] J. Rosenberg, “UDP and TCP as the New Waist of the Internet Hourglass," IETF Internet Draft, draft-rosenberg-internet-waist-hourglass-00, 2008.

[45] S. Cheshire, J. Graessley, and R. McGuire, “Encapsulation of TCP and other transport protocols over UDP," Internet Draft, Jul. 2013.

[46] B. Trammell, M. Kuehlewind, E. Gubser, and J. Hildebrand, “A New Transport Encapsulation for Middlebox Cooperation,â *Proc. IEEE CSCN ‘15* Oct 2015.

[47] E.K. Lua et al., “A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Survey & Tutorial*, March 2004.

[48] J.J. Garcia-Luna-Aceves, “System and Method for Discovering Information Objects and Information Object Repositories in Computer Networks,” U.S. Patent 7,162,539, January 9, 2007.

[49] M. Nowlan et al, “Fitting Square Pegs Through Round Pipes: Unordered Delivery Wire-Compatible with TCP and TLS," *USENIX NSDI ‘12*, 2012.

[50] D.K. Gifford, “Replica Routing," U.S. Patent 6,052,718, April 18, 2000.

[51] J. Raju, J.J. Garcia-Luna-Aceves, and B. Smith, “System and Method for Information Object Routing in Computer Networks," U.S. Patent 7,552,233, June 23, 2009.

[52] M. Gritter and D. Cheriton, “An Architecture for Content Routing Support in The Internet," *Proc. USENIX Symposium on Internet Technologies and Systems*, Sept. 2001.

[53] D. Florez-Rodriguez et al., “Global Architecture of the COMET System," Seventh Framework STREP No. 248784, 2013.

[54] FP7 COMET project. [Online]. Available: http://www.comet-project.org/

[55] FP7 CONVERGENCE project. [Online]. Available: http://www.ictconvergence.eu/

[56] T. Koponen et al., “A Data-Oriented (and Beyond) Network Architecture,” *Proc. ACM SIGCOMM ‘07*, Aug. 2007.

[57] FP7 PSIRP project. [Online]. Available: http://www.psirp.org/

[58] FP7 PURSUIT project. [Online]. Available: http://www.fp7-pursuit.eu/PursuitWeb/

[59] FP7 SAIL project. [Online]. Available: http://www.sail-project.eu/

[60] FP7 4WARD project. [Online]. Available: http://www.4ward-project.eu/

[61] G. Xylomenos et al., “Caching and Mobility Support in a Publish-Subscribe Internet Architecture," *IEEE Communications Magazine*, July 2012

[62] NSF Mobility First project. [Online]. Available:http://mobilityfirst.winlab.rutgers.edu/.

[63] I. Seskar et al., “Mobility First Future Internet Architecture Project," *Proc. AINTEC ‘11*, Nov. 2011.

[64] D. Han et al., “XIA: Efficient Support for Evolvable Internetworking," *Proc. USENIX NSDI ‘12*, 2012.

[65] B. Tremblay et al., “(D.A.3) Final Harmonised SAIL Architecture," Report FP7-ICT-2009-5-257448-SAIL/D-2.3, Feb. 2013.

[66] E.G. AbdAllah, H.S. Hassanein, and M. Zulkernine, “A Survey of Security Attacks in Information-Centric Networking," *IEEE Communications Surveys & Tutorials*, 2015.

[67] B. Ahlgren et al., “A Survey of Information-Centric Networking," *IEEE Communications Magazine*, July 2012, pp. 26–36.

[68] M. Ambrosin et al., “Security and Privacy Analysis of National Science Foundation Future Internet Architectures," *IEEE Communications Surveys & Tutorials*, 2018.

[69] G. Xylomenos et al., “A Survey of Information-centric Networking Research," *IEEE Communication Surveys and Tutorials*, July 2013.

[70] M.F. Bari et al., “A Survey of Naming and Routing in Information-Centric Networks," *IEEE Communications Magazine*, July 2012, pp. 44–53.

[71] G. Carofiglio et al., “Enabling ICN in the Internet Protocol: Analysis and Evaluation of the Hybrid-ICN Architecture," *Proc. ACM ICN ‘19*, Sept. 2019.

[72] L. Muscariello et al., “Hybrid Information-Centric Networking," IETF Internet Draft, Oct. 2019.

[73] A. Dabirmoghaddam, M.M. Barijough, and J. J. Garcia-Luna-Aceves, "Understanding Optimal Caching and Opportunistic Caching at the Edge of Information-Centric Networks," *Proc. ACM ICN '14*, Sept. 2014.

[74] A. Dabirmoghaddam, M. Dehghan, and J.J. Garcia-Luna-Aceves, "Characterizing Interest Aggregation in Content-Centric Networks," *Proc. IFIP Networking 2016*, May 17–19, 2016.

[75] S. K. Fayazbakhsh et al., "Less Pain, Most of the Gain: Incrementally Deployable ICN," *Proc. ACM SIGCOMM '13*, 2013.

[76] J. J. Garcia-Luna-Aceves, "Name-Based Content Routing in Information Centric Networks Using Distance Information," *Proc. ACM ICN '14*, Sept. 2014.

[77] C. Ghasemi et al., "MUCA: New Routing for Named Data Networking," *Proc. IFIP Networking '18*, May 2018.

[78] E. Hemmati and J.J. Garcia-Luna-Aceves, "A New Approach to Name-Based Link-State Routing for Information-Centric Networks," *Proc. ACM ICN '15*, Sept. 2015.

[79] L. Wang et al., "A Secure Link State Routing Protocol for NDN," *IEEE Access*, March 2018.

[80] J.J. Garcia-Luna-Aceves and M. Mirzazad-Barijough, "Content-Centric Networking Using Anonymous Datagrams," *Proc. IFIP Networking '16*, May 2016.

[81] J.J. Garcia-Luna-Aceves, M. Mirzazad-Barijough, and E. Hemmati, "Content-Centric Networking at Internet Scale through The Integration of Name Resolution and Routing," *Proc. ACM ICN '16*, Kyoto, Japan, Sept. 2016.

[82] J.J. Garcia-Luna-Aceves and M. Mirzazad-Barijough, “Efficient Multicasting in Content-Centric Networks Using Datagrams,” *IEEE Globecom ‘16*, Dec. 2016.

[83] J.J. Garcia-Luna-Aceves, “New Directions in Content Centric Networking,”*Proc. IEEE CCN ‘15* Oct. 2015.

[84] J.J. Garcia-Luna-Aceves, Q. Li, and Turhan Karadeniz, “CORD: Content Oriented Routing with Directories," *Proc. IEEE ICNC ‘15*, Feb. 2015.

[85] T. Dierks, “The Transport Layer Security (TLS) Protocol Version 1.2," 2008.

[86] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta and M. Hafeez. Explicit Trusted Proxy in HTTP/2.0. IETF Internet-Draft, 2014.

[87] R. Peon. Explicit proxies for HTTP/2.0. IETF Informational InternetDraft, 2012.

[88] J. Sherry, C. Lan, R. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. Proc. ACM SIGCOMM, 2015.

[89] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. Rodriguez, and P. Steenkiste. multi- ´ context TLS (mcTLS): Enabling secure in-network functionality in TLS. Proc. ACM SIGCOMM, 2015.

[90] S. Sevilla, J.J. Garcia-Luna-Aceves, and H. Sadjadpour, “GroupSec: A New Security Model for the Web," *Proc. IEEE ICC ‘17*, 2017.

[91] S. Arianfar et al., “On Preserving Privacy in Content-Oriented Networks," *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*, 2011.

[92] R. Atkinson, S. Bhatti, and S. Hailes. “ILNP: Mobility, Multi-homing, Localised Addressing and Security through Naming," *Telecommunication Systems*, 2009.

[93] W.M.Eddy, “At What layer Does Mobility Belong?," *IEEE Communications Magazine*, 2004.

[94] Z. Gao, A. Venkataramani, and J.F. Kurose, “Towards a Quantitative Comparison of Location-Independent Network Architectures," *ACM SIGCOMM CCR*, 2014.

[95] D. Le, X. Fu, and D. Hogrefe, “A Review of Mobility Support Paradigms for the Internet, *IEEE Communications Surveys & Tutorials*, 2006.

[96] E. Perera, V. Sivaraman, and A. Seneviratne, “Survey on Network Mobility Support," *ACM SIGMOBILE Mobile Computing and Communications Review*, 2004.

[97] D. Saha et al., “Mobility Support in IP: A Survey of Related Protocols," *IEEE Network*, 2004

[98] S. Sevilla and J.J. Garcia-Luna-Aceves, “Freeing The IP Internet Architecture from Fixed IP Addresses," *Proc. IEEE ICNP ‘15*, Nov. 2015.

[99] S. Sevilla and J.J. Garcia-Luna-Aceves, “A Deployable Identifier-Locator Split Architecture," *Proc. IFIP Networking ‘17*, June 2017.

[100] S. Sevilla, P. Mahadevan, and J.J. Garcia-Luna-Aceves, “iDNS: Enabling Information Centric Networking through the DNS," *Proc. IEEE INFOCOM Workshop on Name-Oriented Mobility ‘14*, 2014.

[101] E. Demirors and C. Westphal, “DNS++: A Manifest Architecture for Enhanced Content-Based Traffic Engineering," *Proc. IEEE GLOBECOM ‘17*, 2017.

[102] A. Eriksson and A. Mohammad Malik, “A DNS-Based Information-Centric Network Architecture Open to Multiple Protocols for Transfer of Data Objects, *Proc. IEEE ICIN ‘18*, 2018.

[103] S. Arianfar, P. Nikander, L. Eggert, and J. Ott, “Contug: A Receiver-Driven Transport Protocol for Content-Centric Networks," *Proc. IEEE ICNP ‘10* (Poster session), 2010.

[104] G. Carofiglio, M. Gallo, and L. Muscariello, “ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking," *Proc. IEEE NOMEN ‘12*, March 2012.

[105] N. Rozhnova et al., , Y. Wang, A. Narayanan, D. Oran, and I. Rhee. “An Improved Hop-by-hop Interest Shaper for Congestion Control in Named Data Networking," *Proc. ACM ICN ‘13*, 2013.

[106] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A Case for Stateful Forwarding Plane. Tech. Report NDN-0002, July 2012.

[107] K. Schneider et al., , C. Yi, B. Zhang, L. Zhang “A Practical Congestion Control Scheme for Named Data Networking," *Proc. ACM ICN ‘16* , 2016.

[108] S. Braun et al., “An Empirical Study of Receiver-based AIMD Flow Control for CCN," *Proc. IEEE ICCCN ‘13*, 2013.

[109] T. Berners-Lee, R. Fielding, L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, 2005

[110] I. Moiseenko “Fetching content in Named Data Networking with embedded Manifests" 2014.

[111] M. Mosko, I. Solis, C. Wood, “Content-Centric Networking (CCNx) Messages in TLV Format," RFC 8609, July 2019

[112] V. Jacobson, “Congestion Avoidance and Control, *Proc. ACM SIGCOMM ‘88*, Aug. 1988.

[113] Q. Chen et al., “Transport Control Strategies in Named Data Networking: A Survey," *IEEE Communications Surveys and Tutorials*, 2016.

[114] X. Xu, Y. Jiang, E. Katz-Bassett, D. Choffnes, and R. Govindan, “Investigating Transparent Web Proxies in Cellular Networks," *PAM 2015*, NY, USA

[115] F. Urbani, W. Dabbous, and A. Legout, NS3 DCE CCNx quick start, INRIA, Nov. 2011.

[116] NS-3 based Named Data Networking (NDN) simulator [Online]. Available: https://ndnsim.net/current/index.html

[117] I. Moiseenko and L. Zhang. “Consumer-producer API for Named Data Networking," *Proc. ACM ICN ‘14*, 2014.

[118] P. Vixie et al., “Dynamic Updates in the Domain Name System," IETF RFC 2136, 1997.

[119] L. Saino, C. Cocora, and G. Pavlou, “CCTCP: A Scalable Receiver-Driven Congestion Control Protocol for Content Centric Networking," *Proc. IEEE ICC ‘13*, 2013.

[120] A. Albalawi and J. J. Garcia-Luna-Aceves, “A Delay-Based Congestion-Control Protocol for Information-Centric Networks," *Proc. IEEE ICNC ‘19*, 2019.

[121] A. Albalawi and J.J. Garcia-Luna-Aceves, “NDT ns-3 Simulator." Available at: https://github.com/aalbalaw/NDT.

[122] S. Ariyapperuma and C. Mitchell, “Security vulnerabilities in DNS and DNSSEC," *Proc. IEEE ARES ‘07*, April 2007.

[123] D. E. Eastlake 3rd, “Domain Name System Security Extensions," RFC 2535, 1999.

[124] Y. Wu, J. Tuononen, and M. Latvala, “Performance Analysis of DNS with TTL Value 0 as Location Repository in Mobile Internet," *IEEE WCNC ‘07*, March 2007.

[125] X. Chen et al., “Maintaining Strong Cache Consistency for the Domain Name System," *IEEE Transactions on Knowledge and Data Engineering*, August 2007

[126] P. Mockapetris, “Domain Names – Implementation and Specification," RFC 1035, Nov. 1987.

[127] S.E. Deering and D.R. Cheriton, “Multicast Routing in Datagram Internetwork and Extended LANs," *ACM TOCS*, May 1990.

[128] B.N. Levine et al., “Consideration of Receiver Interest for IP Multicast Delivery," *Proc. IEEE Infocom ‘00*, March 2000.

[129] R. Pries, Z. Magyari, and P. Tran-Gia, “An HTTP Web Traffic Model based

on the Top One Million Visited Web Pages," *Proc. IEEE Euro-NF Conference on Next Generation Internet '12*, 2012.

[130] P. Gusev and J. Burke, "NDN-RTC: Real-Time Video conferencing over Named Data Networking," *Proc. ACM ICN '15*, Sept. 2015.

[131] V. Jacobson et al., "VoCCN: Voice-over Content-Centric Networks," *Proc. ACM ReArch '09*, Dec. 2009.

[132] A. Stubblefield and D. Wallach, "Dagster: Censorship-Resistant Publishing Without Replication," Rice University, Dept. of Computer Science, Tech. Rep. TR01-380, 2001.

[133] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698, 2012.

[134] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, LEDBAT: "The New BitTorrent Congestion Control Protocol," *IEEE ICCCN '10*, Aug. 2010.

[135] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internets with heterogeneous transmission media," *Seventh International Conference on Network Protocols*, Toronto, Canada, 1999

[136] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, December 2012.

[137] G. Carofiglio, L. Muscariello, D. Rossi and S. Valenti, "The Quest for LEDBAT Fairness," *GLOBECOM 2010*, Miami, FL, 2010