submitted at the University of Stuttgart

Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr.

Design and Implementation of a QoS-Aware Communication Protocol for Named Data Networks

Mustafa Karadeniz

Course of Study:	Computer Science
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Dr.-Ing. Mohamed Abdelaal
Commenced:	2018-11-02
Completed:	2019-05-02
CR-Classification:	C.2.4, I.4

Abstract

In the recent decade, a significant discrepancy has been identified between the IP-based Internet architecture and its primary use. Despite being fundamentally designed for facilitating ubiquitous interconnectivity between communication endpoints, the TCP/IP protocol suite is overwhelmingly used for content distribution. In fact, the IP-based architecture has several limitations when dealing with large-scale content distribution, including multicasting efficiency, handshaking overhead, mobility, routing scalability, and security. These limitations, facing the IP paradigm, have fostered the design of novel Internet architectures to meet the accelerating expansion in consumer Internet traffic.

Named Data Networking (NDN) is a data-centric Internet architecture solution. It has recently received much attention as an efficient alternative Internet architecture to the IP-based Internet architecture. Generally, NDN represents a clean-slate receiver-oriented design which utilizes the Interest/Data model to retrieve data from possible source nodes [ZAB+14]. NDN architecture is based on four main concepts: content forwarding, in-network caching, multicasting, and built-in security at the data level. In-network caching and multipath forwarding strategies are the two significant features of NDN architecture. Content forwarding directly leverages data names—that carry application semantics—to exchange two types of packets, i.e. Interest and Data.

In fact, the current routing algorithms of NDN networks do not make use of the unique NDN primitives, such as in-network caching and multipath forwarding. Therefore, this thesis is devoted to the development of a novel routing algorithm that improves the performance of NDN networking in terms of overall network resource-usage efficiency, data retrieval delay, and bandwidth. Aside from routing, the caching strategy of NDN networks completely ignores the topology structure of the network, thus the cache memories are exhausted in vain. Accordingly, a challenge of improving the routing strategy together with increasing the caching efficiency emgeres.

To tackle this challenge, three different methods to optimize the performance of the current caching capability and the forwarding mechanism of NDN are proposed, namely the Hop Distance Aware Caching (HDAC) strategy, the Segmented Data Aware Routers (SDAR) method, and the Knowledge Sharing-Based Forwarding (KSBF) strategy. The purposed KSBF strategy employs statistical information shared by neighbor routers and makes a forwarding decision for an Interest packet based on this knowledge. In other words, each NDN router selects the next hop based on what is called the "most probable path" rather than the best path determined by computing the Dijkstra algorithm. The process of sharing of statistical information is mainly controlled by the NDN forwarding mechanism. The SDAR method focuses on reducing the segmented data retrieval latency. In addition, a novel caching policy, i.e.

HDAC strategy, is proposed to make a decision of whether to store the incoming packets in the memory based on the distance between the various NDN routers. Through evaluating the performance of the proposed methods, it is found that they significantly reduce the data retrieval time, the total resource usage and the overall bandwidth consumption.

Contents

List of Figures

List of Tables

List of Abbreviations

NDN	Named Data Networking
CCN	Content-Centric Networking
QoS	Quality of Service
NFD	Named Data Networking Forwarding Daemon
NLSR	Named-data Link State Routing Protocol
PIT	Pending Interest Table
CS	Content Store
FIB	Forwarding Information Base
MPP	Most Probable Path
NAT	Network Address Translation
BRS	Best Route Strategy
NACK	Negative Acknowledgement
HDAC	Hop Distance Aware Caching
SDAR	Segmented Data Aware Routers
KSBF	Knowledge Sharing Based Forwarding

Chapter 1

Introduction

In this chapter, a brief information about Named Data Networking is given first. Then, the problem statement explains the problems of the current NDN implementation.

1.1 Motivation

Named Data Networking (NDN) is a new network architecture based on naming the exchanged data in network communication. Today's IP-based network is built on *where* the data is located. In other words, the location of the data is a major consideration to exchange information. The IP-based network packet includes a destination address and a source address to request a data. These address fields are necessary information in order to forward a data exchange request.

About 20 years ago, IP-based communication architecture has been very successful to supply global -scale communication, like point-to-point communication or basic Internet usage over the world. However, by the huge improvement and development in today digital world, we produce globally $50,000$ GB data per second [9] in 2018. Although it was just 100 GB data in 2002. That means the amount of data that we have is increased so the amount of data exchange is increased. For example, in 2018, the number of videos watched per day is 5 billion [16]. The statistical information belongs to one of the popular video sharing websites, YouTube. Therefore, the IP-based communication architecture is entirely focus on the location of data, on the other hand, people in today's world just want to access the content itself as fast as possible.

Nowadays, consumers are more interested in *what* data they want instead of *where* it is located. In other words, people care most about the content rather than the location of the content. Thus, the location of data is no longer important to them. In Figure 1.1, the problem of IP-based networks is represented with a simple scenario. In this scenario, To satisfy the desired content X, the current network, first of all, has to establish a stable connection between the application and the provider Y. However, the only desire for the application side is to retrieve the content asap, not having a stable connection or other protocols. For this reason, NDN provides an efficient solution than the location-centric IP-based network for this kind of communication model [1].

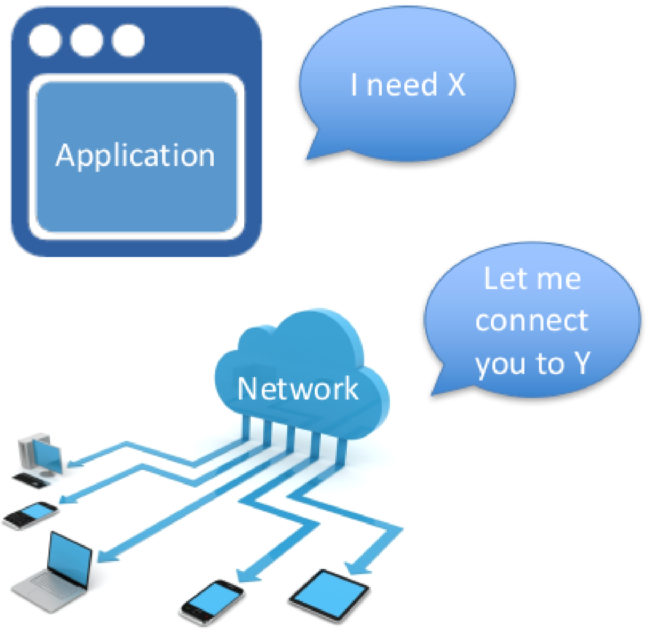


Figure 1.1: The fundamental problem of the current Internet Architecture [23]

First of all, the network packet in NDN only carries a unique name to identify the requested data. There is no source or destination address inside the packet. The routing and forwarding of the network packets are done completely by the NDN architecture itself. The User needs to specify what data they want and NDN takes care of searching, finding and delivering that data back to the user. Aside from the addressability, NDN also provides an efficient solution for data security. All data in NDN is signed by producers before publishing the data to the network. Later, the consumers verify the signed data to make sure it is original data. However, IP-based networks focus on securing the communication channels instead of data itself.

IoT (Internet Of Things) applications are one of the examples which faces inefficient solutions due to the use of the IP-based protocols [22]. IoT consists of multiple different interconnected computing devices to supply valuable information for the monitor and control applications. Smart home automation and smart traffic control system are popular examples of IoT applications. These applications contain many sensors and actuators to process various tasks like monitoring or controlling the ambient environment parameters (temperature, lights, heating system, etc.).

There are several problems to use the IP-based model in IoT applications. The first one is that the embedded devices of an IoT application may generally go into the sleeping mode so that they can save energy and last longer. To realize this scenario in the IP-based network, the data consumer and the data provider must be active at the same time in order to exchange information. However, in IoT applications, it is sometimes hard to keep the communication stable due to the mentioned above reasons. On the other hand, NDN offers in-network caching capability which temporarily stores the received contents and each interim routers may pro-

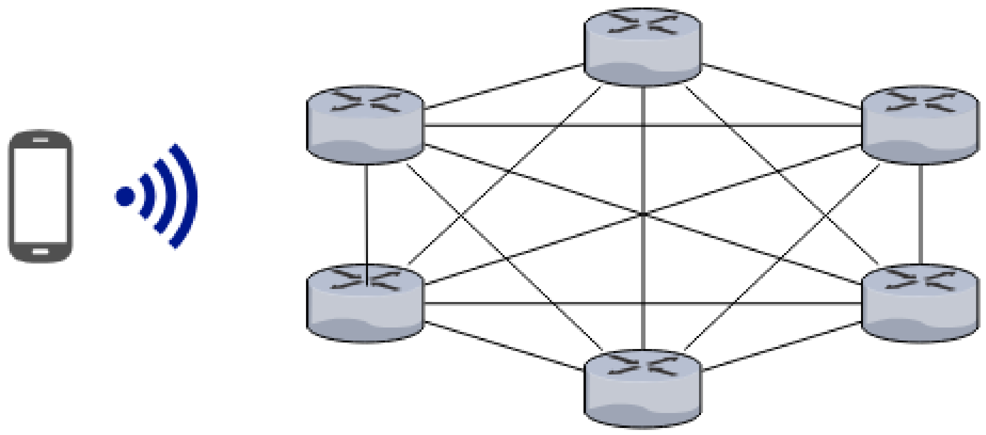


Figure 1.2: Mesh Topology

vide those contents to the user. Therefore, the client does not need to establish an explicit connection with a router to ask for a piece of information.

Another problem of using IP-based protocols in IoT applications is the routing protocol in IoT mesh networks (which is shown in Figure 1.2) which are one of the topologies of typical IoT networks. The routing information must be maintained for each host in mesh topologies. For example, with IP-based routing protocols, the address allocation process may have to be repeated when the topology has some changes such as adding new nodes into the topology. The routing system must handle the address allocation to forward the requests. Maintaining the routing configuration in every node in mesh topology consumes significant memory. However, in NDN there is no need for the address allocation process because it is not used in NDN.

The following section continues to present the overview of the NDN architecture in more detail.

1.2 Problem Statement

Although NDN networks have better efficiency in terms of data delivery than IP-Networks [1], the current implementation of in-network caching and routing algorithm in NDN networks has overheads and resource usage limitations such as the way of using routers' cache and the routing algorithm decision to forward network packets. To improve the efficiency of routing algorithm and in-network caching capability in NDN, three different ideas were implemented and will be explained in this thesis.

First, the current implementation of NDN forwarding mechanism allows every node—along the reverse path of a request packet—to store the incoming data based on the *Content Store admission* and the *replacement* algorithm. This property of NDN is called in-network caching. Indeed, the reason for keeping a copy of the requested data in every node is to increase the chance of data hit in these nodes for incoming interests and this design is implemented taking into consideration an optimistic assumption that cached data will be requested in the future before it is removed from the cache. The problem is that the current adopted Caching Strategy in NDN does not consider the Network Topology. This might cause useless work and unnecessary caching activities. However, the in-network caching capability can be improved and used more efficiently by considering the latency between hops. To address this issue, the idea is to save incoming data when only the data is far enough from the previously cached node. The distance to decide will be more explained in the later section.

Second, there is a separate field at the end of the name convention in NDN to tag segment number for large data. A large data can be divided into multiple chunks and each chunk has own unique name. Then, each segment is retrieved individually by sending an Interest having its name. When a segment is requested from the network, especially for the segmented video data, the requester most probably will wish to get the rest of the segments. However, she/he has to send a separate Interest for each segment one by one to get the entire video. This means extra Interest bandwidth on the network just to obtain one data. Based on this knowledge, the simple idea was presented to overcome this unnecessary bandwidth usage. The design is that the related segments can be sent to the client without a request when the client sends an Interest for an individual segment. Therefore, the interests of the remaining segments will not come along the same path to the data producer and they will satisfied in early routers.

The last point is the routing and forwarding unit. NDN uses the Named-data Link State Routing protocol (NLSR) as a routing protocol. NLSR computes the best path for each data to its destination. Routers always follow the best path to forward incoming interest and if that path does not work to get the data, then NLSR chooses the second best-path to achieve its goal. In fact, the data might be around that router but not on the route of the best path,therefore if the same data requested from some other clients, the interest to request the data needs to go along the best path to get the data from its origin router The problem is here, the current implementation of NLSR does not exploit the unique features of NDN, such as in-Network Caching. Multi-Path forwarding. The new approach is here that if routers share their knowledge about their content store with their neighbors, then this knowledge can be used for forwarding interests. Therefore, these three points are valuable for us to investigate and research.

1.3 Contributions

To tackle the problems discussed in the previous section, we dedicate the work in this thesis to develop innovative solutions that improve the performance of NDN networks. Specifically, the thesis provides three different contributions: namely, *distance-aware caching strategy*, *segmented data policy*, and *dynamic forwarding*. The first contribution of this thesis is developing a novel in-networking caching strategy through which less caching activities for the overall network are required. During the return of a data packet, instead of caching the data packet in every hop, it is cached in hops which are 100 ms far from the previous node that have the copy of the data. As a result, the less number of caching activities will cause less process overhead and less power consumption. The future chapters show the details about the design, implementation and the result of this architecture.

The second contribution of this thesis is new policy for large segmented data. In NDN, each segmented data has own unique name prefix and to get each segment, one has to send a separate interest for every segment. The idea is here to predict the user intention for the requested segment based on the data type, size, and priority and then the decision algorithm running on each router sends blindly rest of segments which are belong to the same data. The design technique is similar to *SpatialLocality* in CPU architecture design. That is, if consumer wants to get the some segments of a data, the decision algorithm checks data details like type, size and then it decides to send other segments of the data to the consumer after sending the requested segment. The evaluation for this study will in the following chapters.

The last contribution is the design and implementation of a new routing algorithm on the top of the current routing algorithm. This new routing is based on the data statistics and sharing of these statistics with neighbors. In fact, the forwarding decision is made according to those statistics. If, however, the requested data could not found, then the default routing algorithm is used to search the data from the network. To apply this routing policy, every router creates a new major statistic table for the incoming data. The name prefix of each processed data going downstream is saved with some detailed information in this table 4.1. Then, the information which is gathered sends to the neighbor router. Moreover, each router calculates a probability value based on the MPP table entries for an incoming interest to forward it to the the next hop. Performance and overheads will be discussed more in the following sections.

1.4 Thesis Overview

The thesis is organized as follow:

Chapter 1 provides an overview of the tackled problem and the contributions developed throughout the thesis.

Chapter 2 provides background information about NDN. The chapter explains how NDN communication architecture works and which parts of it are different than the IP-based network. NDN benefits will be also presented in this chapter. In addition, the overview of the previous studies in the same field will be discussed in this section. Their strategies and some results will be discussed in more detail.

Chapter 3 presents the purposed three new ideas and how they will be integrated into existing NDN architecture. This chapter explains how the existing design works and what are the new methods to change or modify the existing system. The three models which are Caching Strategy, Segmented Data Aware Routers, and Forwarding Strategy will be presented and explained in more detail.

Chapter 4 presents the design and implementation details of the system. The three new methods are showed how they are designed and implemented by providing pseudocodes for the major algorithms and data structures. One can easily understand the design and implementation details of these new system by reading this chapter.

Chapter 5 provides the evaluation and results and shows the outcomes of the new design. The chapter begins with the environmental setup. The setup section presents which simulation environment is used and which kind of tools are used to test the newly developed features. Then, each model is investigated one by one by showing the result graphics and outcomes. Of course, the drawbacks of the each model are also mentioned in this chapter.

Chapter 6 summarizes the thesis and gives possible future extensions in the NDN architecture.

Chapter 2

Background

In this chapter, the necessary background knowledge and important details about the NDN architecture are presented. There are several major benefits in NDN and they are also mentioned in this part. In addition, related studies will be investigated by explaining their methods and drawbacks at the end of this chapter.

2.1 Content-Centric Networking

Content-Centric Networking is a communication architecture which is based on named data rather than IP address [1]. It is an alternative option for Today's IP Networking. The Internet which we use currently is mostly the host-centric networking and every host in that network is identified by a unique address. A network packet in CCN, on the other hand, is represented by a unique name. A consumer requests the content by sending its name to the network. The network is responsible for searching, finding and returning the content back to the consumer. The simple representation of this method is shown in Figure 2.1. The data exchanged is done by using two special network packets which are called *Interest* and *Data*. The user simply presents the name of desired content in Interest packet and send it to the network. Then, CCN returns a Data packet containing the requested content back to the user.

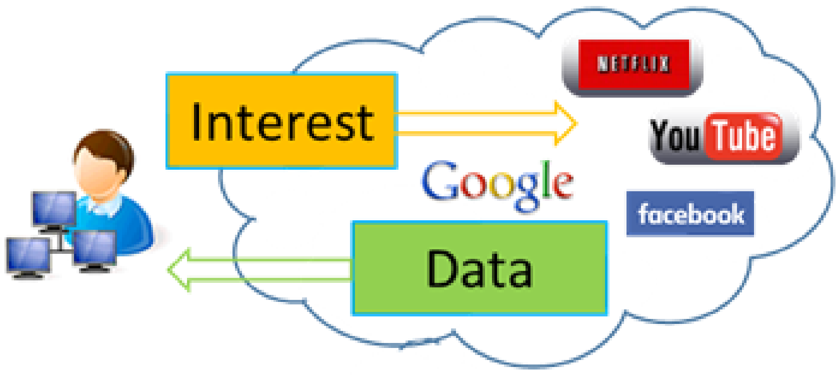


Figure 2.1: General Idea for Network Protocol based on Named Data [6].

The content can be found any location in the network thanks to network caching capability. The network caching technique is a new feature in CCN and one of the major advantages of this [3] is to reduce the transmission of the data, which is requested by multiple consumers over the network. A network element having in-network caching capability can store the received content for the future requests. Another point of CCN is that the same requests from multiple consumers can be aggregated in a router to send only one request to the network. Thus, it contributes to the network by decreasing the number of requests for the same content. When these points are considered, CCN seems significant importance over the current IP host-centric communication architecture.

2.2 Named Data Networking

Named Data Networking [1] is a primary research area among the present CCN architectures. NDN is a content-centric and receiver initiated communication architecture based on named data. The main focus of this type of network is the name of the content. The user only presents the name of the content and the desired content will be automatically returned back to the user. There are no other restrictions like providing a destination address. Moreover, the name of the content can be any type of data such as a video file, PDF file and any type of pictures.

NDN presents a different logic in network service. In the current host-centric network service packets are delivered to a given destination address, whereas in NDN packets are fetched according to a given name. It names data itself rather than the container that has the data. Thus, you can only express what you want from the network, and the rest will be handled by the NDN architecture.

Simple communication flow in NDN is listed below and also it is shwon in Figure 2.2:

1. A consumer sends out an Interest packet containing the name for the desired data to the network.
2. A router, first, takes the Interest packet and saves the requester interface information so that it can return the data back to the requester.
3. Then it checks its content store (cache storage) to decide whether the data already exists or not.
4. If the data is not available inside its content store, then it forwards the Interest packet to the next router. Forwarding strategy decides the next hop to which the Interest should be forwarded based on the routing protocol.
5. The Interest is carried and forwarded through the network until a router has the requested data in its content store.
6. Finally, when the data is found in a router content store, it is sent back to the consumer

by using the reverse path.

During this entire communication process, IP address or any host address are not used.

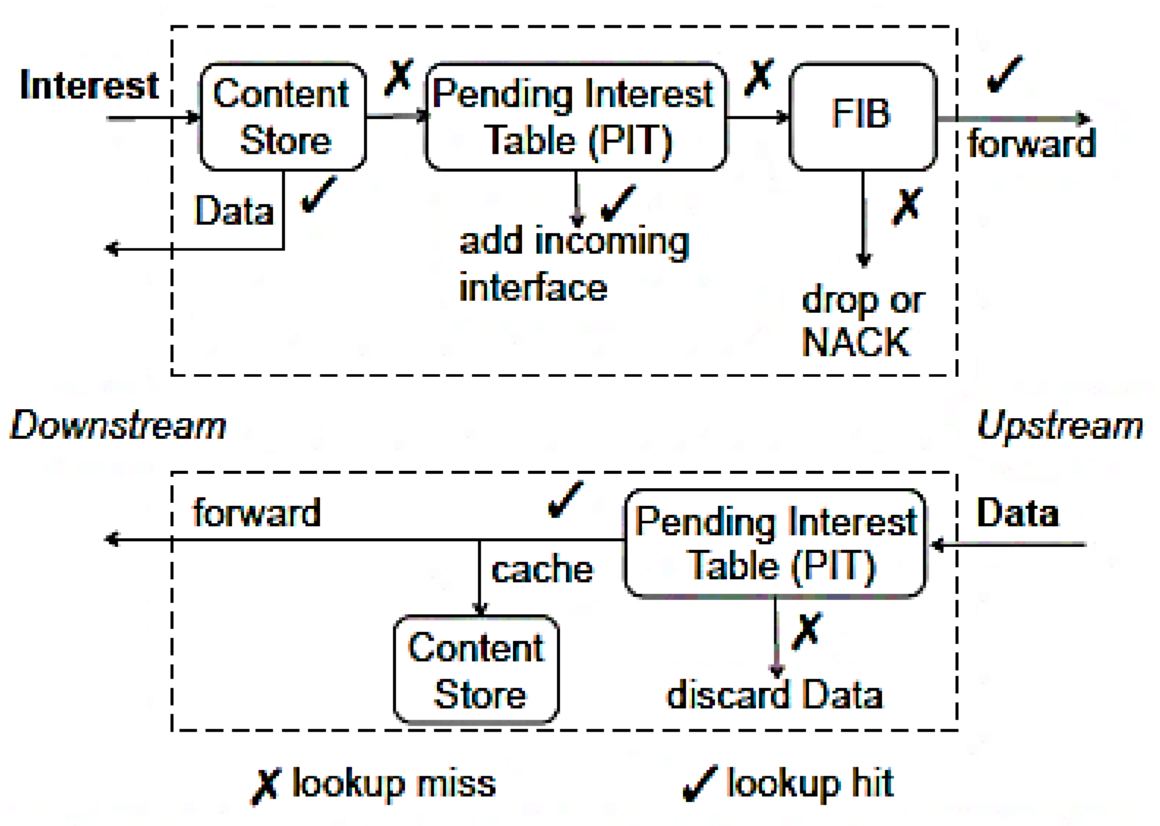


Figure 2.2: Interest and Data Forwarding Process in NDN [14].

There are two packet types in NDN. They are *Interest* and *Data* packets which are shown in Figure 2.3. A *Interest* packet contains a unique name to identify the requested data. A Data packet has also the same name. Hence, the network element can match the request packet with the corresponding data packet by comparing their names. *Nonce* field in Interest packet is a random number which uniquely identify the Interest packet together with *Name*. It is used to detect looping interests. *Signature* is used as a digital signature of the content which provides a secure communication.

To accomplish the implementation of this new communication architecture, NDN requires several important data structures like Content Store, Pending Interest Table, and Forwarding Information Base. These tables are essential during the packet processing logic in NDN. The following part explains more about these data structures which are currently used in NDN implementation.

- **Pending Interest Table (PIT)** The routers save all incoming interests which are waiting for the requested data and also their incoming interface in a special table called

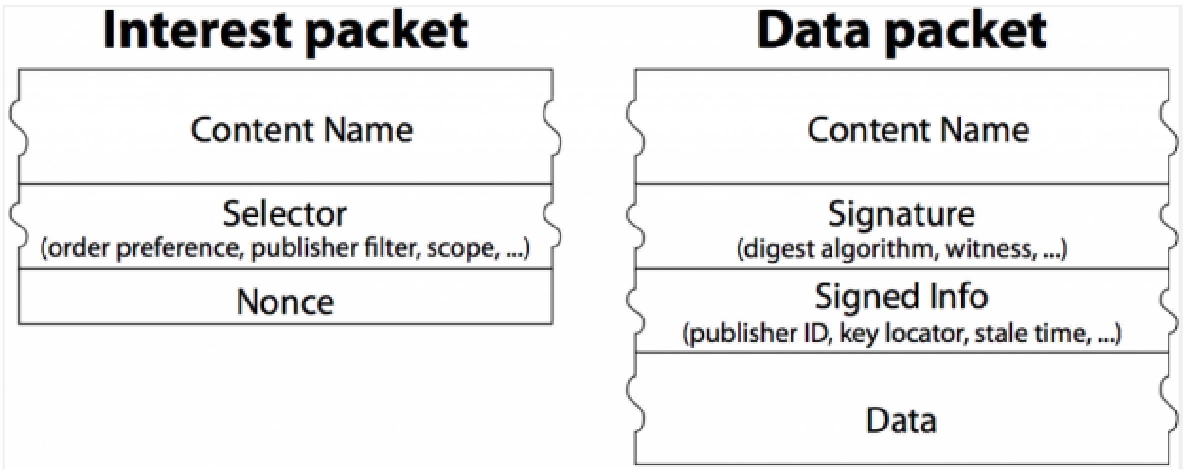

Interest packet
Content Name
Selector
(order preference, publisher filter, scope, ...)
Nonce
Data packet
Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 2.3: Two Packet Types of NDN [5].

$Pending\,Interest\,Table\,(PIT)$. The beauty of the PIT table is that only one interest is sent upstream if a router receives multiple interests for the same data. This feature leads to less interest traffic in the network, and so less bandwidth usage. Moreover, the PIT table does not remove the recently satisfied Interests immediately from its records [15]. But, it keeps them for some time. There are two reason for this. The first one is to detect loops which might be caused multiple same requests. The second purpose is because of measurements.

There are two major fields in a PIT table which are the name of the interest and the set of interfaces which belong to the consumers want to get the same data. Once the router receives the data, it finds the corresponding PIT entry and then forwards the data to all interfaces stored in the interface list. Later the router removes the satisfied PIT entry from the list.

- **Content Store (CS)** The Content Store is a cache memory for the Data packets in NDN. Every node contains the CS table in NDN. The router temporarily keeps the incoming data inside its content store table to satisfy future interests. This is called $in-networkcaching$ in NDN. A router stores the incoming data into the CS table after it finds a satisfied a PIT entry for that data. This is the way how the CS table of the router is populated. Later, when the router receives an incoming Interest it searches its CS by comparing the Interest name with the names in CS entries to satisfy the Interest before actually passing the Interest to the forwarding unit.

  In addition, because CS has a limited storage, there is a replacement policy to replace an entry with a new incoming entry. The overall performance of network can be significantly effected by CS performance [15]. Therefore, the efficient usage of the CS operations like looking up a data, inserting an entry, deleting an entry, and cache replacement policy is very important in this manner.

- **Forwarding Information Base (FIB)** $ForwardingInformationBase(FIB)$ is used to forward Interest packets to the next potential hop based on the router forwarding strategy. A FIB table consists of a name prefix and $NextHop$ records which must contain at least one record. The NextHop field contains the interface information and the route-cost of that interface in order to reach the potential data provider. The router which wants to forward an interest performs a longest prefix match operation on the FIB to find the outgoing interface. The $Longest\,Prefix\,Match$ operation is an algorithm which takes the name in a Interest packet as an input and returns a FIB entry that satisfies the longest string matching condition.

  The FIB table is updated and managed by the FIB manager in NFD module. The FIB manager controls the route information updates coming from NLSR side. it is responsible to deploy new static or dynamic routing information into the FIB table. In this way, the forwarding strategy can use the shortest path to forward an Interest packet.

The high-level diagrams of each table are also shown in Figure 2.4 with an example of a network packet. Necessary column labels are also given in that diagram for each table. In fact, the below diagram shows the NFD forwarding model together with these tables.

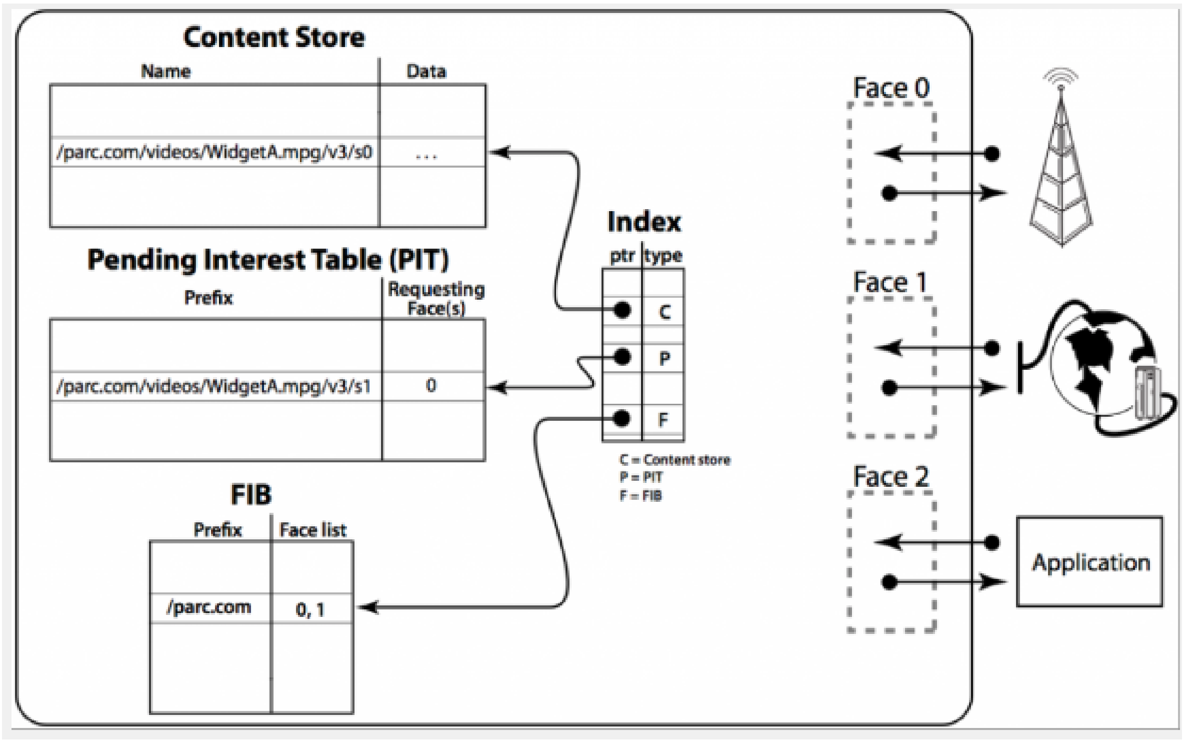


Figure 2.4: Tables and Packets in NDN [7].

In order to understand why the NDN methodology is more suitable for the current Internet needs than IP-based networks, we need to investigate the NDN architecture differences. These

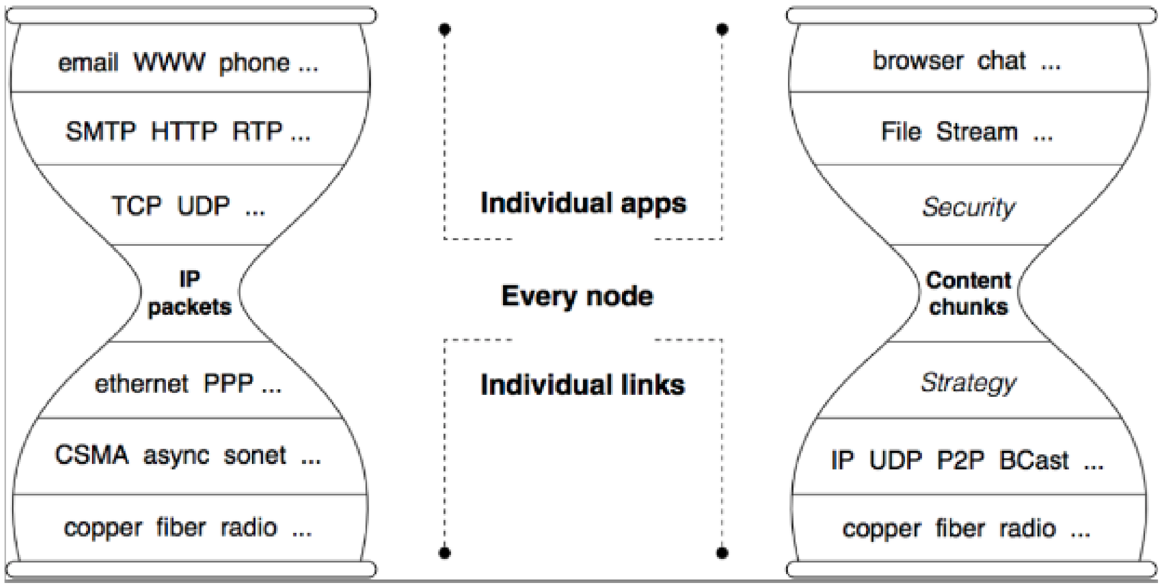


Figure 2.5: The IP and NDN Protocol Stacks [1].

differences will be explained step by step in the following part. There are six major points in this manner which are Name, Routing and Forwarding, Data Security, In-network caching, Intelligent Data Plane and Transport.

1. **Name**

   NDN uses semantically meaningful and hierarchical names to represent a data packet as shown in Figure 2.6. There is no other unique address used for identify the network packet in NDN. For example, source and destination addresses used in IP-based network packets are not used in NDN. The elements of NDN packets can also be seen in Figure 2.3. There are several advantages of naming data. First, it enables to have caching capability in routers. Because contents in NDN are requested by a unique name, so it does not matter where the content is located as long as the network find the requested content in somewhere and returns it back to the consumer. Therefore, contents can be cached in internal routers to satisfy any request in the future.

   Secondly, The NDN network packets, Interest and Data, do not carry any specific information about the router which might be private information. This contributes the secure communication and it will be more explained in future sections. Lastly, naming data allows the system to multicast data delivery. In IP-based networks, the multicast data delivery is a challenging task because more than one packets carry the same data to forward to multiple consumers due to host-centric forwarding mechanism. However, in NDN one data package is used to satisfy the multiple requester at the same time.

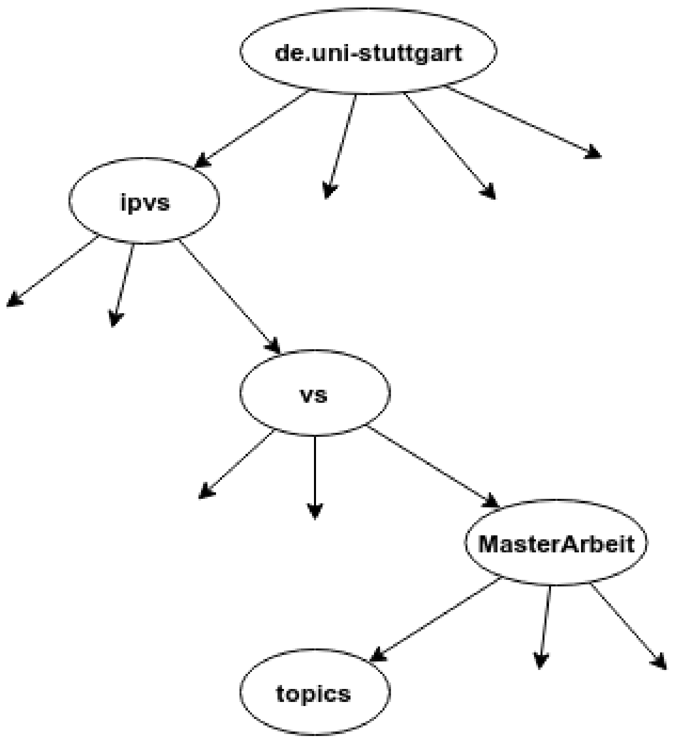


Figure 2.6: Hierarchical Name Structure

2. **Routing and Forwarding**

   Routing and Forwarding in NDN are done by using names. Routers use names to forward Interests and keeps those names in PIT to match the corresponding data that arrives at the router. This name-based forwarding has important advantages such that there is no need to handle Network Address Translation (NAT) operation or address management. Because there is no source or destination address in NDN network packets, so NDN does not require NAT conversion. Moreover, there is also no need to address assignment and management in local networks. This especially brings an advantage for the sensor networks.

   Routing security is another significant point in this section. In NDN all data and also routing messages are signed and this prevents NDN from malicious attacks. As mentioned before, network packets in NDN do not carry any information about source and destination. This supplies more privacy and also one cannot send a malicious attack to a specific target because there is no such address.

3. **Data Security**

   In NDN every data packet is digitally signed and the signature is located inside the data packet as shown in Figure 2.3. The name, the content and the metadata are signed and kept bind together securely during the transportation of the data. The security is built on the content itself rather than host or channel centric security such as IP-Network. Moreover, NDN Interest and Data packets contains only unique name and this is the only field in the network packet that routers can identify the packet. There is no requester or destination addresses in the network packet. Therefore one can not track or find the client or provider information by using the network packet or accessing a router database. In fact, routers do not contain any information about the consumer or content provider. As a result, clients and data providers are kept as private and secure.

4. **In-Network Caching**

   In-network caching is a powerful feature in NDN. Each router has a special data buffer called the content store to cache incoming data for a limited time. The aim is here to satisfy future requests. The procedure of the caching method works as follow. For each incoming interest packet, the router first queries its content store to match the requested interest name with any data name inside the content store. If there is a match, the router sends the data back to the requester immediately.

   Routers in IP-Network also buffer data for a short time. The difference between IP-routers caching and NDN routers caching is that IP routers cannot reuse those data because the forwarding mechanism is based on the host address and they can only check packet destination address and forwards the data to the destination. However, NDN identifies the data packet by the data names, therefore routers in NDN can reuse the data in the content store to satisfy a request.

5. **Intelligent Data Plane** The PIT data structure allows NDN to have a stateful data plane whereas the data plane in IP-based networks is stateless. First of all, all incoming Interest packets and their incoming interface information are stored in the PIT table. Then, these pending Interest packets are removed from the PIT table after the corresponding Data is retrieved or the timeout occurs. There are several important advantages of having stateful data plane in NDN.

   First, a router in NDN can easily collect statistical information about incoming data and timeout cases. These knowledge allows NDN to measure the performance of the different interfaces and detect packet loss information thanks to the PIT table. Secondly, together with the PIT entry information and $Nonce$ field in an Interest packet, the local router can detect Interest and Data loops. This causes loop-free data plane. Moreover, the local routers can take smart decision to forward an Interest based on the feedback and the performance information collected from the data plane. In fact, forwarding strategy can be changed according to load balancing, multiple interface services or detected problems. In this way, adaptive and error handling forwarding logic can be implemented in NDN.

6. **Transport**

   Transport layer in IP-based networks is separate and necessary layer in order to establish a stable end-to-end connection. IP and Transport layer together act like a virtual pipe between two IP-based endpoints. However, in NDN there is no such separate transport layer to handle data exchange procedure. Internet applications have to maintain the functionality of transport layer protocols.

Application side must control data delivery process by itself. For example, a consumer must re-transmit the Interest which is not satisfied in certain time for timeout or any other reasons if the consumer still wants to get that data. Another point is that the traffic load management is under routers' control. Thus, routers must handle the Interest forwarding rate and checks the overloaded hops.

Lastly, according to the NDN Project [19], NDN protocol design principles follow briefly. the first deisgn priciple is Universality. Based on this principle, "NDN should be a common network protocol for all applications and network environments". Secondly, "Data-Centricity and Data Immutability: NDN should fetch uniquely named, immutable 'data packets' requested using 'interest packets'". Third one is Securing Data Directly which explains that "security should be the property of data packets, staying the same whether the packets are in motion or at rest". Hierarchical Naming is another pronciple which says that packets should carry hierarchical names to enable demultiplexing and provide structured context". The next one is In-Network Name Discovery and according to this principle, "interests should be able use incomplete names to retrieve data packets". The last principle is Hop-by-Hop Flow Balance. Based on this, "ver each link, one interest packet should bring back no more than one data packet".

In the next section, after explaining and presenting background information about the NDN architecture and design, related studies will be investigated in this subject.

2.3 Related Work

Named Data Networking attracts more researchers nowadays to study this field [19]. Because NDN provides more efficient solutions than IP-based networks to supply Internet usage in today's smart and connected world in terms of forwarding mechanism, naming data directly and secure communication. Therefore, there are several studies on caching strategy and routing and forwarding strategy in NDN.

The first study [20] is about in-network caching usage of NDN. The article presents a cache management system to reduce content access latency, and minimize the bandwidth cost and network congestion. The purposed approach is based on distributed cache manager devices which control the items in every content store. The cache manager devices are located near routers as separate hardware modules and they communicate with each other to exchange the necessary information to control caching activities. The control mechanism uses the user demand patterns in real-time to develop a cache replacement policy. Moreover, the overall network bandwidth gain is also considered during the decision of item replacement in a cache.

This coordinated and distributed cache management system could be installed in every router having the in-network caching capability. The control managers on each router can determine the cache allocation and replacement strategy. In fact, they can exchange the configuration information and the item demand pattern among themselves. In addition, The item demand is obtained by listening to the PIT by observing the forwarded Interests.

There are some drawbacks of this purposed systems. First of all, it requires external hardware module for every router in the network to achieve its goal. This causes additional hardware cost as well as the increased maintenance and operating cost. The other point is the Interest/Data message exchange between the cache manager devices for sharing the necessary information. It will possibly affect the overall bandwidth usage and cause the traffic overhead due to flooding of Interests to search for cache managers.

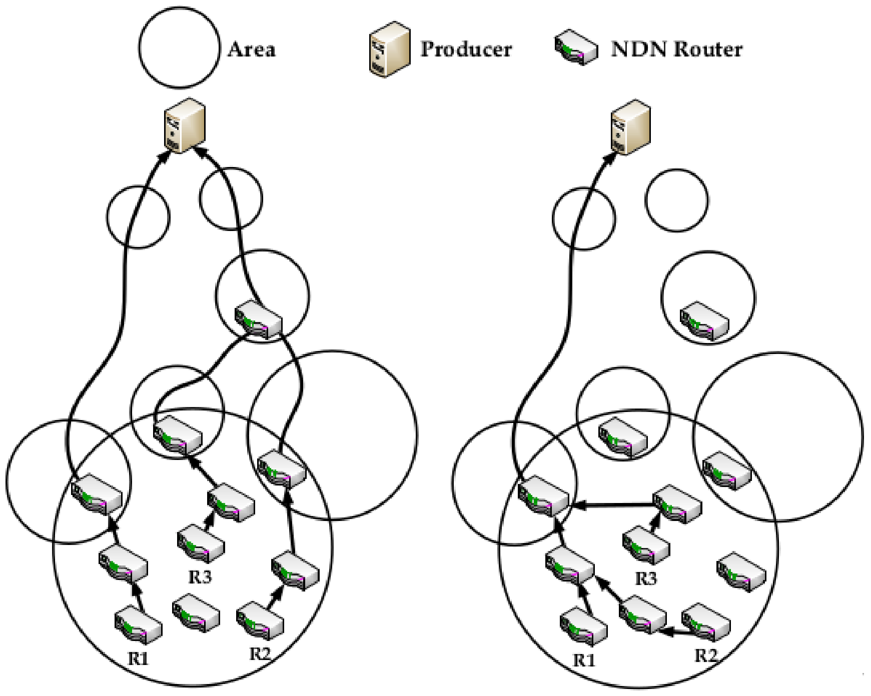


Figure 2.7: NLSR vs. MUCA Forwarding Architecture [13].

The second study [13] purposes a new routing design for NDN. This study has the same name strategy for forwarding as *MostProbablePath* (MPP) with the MPP policy in this thesis, but the design and implementation are quite different. The objective of their purposed MPP design is also to forward similar Interests over the same route to use the cache resources more efficiently. The method aims to merge the Interests coming from different consumers for the same content as early as possible in the network. By doing this, they desire to increase the cache hit ratio.

In this design, the network is divided into areas and each area has some border routers which reside in more than one areas as shown in Figure 2.7. In fact, they are like a communication bridge between two entities. The aim is to forward similar network traffic over the same border router so that resource usage will be utilized efficiently. Moreover, the additional fields in the NDN data header have been created to realize their MPP method. These are $AreaID$, $BorderRouterInfo$, and $ModifiedTime$. These fields are modified only when the requested data at the border router during the return path. Internal nodes do not change these parameters.

The parameters are changed according to the current area, border node and the arrival time of the data. As a result, the forwarding decision is made based on these fields and all similar Interests follow the same route through the same border router. The border routers update the $ModifiedTime$ field when a data name is advertised or updated by NLSR. When internal nodes receive an Interest packet having those name prefix, they use the modified time field to forward the Interest packet towards the border router. The purposed method for forwarding is practical and has significant performance results. However, the forwarding depends on partially static information. Because data names are advertised only when they have some update such as the first time data publishing and version number change. Moreover, there is need for a configuration setup for area and border router.

A diffusive name-based routing protocol (DNRP) was also proposed in [21] which computes the shortest paths to the nearest copies of name prefixes without requiring the network topology information. The shortest-path computation is based on distance information. In this method, the routing information such as the list of all active neighbor and the cost of the links are stored in a table for each router. Every known name prefix is also stored in this table with the distance to the nearest instance of that prefix. Then, each router exchanges this information between each other to compute the shortest paths for those name prefixes. Moreover, the purposed system updates the routing information when any change occurs in the network like a failure of a node/link, a cost change for a link or deletion of a name prefix. The study aims to have efficient name-based routing for the content-centric networks by using only distance information. However, DNRP does not implement in-network caching capability in its design although content store is an important element in the NDN implementation.

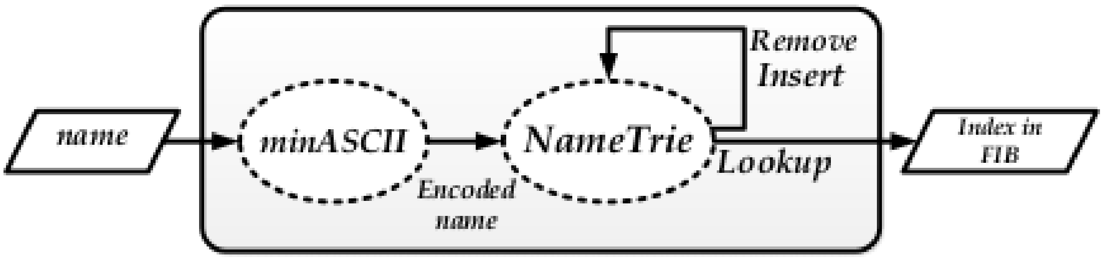


Figure 2.8: The main design of purposed NameTrie data structure [24].

There is another study which purposes a fast and memory efficient data structure [24] for storing the forwarding table entries. In this research, the objective is to improve name lookups performance and contribute to have a fast and memory efficient packet processing logic. $NameTrie$ is designed and implemented as a new data structure for forwarding table entries by the authors of this study. The design presents a new character scheme and also optimized version of the Trie structure for solving the major problems of the Trie such as traversing a trie and slow child node access. The system model of the NameTrie data structure is also shown in Figure 2.8.

The purposed NameTrie data structure presents a low memory usage to store the forwarding table entries and also decrease the time for name lookups. However, the functionality of the purposed design requires additional encoding and tree structure conversion operations. This may lead to processing overheads.

Chapter 3

Overview of The Purposed System

In this chapter, the model of the three different techniques which are to improve the efficiency and resource usage of the existing NDN implementation will be described. Moreover, it includes the assumptions which have been made during the design solution for the NDN efficiency and resource usage problem. I have collected these three forms under one name which is called The Hop Distance Aware Caching and Knowledge Sharing Based Forwarding Unit. It includes The Hop Distance Aware Caching Strategy, Segmented Data Locality Policy, and The Knowledge Sharing Based Forwarding Unit.

3.1 The Hop Distance Aware Caching

Before starting, I would like to explain how the related modules of the existing NDN implementation work. The first major module in NDN is NDN Forwarding Daemon. "NDN Forwarding Daemon (NFD) is a network forwarder that implements and evolves together with the Named Data Networking (NDN) protocol [15]". It is responsible for forwarding Interest and returning Data packets back to the consumers. To do this, it maintains basic data structures such as CS, PIT, and FIB with the implementation of the network packet processing logic unit.

The current caching policy in NFD is straightforward. For incoming data, the first step of the NFD module is to check the Interest table to see if there are PIT entries that can be satisfied by this Data packet. If there are valid PIT entries to satisfy this Data, then immediately the NFD module insert the Data into the router's CS and forwards the Data to the downstream for all the matched PIT entries. If there is no PIT entry to satisfy that Data packet, the data is sent to unsolicited data processing logic and it is dropped. NFD does not have any other advanced caching policy to decide whether it should store the incoming data or not. It just saves all incoming data which have at least one matched PIT entry.

In this method, the goal is to use the cache resources of routers efficiently and possibly to reduce power consumption caused by cache activities. As mentioned above, the current implementation of cache policy in NDN requires to process the caching logic for every incoming data. Is it really necessary to store every data in every router along the downstream path of that data? This question deserves to have more investigation. Therefore, the basic solution for this problem is to save the incoming data periodically in terms of latency between routers.

Figure 3.1 shows an example of caching saving activity for both NDN architecture and the new method presented in this thesis. The distance threshold has been taken as 100 ms. Because of the acceptable delay impact for most of the real-time application [12] is around 150 - 200 ms. This distance value can be changed for different applications. The value of 100 ms was used in this thesis during the evaluations.

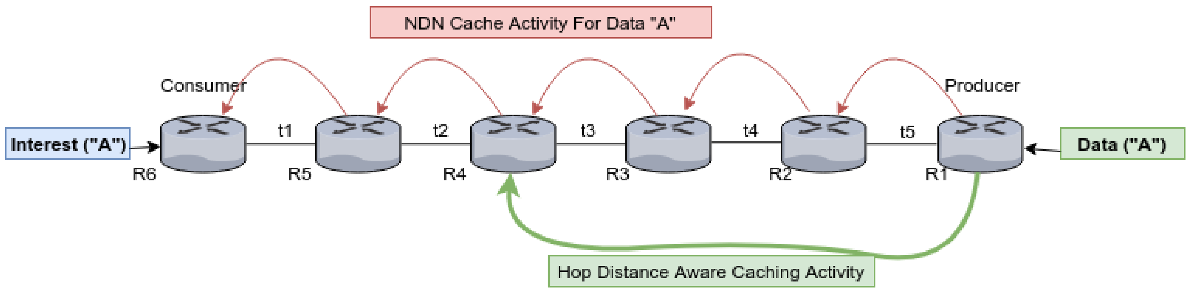

NDN Cache Activity For Data "A"
Consumer
Producer
Interest ("A")
Data ("A")
t1
t2
t3
t4
t5
R6
R5
R4
R3
R2
R1
Hop Distance Aware Caching Activity

Figure 3.1: Cache Saving Activity for NDN and HDAC. ($t1 < t2 = t5 < t4 < t3$)

To employ this new method, a new field was defined in the data header. This field is a simple travel cost counter. It starts with 0 value on a data provider point, and along with each transportation, the respective link cost is added into this counter. Each router on the reversed path checks this value to decide to save the incoming data into their CS. if the delay counter is bigger than or equal to 100 ms, then the respective router which is R4 in example figure caches the content and resets the travel cost counter to 0 value again. This is the basic functional logic for this method.

The significant point is in this method when the consumer wants to get the same data for the second time or more, the closest router having the data will be R4 and it will cost an additional $2t1 + 2t2$ ms delay. In fact, the overhead delay can be between 0 and $200ms$. This can be seen as a trade-off between resource usage and performance.

3.2 Segmented Data Aware Routers

Segmentation is sometimes required to store large data in multiple chunks. Because the content of a large data cannot fit into a network packet, so it needs to split into several pieces [11]. Segmentation is additionally useful for re-transmission of broken packets. In this way, it is convenient to re-transmit the individual segment of the data instead of sending the entire data block again. This helps to reduce bandwidth loss and delay when any failure occurs such as timeout and node failure in content retrieval.

According to the NDN Naming Conventions [10] each segment is represented by its own unique name as shown below example: In the example, the content called $video1$ consists of more than one segments and the name prefix of each segment is created by adding the segment number at the end of original content name.

$/Uni/Stuttgart/ipvs/lecture/video1/0$

$/Uni/Stuttgart/ipvs/lecture/video1/1$

$/Uni/Stuttgart/ipvs/lecture/video1/2$

$/Uni/Stuttgart/ipvs/lecture/video1/3$

.....

Segment number is located at the end of the name prefix [10]. It starts at 0 value and incremented one by one. Each segment can be requested or retrieved individually. In fact, the content consisting of multiple data segments has to be requested by sending a sequence of interests for all segments and this is under consumer responsibility. For example. if a network element wants to download a big video frame split into data segments, it must generate interests for each data segments and at the same time, it should control the sequence of the segments whether they have arrived or not.

The goal of the segmentation locality is to reduce the total delay (latency) for retrieving a segmented data. I assume that a client who has requested the first segment of a data will most probably want to get the rest of the segments of that data. Based on this assumption, I implemented a decision algorithm to send the rest of the segments which are without being requested. The functionality of the method is triggered if a segmented data requested by a consumer. The data provider for that specific segment, first, send the requested segment back to the client. Then, it runs the decision algorithm to determine the number of segments to be sent to the client.

There are three different data features playing a role in making a decision. Those are data type, priority and size. These three parameters are used as inputs to the decision algorithm. The aim of these parameters is to decrease the bandwidth loss in case the client does not want to request the rest of the data. This will be explained in the evaluation part in more details. After that, the provider router checks its own content store to search for other segments belong to the same data. If there are some segments, it collects them and sends them back to the client one by one until the decision count comes from the decision algorithm. As a result of this operation, the unsolicited data are cached during the reversed path of the first segment. When the client presents other interests for the remaining segments to the network, thanks to this new method, the requested segments will be already at the closest hop. Therefore, the client gets the data faster. Moreover, the interests for the rest of the segments will not be forwarded to the network. This will contribute to having low-interest bandwidth over the network.

To keep track of the reversed paths for the segmented data, the NFD module needs to have a similar table like PIT. But it is a small and special to the segmented interests only. The reason for having such an additional table is that after sending the requested first segment to the client, the forwarding mechanism of NDN automatically removes the corresponding PIT entry from its PIT table. Then, we lost the reverse path going back to the client or possibly more than one client. Therefore, we need another special table for the interests which want to request segments. The table records name prefix for the originally requested segment and the list of interfaces which the interest comes from. The table is populated when an incoming and segmented data arrives at the router and if the router does not have that segment in its content store. The detailed design and implementation are shown in the next chapter.

3.3 Knowledge Sharing Based Forwarding

In NDN, the routing plane is separated from the forwarding plane. the current implementation of the routing plane is called the Named-data Link State Routing protocol (NLSR). The NLSR module only calculates the shortest path from each node towards each data provider for every published data. Alternative secondary shortest paths are also calculated. Then this information is sent to the forwarding plane. The forwarding plane has all the responsibility about forwarding the interests. Moreover, the information coming from NLSR is updated continuously if there is any change in the network topology.

There are three major tables in the NDN forwarding plane as shown in Figure 3.2. These are Content Store table, Pending Interest Table (PIT), and Forwarding Information Base (FIB). CS is temporary storage for incoming data. PIT is used for keep tracking of unsatisfied interests with their downstream interfaces. Finally, FIB table has the list of published data names and with their path cost towards to data provider.

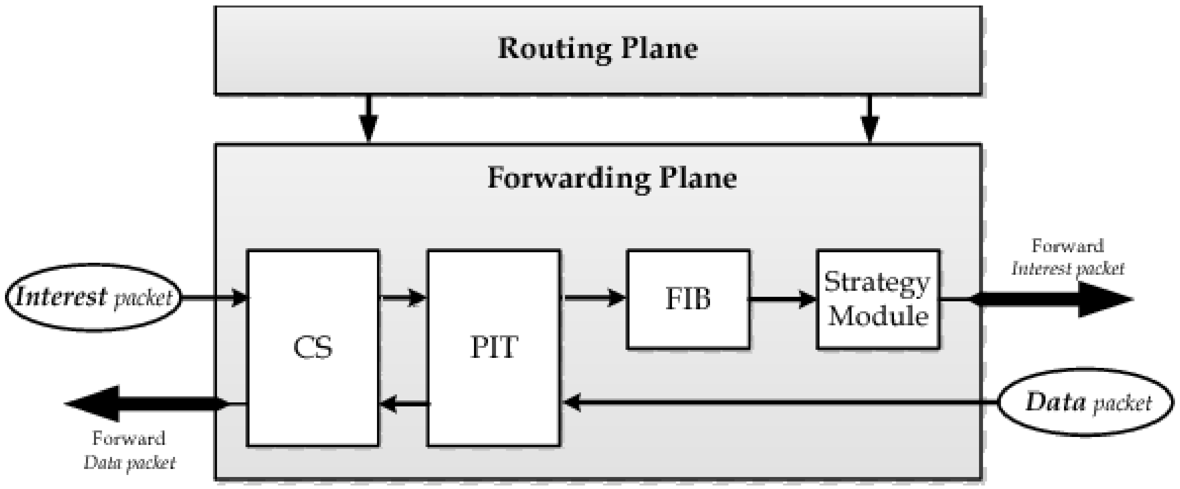


Figure 3.2: NDN Forwarding Plane [13]

The current processing logic of the implementation of the forwarding unit is shown in Figure 2.2. A consumer initiates the process by sending an Interest packet to the network. Then, a router receives the Interest packet and it, first of all, searches its CS table for the requested interest name. If the searched content exists in the CS, then the content of the desired name is returned back to the client. If it does not, then the router looks for a PIT entry to match with the interest. If PIT has an entry having the same name with the interest, this means that the same content has been requested from some other consumers and the corresponding interest had been already forwarded to the upstream and currently the router is waiting for the desired content from the network. Thus, there is no need to create a new entry in PIT. Instead, the existing PIT entry is modified by adding the incoming interface into the list so that once the requested data arrives, the router can forward the data to all downstream hops which are on the PIT entry list. But, if the interest does not match a PIT entry, then a new entry is created for the interest in PIT and the interest is forwarded to the next hop.

The Figure 3.3 summarizes the steps up to the forwarding stage. The forwarding is done by forwarding strategy and the forwarding strategy uses FIB to decide the next hop. The current default forwarding strategy in NDN is the Best Route Strategy [15] which is based on the lowest routing cost to forward an Interest to the upstream.

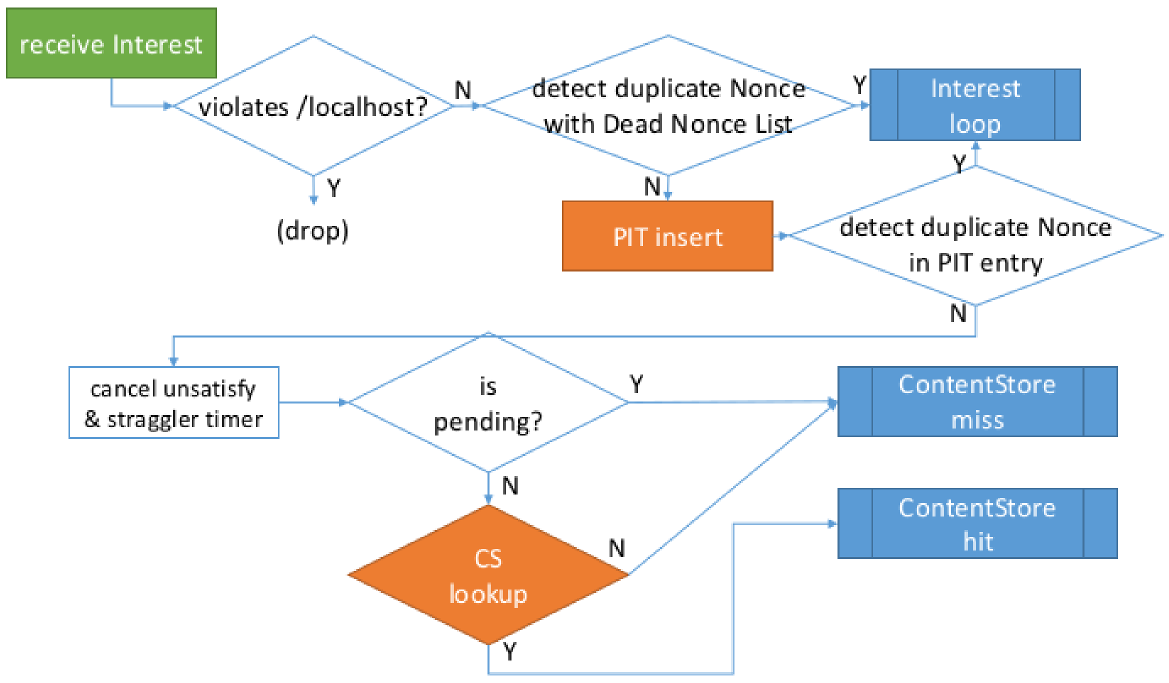


Figure 3.3: NFD Incoming Interest Processing [15]

The Best Route strategy starts with the lowest-cost next hop to forward a new interest. If the transmission fails and the consumer sends the same interest again, the forwarding strategy chooses the second-lowest-cost next hop to forward the interest. In fact, the strategy tries to use all hops (except the downstream hop) one by one if re-transmission occurs. The order is determined by the lowest-cost path. If the router uses all hops, then it starts with the first one again.

Although the Best Route strategy guarantees the lowest-cost to the data provider, the outcome does not mean that it is optimal when we consider the overall topology. For example, we can look at the scenario in Figures 3.5 and 3.6 to understand the disadvantages of the BRS. The known parameters and information in this example are given below.

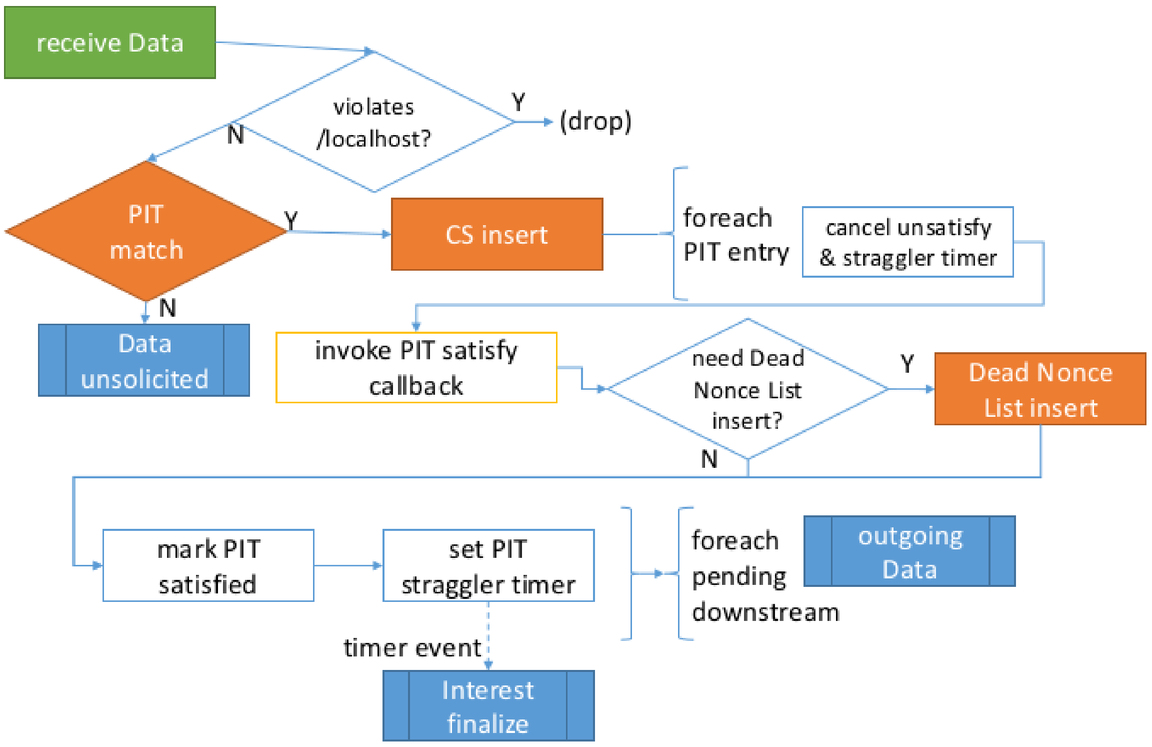


Figure 3.4: NFD Incoming Data Processing [15]

- The costs of each link are represented by $t1$ and $t2$.
- The value of $t2$ is higher than $t1$ ($t1 << t2$).
- The router R8 is the publisher for the data "A" and the router R1 and R2 want to receive the data "A".
- The lowest-cost paths for the router R1 and R2 are shown the reversed and thick links respectively on the figure.
- R1 presents the interest for the data "A" and then R2 send the same interest for the same data.

The router R1 starts the transmission by sending a interest packet for the data "A" to the network. The interest packet follows the lowest-cost path ($R1- > R3- > R5- > R7- > R8$) to reach to the data publisher router R8. The total cost for this path is $3t1 + t2$. After the interest has reached the router R8, the requested content is found in CS of R8. Then R8 sends the data back to the consumer router and the data follows the reversed path to reach the requester node R1. The content "A" is cached by each router on the reverse path.

When the node R2 presents the same interest for the data "A", the forwarding strategy of NDN chooses the lowest-cost route for this data. In this case, it follows $R2- > R4- > R6- >$

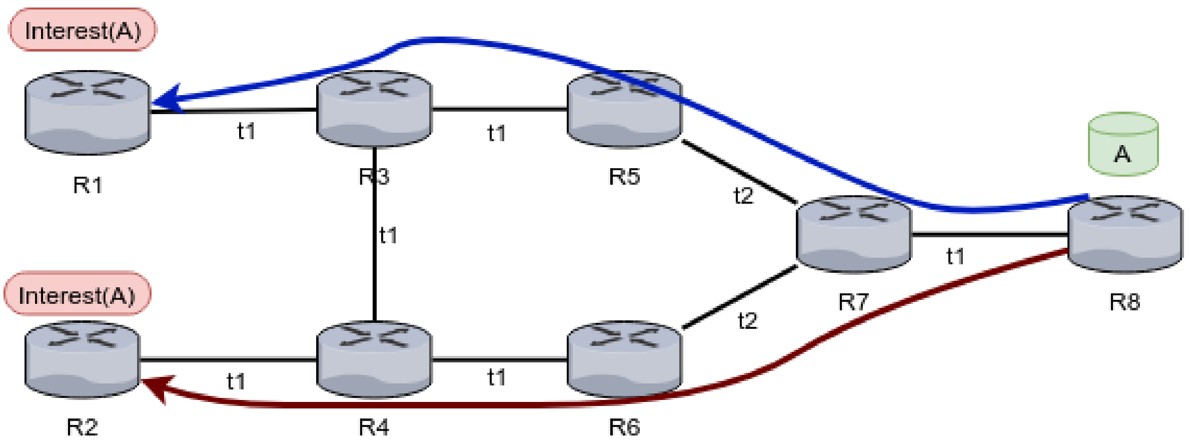


Figure 3.5: NDN Forwarding Example: Router R1 request data "A". The upper link L1 (from R8 to R1) is the lowest-cost path for the router R1 and the bottom link L2 (from R8 to R2) is the lowest-cost route for the router R2.

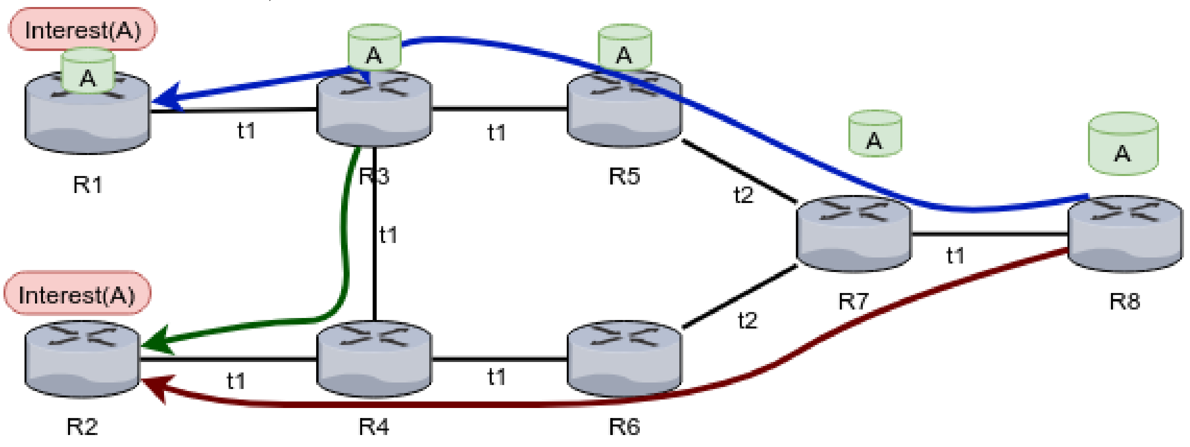


Figure 3.6: NDN Forwarding Example: Router R2 request data "A".

$R7- > R8$. The total cost for this route is also the same as $3t1 + t2$. The desired content will follow the reverse path and will reach the router R2. The only difference is here, the content will be already in the node R7 and it will be returned by this router immediately. Therefore, the total cost will be $t1$ time unit less than the total cost.

The problem in this scenario is that although the content "A" is located in the node R3 which is near than R8 as shown in Figure 3.6, the current forwarding strategy (BRS) is not smart enough to know this information. Therefore, the interest is forwarded towards the router R8. The new method is constructed based on this fact. The main goal of this new forwarding mechanism is to reduce data latency and bandwidth usage of the overall network.

The knowledge sharing based forwarding unit consists of two parts. The first one is to share the content name with all neighbors (except downstream neighbor which is the one who has sent data). This information is sent to the entire network. Each router stores the content name and the incoming interface. in a special table which is called Most Probable Path table (MPP table). Then if a router receives the same interest for that specific data, it simply checks its MPP table and forwards the interest to the corresponding interface instead of using the lowest-cost route. This operation is done for all incoming content.

The below shows some important use cases.

- A new entry is created for an incoming content name if the same content name does not exist inside the MPP table.
- The same content name might come to a router more than one. The new forwarding mechanism has a comparison function which uses two parameters specific to the shared content name. These are the total cost and the total hop count. Every shared content also carries these two parameters and if a router has already the same content name in its MPP table, it uses these two parameters to decide to update its record with the new shared information.
- During forwarding an Interest to the upstream, the first routing choice should be MPP.
- If MPP does not have enough knowledge to forward an Interest, the second routing policy which is probability path should be used.
- If the probability value is not enough to decide forwarding, then the default route, best-path, should be selected.
- In case of getting NACK message after an interest packet has been forwarded by using MPP table, the forwarding switch the default forwarding strategy which is the BSR. After the next successful transmission of the same content, the forwarding strategy goes back to the knowledge sharing based forwarding strategy again.

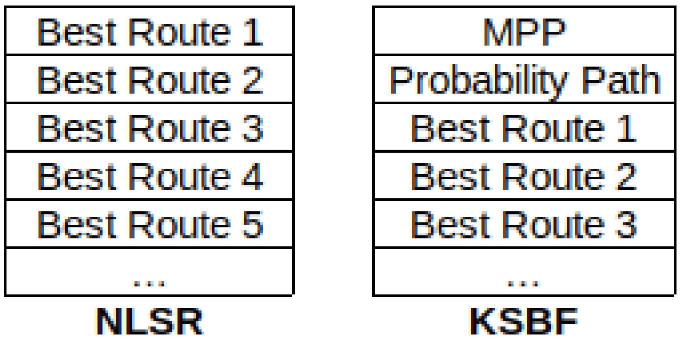


Figure 3.7: NLSR vs. KSBF Route Lists

The second part of the knowledge shared based forwarding is to calculate a probability value based on the records in the MPP table. The assumption is that for an incoming interest we can extract a relation value by using the existing content names in the MPP table. This value can be used to forward the incoming interest to the related interface. The probability value is calculated by using the relation between the incoming interest name and the existing content names in the table. The probability value must be bigger than the threshold value which is 0.6 so that the related interface can be used for forwarding.

As a result, the knowledge sharing based forwarding module presents two new forwarding routes on the top the default forwarding strategy (BSR). The new list of forwarding paths is shown in Figure 3.7

Chapter 4

Design and Implementation

In this chapter, the design, and implementation of three methods are described. The section also presents some high-level code snippets when it is necessary for each method.

4.1 CS Hop Distance Aware Caching Approach

This section describes the design of the Hop Distance Aware Caching method and how it is implemented and integrated into existing NDN platform. There are two modules which need to be modified in order to achieve the HDAC approach. The first one is NDN-CXX module. NDN-CXX is a C++ library which implements NDN primitives. It is used for developing different NDN applications and projects such as NFD and NLSR. The HDAC needs to have an additional field in data header to keep track of the reverse path cost. The related code files for this operation are in ndn-cxx library. First of all, a $unsigned\,integer$ value is defined in $Tags$ header file of the ndn-cxx library. Moreover, this value must be individually decoded and encoded before sending and receiving the data packet respectively. This is done in NFD Generic Link Service source file. The information about related function and the algorithm are shown in Algorithm 1 for both decoding and encoding of a field for data header.

This new field allows data to carry its own individual path cost in itself. The cost starts with 0 value and incremented by adding the cost of each link in which is used during data delivery on the reverse path. In this way, a router can check the value of this field and understand the total cost of the data until that point. If the traveling cost is bigger than or equal to $100ms$, then it reset it to $0ms$ and saves the content into its own CS. If it is not, the hop does not cache the content and simply forwards the data to the downstream.

The second part is to use this new data header field to implement the functionality of the HDAC method. The logic is straightforward. The travel cost field should be controlled by comparing with the threshold value for every incoming data. After that, the decision is made for caching the content or not. The related algorithm is given in Algorithm 2.

The process of the HDAC does not require intensive processing as can be seen as the above code piece. Because it consists of a simple value reading and writing into the data field as well as an if-condition statement. This small patch is used to determine whether an incoming

Algorithm 1 Encoding and Decoding Of A New Data Field

```
1:  Input: NetworkPacket, LinkServicePacket
2:  /*Encoding*/
3:  $travelCost \leftarrow NetworkPacket.getField(DATAPATHCOSTFIELD)$
4:  if $travelCost$ is not NULL then
5:      $travelCost \leftarrow Encode(travelCost)$
6:      $InsertNetworkPacket(travelCost)$
7:  else
8:      $travelCost \leftarrow 0$
9:      $InsertNetworkPacket(travelCost)$
10: end if
11: /*Decoding*/
12: $data \leftarrow newData()$
13: $travelCost \leftarrow LinkServicePacket.getField(DATAPATHCOSTFIELD)$
14: if $travelCost$ is not NULL then
15:     $data.addTag(travelCost)$
16: end if
17: return
```

Algorithm 2 Caching Decision Procedure

```
1:  Input: inFace, data
2:  $linkCost \leftarrow inFace.getCost()$
3:  $dataSavingPolicyFlag \leftarrow FALSE$
4:  $travelCost \leftarrow data.getTag(DATAPATHCOSTFIELD)$
5:  if $data - travel - cost$ is not NULL then
6:      $travelCost \leftarrow travelCost + linkCost$
7:      if $travelCost > 100$ then
8:          $dataSavingPolicyFlag \leftarrow TRUE$
9:          $travelCost \leftarrow 0$
10:     end if
11:     $data.setTag(travelCost)$
12: end if
13: .....
14: if $dataSavingPolicyFlag = TRUE$ then
15:     $ContentStore.insert(data)$
16: end if
17: return
```

data should be saved into content store or not. The outcomes and analysis of this new caching policy will be presented in the next chapter.

4.2 A New Policy For Segmented Data

The design and implementation of the Segmented Data Aware Routers method are presented in this section. The design of the method can be divided into three steps. The first part is the detection of segmented data when data in a router is discovered in the content store for an incoming interest. This step triggers the initial process of the SDAR functionality. A new function, *processHitDataSegment*, is called at the end of the primitive function of NFD module which is $Forwarder :: onContentStoreHit$. The original desired content is sent back to requester first. Then, the function *processHitDataSegment* is executed to start to understand whether the requested data is a segment of a data block or not. if the determined amount of the segments are found in the content store, they are shifted to the downstream pipeline. The Algorithm 3 shows the related pseudocode for this operation. The three new fields have been added into the data header. They are a data type, data priority, and data size. They are used during the decision operation for the number of segments to be sent to the downstream. This also requires ndn-cxx library modification as mentioned in the previous section.

Algorithm 3 Processing Data Segments After Sending the First Segment

```
 1: Input: outFace, data
 2: if $isSegmented(data) = FALSE$ then
 3:     return
 4: end if
 5: $namePrefix \leftarrow data.getName()$
 6: $segmentNo \leftarrow data.getSegmentNo()$
 7: $N \leftarrow decideNuberOfSegment(data)$
 8: $segmentList \leftarrow getSegmentsFromCS(data, N, segmentNo)$
 9: while $i < segmentList.size()$ do
10:     $s \leftarrow segmentList[i]$
11:     $sendData(s)$
12:     $i \leftarrow i + 1$
13: end while
14: return
```

The first step starts the process by sending the unsolicited segments to the downstream. The second step follows. The data comes to the incoming data pipeline. However, the data has not been requested and there is no PIT entry for it. Hence, the incoming data pipeline process is terminated by sending the data to the method which is called $Forwarder :: onDataUnsolicited$. In this method before we exit the process, I injected another algorithm to check whether the name of the incoming data is segmented or not. Thus, we can understand it is our data segment.

This is done by a new function which is called *processIncomingDataSegments*. It is defined in NFD forwarder module and executed inside the primitive function of NFD which is *Forwarder* :: *onDataUnsolicited*. The important steps in procedure is shown in Algorithm 4. The incoming data pipeline sends the data which do not have any corresponding PIT entry (that means 'unsolicited') to the *Forwarder* :: *onDataUnsolicited* method and here the process for the incoming data ends. Therefore, the segments that we have sent to the downstream fall into this method. Because they have not been requested, so there is no PIT entry for them.

As a result, the function collects these data and checks whether they are segmented or not. If it is segmented, then we can understand that it is our segment. Now the forwarding information is needed to send the segment to the downstream. That is, we need the list of downstream hops which have previously requested the first segment of the related data block. The third step is used for this purpose. As mentioned earlier, a simple table is created to keep track of the incoming segmented data name and interface information. The table structure is defined as below code snippet.

```
/*structure for the segmented data*/
struct Segmented_Interest_Table {
  Name;                        //Name of the incoming interest
  FaceId;                      //Interface Identification
  Original_Segment_No;         //The requested segment number
  Nonce;                       //Nonce of the Interest
  SegmentList;                 //Segment List for downstream
  Entry_Creation_Time;           //Used for deleting the Timeout };
```

Algorithm 4 Processing Incoming Unsolicited Data Segments

```
1: Input: data
2: if isSegmented(data) = FALSE then
3:   return
4: end if
5: $namePrefix \leftarrow data.getName()$
6: $faceList \leftarrow getFaceList(SegmentedInterestTable, namePrefix)$
7: if $faceList$ is not NULL then
8:   $i \leftarrow 0$
9:   while $i < faceList.size()$ do
10:    $outFace \leftarrow faceList[i]$
11:    $outFace.sendData(data)$
12:    $i \leftarrow i + 1$
13:  end while
14: end if
15: return
```

The third step also requires an additional method to populate the table. This method must be called from the original function of forwarder in NFD which is *onContentStoreMiss*. The necessary data information is stored at the table if the data is segmented and it is used at the second step to forward the incoming unsolicited data segments to the downstream.

4.3 Dynamic Forwarding Unit: Most Probable Path

In this section, the design and implementation details of the MPP method are presented. First of all, among the all three new features which have been purposed in this thesis, the MPP method is the most complex and challenging feature. Because MPP is located at the top of the forwarding list and it must handle all variants of the forwarding unit such as NACK and timeouts. It should have the capability of switching between the list of forwarding paths. Hence, the design and implementation will follow one by one to be more understandable for this feature.

The first action is to populate the MPP table and share this knowledge with the entire network. The first design requirement is a table to store the incoming data information such as data name and interface. It is called the MPP table. Some significant elements of the table and the structure is shown below.

Listing 4.1: MPP Information Table

```
/*Statistic Table for MPP strategy*/
struct MPP_Structure_Table {
  Name;             // The Name Prefix of Data
  FaceId;           // Interface Address
  HopCount;         // Hop Distance from the Data Provider
  TravelCost;       // Travel Cost
  Probability;      // Probability Value
  NackCounter       // Failure case flag for Nack/Timeout
  .....
};
```

The following steps include MPP processing logic:

1. A new method is called at the end of the NFD primitive function which is *Forwarder* :: *onIncomingData* for populating the MPP table. The incoming data and its incoming interface information is passed as an input to the function called *addMPPStatisticTable* The travel cost and hop distance fields of the incoming data are set to 0 because the content is cached in the current router content store. Then, the gathered information is saved into the MPP table. The Algorithm 5 presents the related code part which insert an entry into the MPP table.

2. After inserting the incoming data information into the MPP table, a virtual data is created with a special name which is 'ndn:/mpp/ProudToShare'. Then, the information composed of data name, incoming interface, travel cost, and hop distance is stored in

that data as a content. After that, the virtual data is sent to all neighbors of the router, except the upstream hop. The source code of this step is shown in below Algorithm 6.

3. Then, that special shared data comes to the incoming data pipeline of a router. Before processing the content as a normal incoming data, each router checks the name of the incoming data with the keyword mentioned the previous step so that it can understand it is a regular content or it is a MPP shared information.

4. If the keyword is found in the incoming data name, the data is separated from the regular incoming data processing pipeline. Then, the content is parsed and the incoming link cost is added into travel cost. Then, the shared info is sent to the function to insert into the MPP table. If the MPP table is updated with this information, the router also shares the same information in the same procedure with its neighbors. In this way, the information is spread to the entire network. The code part for this procedure is shown in Algorithm 7.

5. The next major step is to use this shared information to forward the incoming interests. There are three list of routes available to forward the incoming interests.

6. The first path is chosen if the incoming interest name matches with any name inside the MPP table. That means, one of the neighbors has already that content. Therefore, the interest should be sent to that neighbor node instead of using the NLSR Best Route Strategy. The scenario in Figure 3.6 is a very good example for this step. In that example, the interest to obtain the content 'A' coming from the router R2 follows the red marked paths to reach data provider router R8 if the BRS is used. However, the content 'A' is already cached in the router R3. Thanks to the MPP table, now the router R4 will know this information and it will simply forward the interest to the hop R3. Because, the router R3has the content, the desired content will returned back to the consumer in a shorter time than the normal shortest-path.

7. If the required name is not found in the MPP table, the second route strategy is tried. It is is to compute a probability value based on the entries in the MPP table. Based on this value, the interface of the relative entry is used for forwarding an Interest. The probability is calculated based on the Equation 4.1. In this equation, every name sharing the same interface information is grouped together. Then, an average value is calculated according to the function φ which compute the similarity value for given two name prefixes. Lastly, the maximum average value among all interfaces is used.

8. The incoming interest enters into the $Forwarder :: onContentStoreMiss$ function if it is not found in the router content store. Then, as shown in Algorithm 8, it is forwarded based on the ranked list of the routes which are MPP, Probability Path, and Best Route Strategy.

9. This new forwarding method covers the NACK and Timeout use cases. If one of them occurs, the forwarding control is given under the Best Route Strategy which is the third route to forward an interest to the upstream.

$$p = max(\,(\sum_{c=0}^{k_i} N_i[c] * \varphi)\,/\,k_i\,) \tag{4.1}$$

where:
p - probability value,
i - list of neighbors ,
φ - word matching counter function,
k - size of the name list for each neighbors,
N - list of names for each neighbors.

In this equation. the word matching counter function counts the number of matching components of two name prefixes. An example computation of this function is shown in Figure 4.1. The function only counts up to first mismatched component as can seen in the example. Name 1 and Name 2 have the same components up to 4th component and the function gives result accordingly. This operation is done for each name prefix sharing the same neighbor.

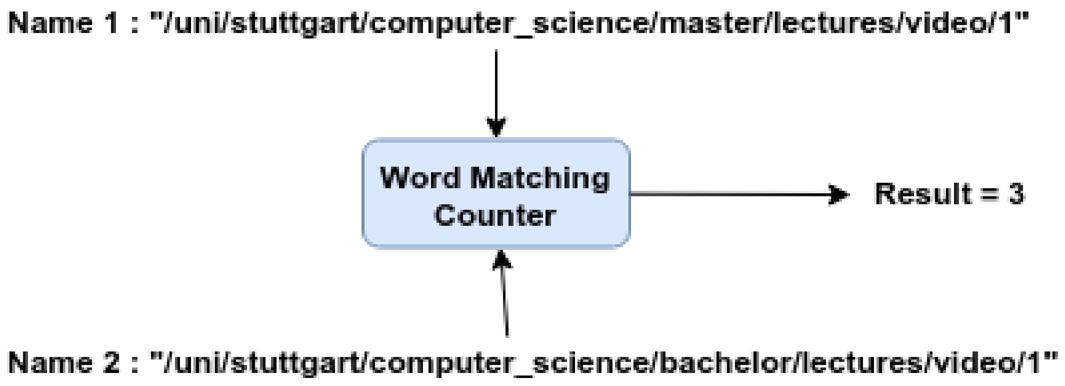

Name 1 : "/uni/stuttgart/computer_science/master/lectures/video/1"
Word Matching Counter
Result = 3
Name 2 : "/uni/stuttgart/computer_science/bachelor/lectures/video/1"

Figure 4.1: φ Function Example

In Algorithm 5, there are five input parameters which are incoming interface identification, content name, probability value of the content, travel cost, and hop count. First, the incoming data name is searched for each entry in the MPP table whether there exists an entry having same content or not. If there is already a same name in the table, travel cost and hop count is multiplied and assigned into parameter k. If the incoming data cost and hop count multiplication is less than the computed value k, then it is decided that the incoming name is less costly and its interface information is replaced by the existing one. If there is entry having the same name, then the function creates a new entry with the incoming parameters and insert it into the MPP table.

Algorithm 6 receives two inputs which are an interface identification and a content for sharing. First of all, an empty data object is created and filled with the required values. Then, for each neighbor except the upstream neighbor, the content is sent to the their interface. In this way, the required knowledge is sent to all neighbors.

Algorithm 5 MPP Table Insertion Operation

```
1: **Input:** faceId, dataName, probability, travelCost, hopCount
2: $i \leftarrow 0$
3: **while** $i < MPPStructureTable.size()$ **do**
4:   **if** $dataName = MPPStructureTable[i].name$ **then**
5:     $k \leftarrow MPPStructureTable[i].cost * MPPStructureTable[i].hopCount$
6:     **if** $travelCost * hopCount < k$ **then**
7:       $MPPStructureTable[i].name \leftarrow dataName$
8:       $MPPStructureTable[i].faceId \leftarrow faceId$
9:       $MPPStructureTable[i].probability \leftarrow probability$
10:      $MPPStructureTable[i].cost \leftarrow travelCost$
11:      $MPPStructureTable[i].hopCount \leftarrow hopCount$
12:      $shareInfo \leftarrow dataName + travelCost + hopCount$
13:      $sendMPPTableToNeighbors(faceId, shareInfo)$
14:    **end if**
15:    **return**
16:  **end if**
17:  $i \leftarrow i + 1$
18: **end while**
19: /*create a new node*/
20: $entry \leftarrow newMPPStructureTableentry$
21: $entry.name \leftarrow dataName$
22: $entry.faceId \leftarrow faceId$
23: $entry.probability \leftarrow probability$
24: $entry.cost \leftarrow travelCost$
25: $entry.hopCount \leftarrow hopCount$
26: $entryinsert(entry)$
27: $shareInfo \leftarrow dataName + travelCost + hopCount$
28: $sendMPPTableToNeighbors(faceId, shareInfo)$
29: **return**
```

Algorithm 6 Sending Statistical Information to all Neighbors

```
**Input:** faceId, sharedInfo
/*creating a virtual data*/
$data \leftarrow newData()$
$data.setName("ndn:/mpp/ProudToShare")$
$data.setFreshnessPeriod(10)$
$data.setContent(sharedInfo)$
$sign(data)$
$i \leftarrow 0$
**while** $i < NeighborsList.size()$ **do**
    **if** $NeighborsList[i].faceId = faceId$ **then**
        /*do not send to downstream*/
        continue
    **end if**
    $outFace \leftarrow getFace(NeighborsList[i].faceId)$
    **if** $outFace$ is not NULL **then**
        $outFace.sendData(data)$
    **end if**
**end while**
**return**
```

Algorithm 7 The incoming Shared Data Processing

```
**Input:** faceId, data
$namePrefix \leftarrow data.getName()$
**if** $namePrefix \neq "ndn:/mpp/ProudToShare"$ **then**
    **return**
**end if**
$linkCost \leftarrow getNeighborLinkCost(faceId)$
$routerName \leftarrow getRouterName(faceId)$
$content \leftarrow parseData(data)$
$name \leftarrow content.getName()$
$cost \leftarrow content.getCost()$
$hop \leftarrow content.getHop()$
/*Increment hop distance by 1*/
$hop \leftarrow hop + 1$
/*add incoming link cost*/
$cost \leftarrow cost + linkCost$
/*try to add the suggested name in to the MPP table*/
$addMPPStatisticTable(name, faceId, 0, cost, hop)$
**return**
```

The shared incoming data comes into the incoming data pipeline. At the beginning of this process, Algorithm 7 is called. It first checks the name of the content whether it contains the keyword for the sharing data. If the content name does not include the keyword, then the function simply exit by returning. If it is a shared data, then the content is parsed and sent for adding it into the MPP table. The link cost and hop count are updated accordingly.

In algorithm 8, an incoming Interest is forwarded to the next hop by trying the list of routes which are MPP, probability path and Best Route in order. First, MMP route is used by calling the function $findFaceIdInMMPTable$ (cf. line 3 in Algorithm 8). This function is responsible to search MPP table for a given Interest. If a valid entry is found, Interest is forwarded according to MPP route. If it is not, then probability path is triggered by calling *calculateProbability*. If the computed probability is enough for deciding forwarding, then probability path is used to forward Interest. If it is not the case, the last route which is Best Route is used to forward Interest.

Algorithm 8 Forwarding an Incoming Interest Procedure

```
 1: Input: incomingFaceId, interest, pitEntry

 2: /*First Route: MPP*/
 3: $faceId \leftarrow findFaceIdInMMPTable(interest)$
 4: if $faceId$ is not NULL then
 5:   $nextHop \leftarrow faceTable.get(faceId)$
 6:   if $netHop$ is not NULL then
 7:     /*There is a record in the MPP table*/
 8:     $onOutgoingInterest(nextHop.interest, pitEntry)$
 9:     return
10:   end if
11: end if

12: /*Second Route: Probability*/
13: $faceId \leftarrow calculateProbability(interest)$
14: if $faceId$ is not NULL then
15:   $nextHop \leftarrow faceTable.get(faceId)$
16:   if $netHop$ is not NULL then
17:     /*The probability value is valid*/
18:     $onOutgoingInterest(nextHop.interest, pitEntry)$
19:     return
20:   end if
21: end if

22: /*Third Route: Best-Path*/
23: $dispatchToBestPathStrategy(incomingFaceId, interest, pitEntry)$
24: return
```

Chapter 5

Evaluation and Results

In this chapter, firstly, the environmental setup is described. The setup section gives necessary information such as the test platform and network tools. The network topologies were used in this thesis are presented in the setup section as well. Next, the performance of the purposed Hop Distance Aware Routers and Knowledge Sharing Based Forwarding is evaluated through simulation test results.

The performance analysis of the purposed system has been done with the network emulation tool which is called Mini-NDN. Mini-NDN is a lightweight and scalable network emulation tool [17]. It is used for testing, experimenting and researching on the NDN platform. Mini-NDN combines the NDN libraries which are NFD, NLSR, and the tools released by the NDN project to provide a single system for simulating an NDN network which is also shown in Figure 5.1. It is an open and free software tool under the GPL 3.0 license. The links between the routers are presented as Virtual Ethernet pairs. In addition, resource allocation such as CPU and Memory usage is configurable for each node. The latest implementation of Mini-NDN has been used in this thesis to test the purposed system. The experiments has been tested on a PC with Intel Core i5 3210M CPU 2.50GHz, 6GB DRAM. The CPU composed of 4 real and 4 virtual cores.

Besides the simulation program, there are several essential tools for NDN to analyze the network traffic and to test the performance of the system. Some of the important tools are listed below.

- **ndn-traffic-generator :** NDN Traffic Generator tool produces Interest and Data traffic in an NDN network. Two configuration files are used to specify the pattern of the NDN traffic which is desired to be generated. One of them is run on the server side and the other is run on the client side. In this way, the data exchange procedure has been started according to the configuration files.

- **ndndump :** It is a traffic analysis tool which captures the NDN network packets on the link. The overview information about Interest and Data packets such as packet length and time-stamp are displayed by the tool. The traffic results can be saved into a file. Later, the bandwidth usage can be computed by using the output file.

- **ndnping :** ndnping is used with ndnpingserver to test the reachability between two hops. Moreover, it computes the latency (RTT) for the Interest and Data exchange.

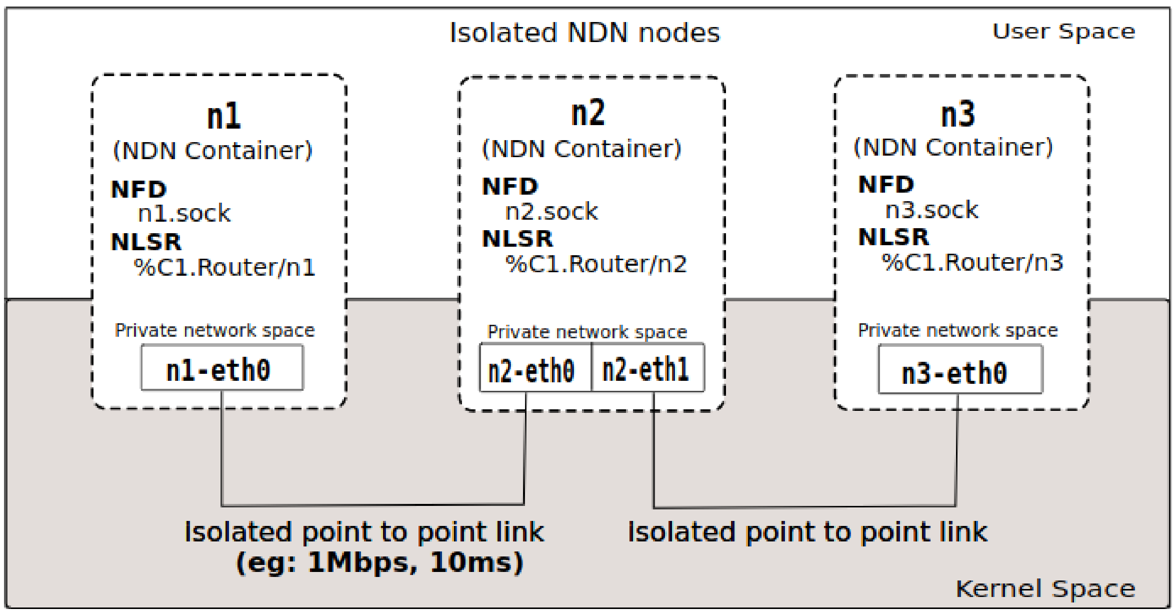


Figure 5.1: Mini-NDN Components and Relationship [17]

- **ndnputchunks and ndncatchunks :** They are used together to transfer a file as Data Segments. The first one, ndnputchunks, works as a data producer program. The second one, ndncatchunks, works as a consumer program which retrieves the segmented data and displays it on the standard output.
- **Mini-NDN Edit :** It is a graphical user interface software tool to create a custom network topology file for NDN. One can configure the host and links separately by using this tool.

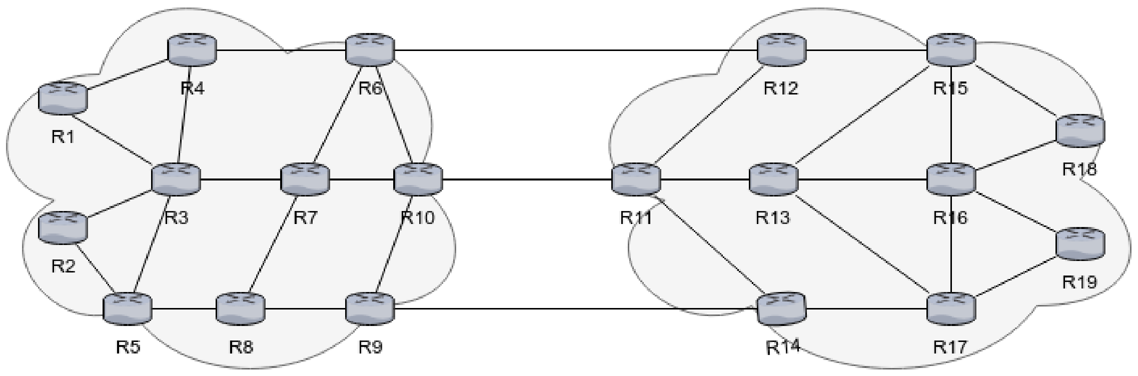


Figure 5.2: Topology 1: NDN Custom Topology

The last point in this section is network topology. Two different topologies were used in this study. The first one is a custom creation topology. It is created by using Mini-NDN Edit software tool which is mentioned earlier paragraphs. It is specially designed to show the positive effects of the purposed system. There are 19 router nodes in this topology. Figure 5.2 presents this topology.

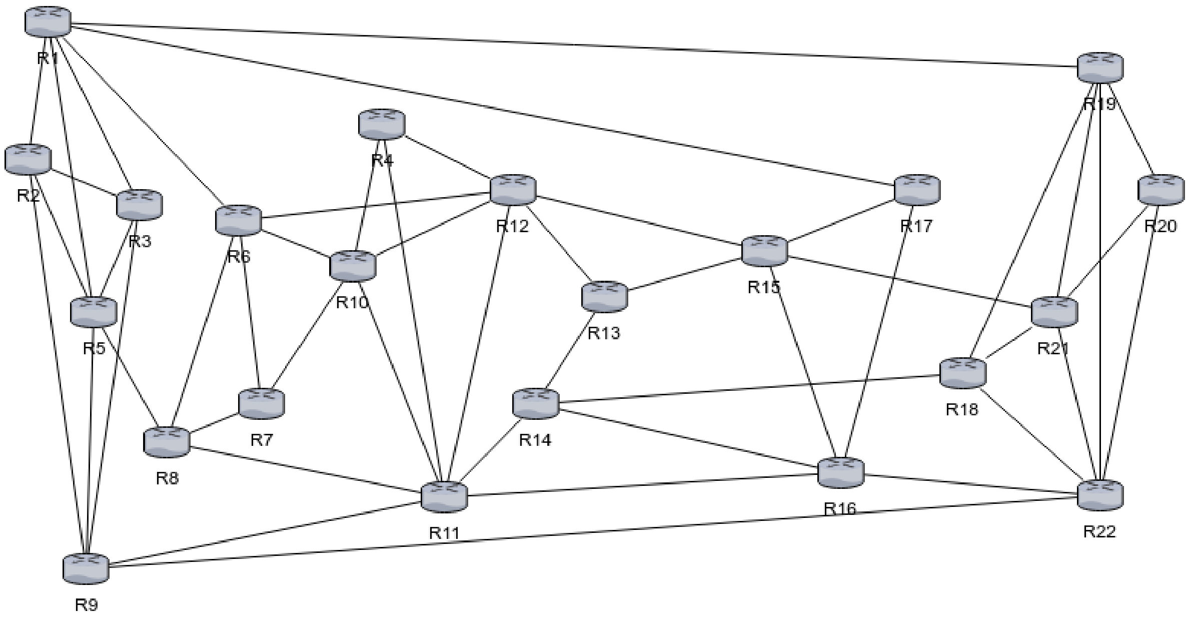


Figure 5.3: Topology 2: NDN Testbed Topology

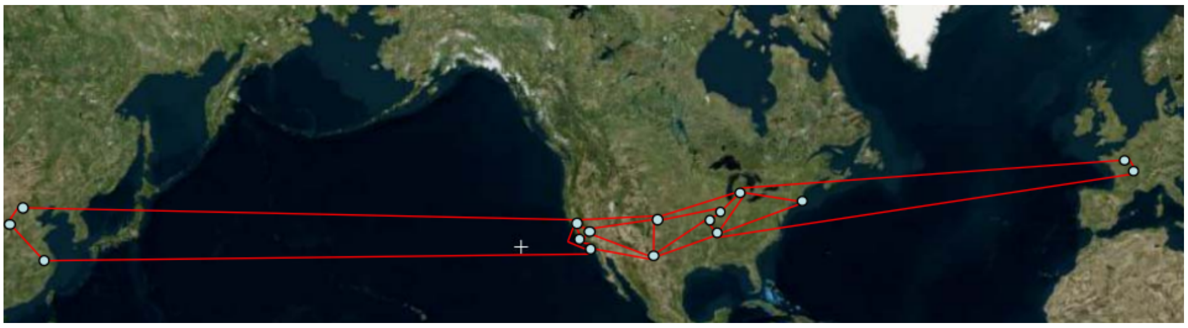
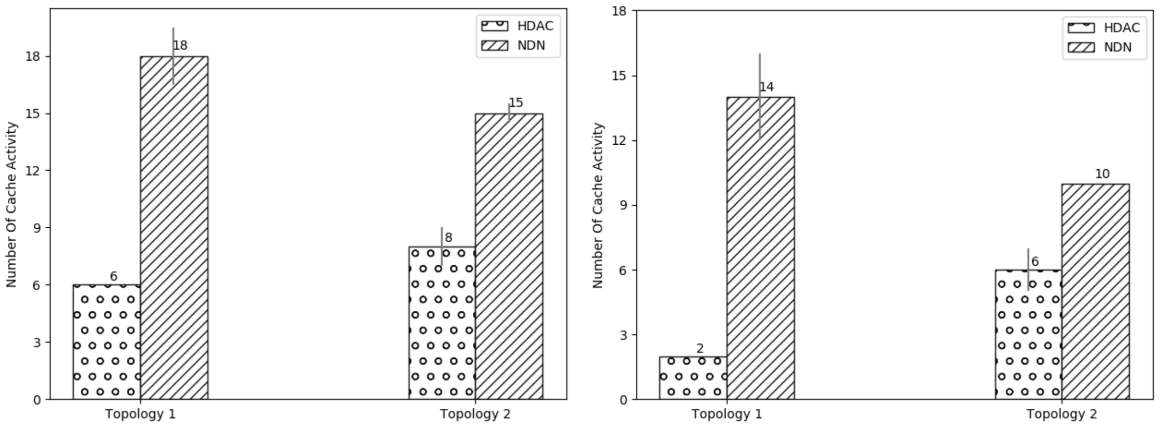


Figure 5.4: NDN Testbed Topology on World Map [18]

The second one is the current NDN Testbed Topology and the second one is a topology based on the real locations on Earth. NDN Testbed topology has been built on the real locations on Earth. It includes 22 router nodes and the nodes are presented on 3 continents. Figure 5.3 shows the NDN Testbed topology.

5.1 HDAR Policy

In this section, the performance of the Hop Distance Aware Router policy is presented. The results have been compared with the current NDN caching policy. Two different scenarios were used to measure the performance of the purposed caching strategy. The first one does not include the MPP support whereas the second one uses the MPP help to have better performance results. The same test was repeated for both topology 1 and topology 2 for both scenarios and the results are the average of two runs. The variance between each run is represented on the bar chart. In some results the variance is 0 because some consumer shares the same shortest path. In addition, As mentioned earlier sections, data is cached in every router on the reverse path. Based on this knowledge, the metric which is used in this test is the number of caching activity.



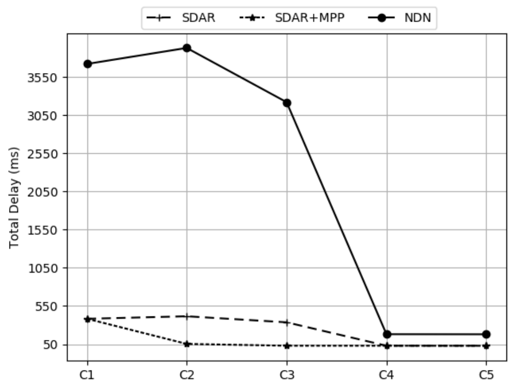
(a) Without MPP Support　　(b) With MPP Support

Figure 5.5: HDAR vs. NDN: Number Of Cache Activity

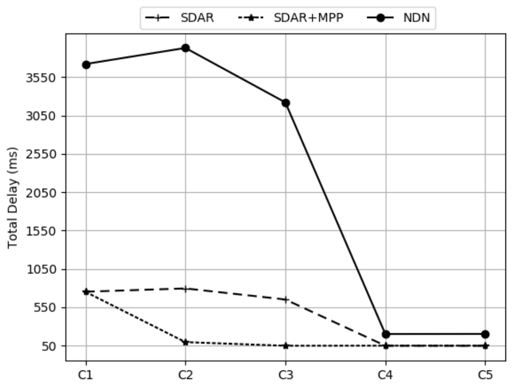
In the first scenario, three consumers request three different data from the same data provider. After three consumers received the requested data, the number of cache saving process is counted for both the current NDN caching policy and the purposed caching strategy. The results of the first scenario are given in Figure 5.5a. According to these results, the number of caching activities significantly has dropped thanks to the HDAC policy. The less number of caching activity causes to have less processing. Moreover, the decrease in the processing unit might lead to consuming less energy.

In the second scenario, three different consumers request the same data from a data provider. The aim of this test is to trigger the MPP process and to see the effects of MPP on the HDAC policy. MPP is used when a data is requested the second time. According to Figure 5.5b, the advantage of MPP can be seen. the caching activity is additionally decreased by the help of the MPP method.

The negative aspect of the HDAC idea is data latency. That is when a user wants to get the same data again (multiple times), the RTT for this data will be between 0 and 200 ms. Because the nearest point which the data cached last time might be 100 ms far from the requester. This aspect of the HDAC method should be considered during design NDN based communication application.



(a) When data priority is $High$



(b) When data priority is Low

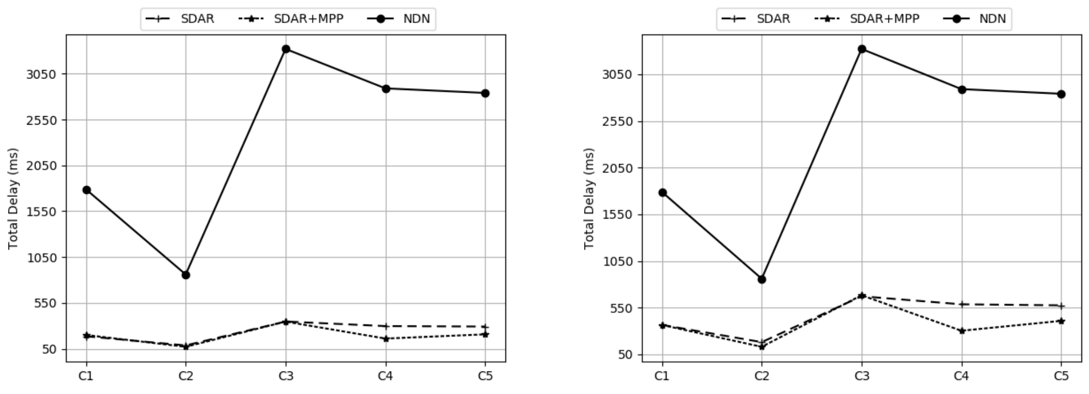
Figure 5.6: SDAR vs. NDN : Total Delay Time For Content Retrieval, Topology 1. Data priority affects the number of segment to be sent back to the requester

5.2 Segment Locality Performance Results

The performance analysis of the purposed SDAR method is presented in this section. Total data retrieving delay is used to measure the performance of this method. The measurement has been done for three different designs which are NDN, SDAR, and SDAR with MPP Support. Accordingly, two different test cases were used to analyze the performance of the developed system.

In the first test case, there are five different users and they request the same data in order. The data is composed of 10 identically sized segments and its priority is $High$. This priority is important for the decision algorithm which decides the amount segment to be sent back to the requester right after the first segment delivery. For example, the segments of the data having the $High$ priority are sent at once. Moreover, along with the RTT measurement, the bandwidth usage was also computed during this experiment. The first test scenario is applied for topology 1 and topology 2.

The results of the first test scenarios are given in Figure 5.6a and 5.7a respectively. the vertical bar shows the total delay for Interest and Data exchange and the horizontal bar represents each consumer. When we look at the first topology in Figure 5.6a, we can see that the total delay is significantly low against NDN results for some consumers such as C1 and C2. The reason for this huge difference is that right after sending the first segment, the remaining segments are also sent back to the consumer, and when the consumer wishes to receive the other segments, the desired segments will be already located near the consumer. Thus, the delay will be decreased accordingly for those segments. On the other hand, in NDN implementation Interests have to go to the data provider for each data segment. Moreover, MPP additionally helps to reduce the delay for some consumers by giving better forwarding decision as shown in mentioned figures.



(a) When data priority is *High* (b) When data priority is *Low*

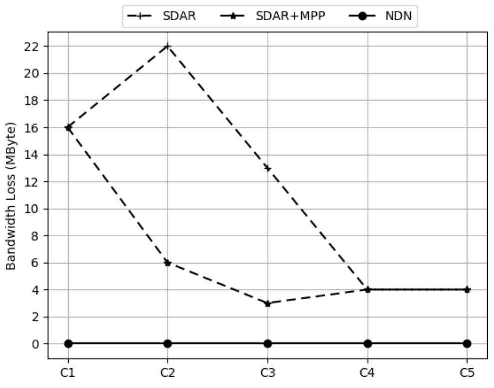
Figure 5.7: SDAR vs. NDN: Total Delay Time For Content Retrieval: Topology 2. Data priority affects the number of segment to be sent back to the requester

In Figure 5.6b, the purposed system takes more time to get the data. However, NDN results remain the same in this case. The reason in this chart is because that data priority is low and the decision algorithm sends fewer segments after the first segment was sent. Therefore, to retrieve the entire data consumers have to send more than one interests. This causes additional delay, but it may help to reduce the drawbacks of this method which will be mentioned more in future paragraphs.

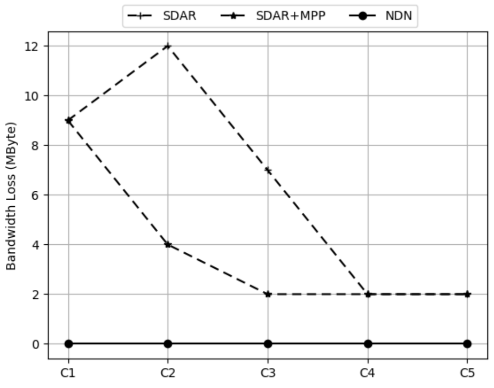
The second part of the first test case is to run the same test by using topology 2. The results are given in Figure 5.7a and 5.7b. The data latency difference of the purposed system is also clear in this network topology. The amount of time to retrieve the entire data is much less than the current NDN design. Moreover, the ascending trend between C2 and C3 is because they have different forwarding paths having different costs.

The SDAR method may help to reduce the data retrieving delay. However, there is a charge for this delay decrease. To test the drawback of the systems, bandwidth usage was used as a performance indicator. This is the second test case in this section. Again, the same five clients request the first segment of the data. But this time, they will not request the rest of the segments. They need the first segment only. As a result of this scenario, NDN will not send the second interest to the network and there will be no Interest and Data exchange after the first segment. Hence, it will not occupy extra bandwidth on the network. However, without the user request the proposed system will send the sequence of segments back to the user right after sending the first desired segment. This will consume additional bandwidth although the unsolicited segments will not be used. To decrease this overhead, data features (type, priority, and size) are given to the decision algorithm as inputs and the decision algorithm takes the responsibility to determine the number of segments to be sent to the user.

The results are expressed as bandwidth loss in MByte unit. The bandwidth loss can be seen clearly in Figures 5.8a and 5.8b for the first topology. For example, the bandwidth loss is



(a) When data priority is *High*



(b) When data priority is *Low*

Figure 5.8: SDAR vs. NDN: Bandwidth Loss when the consumer does not want to get the rest of data segments: Topology 1

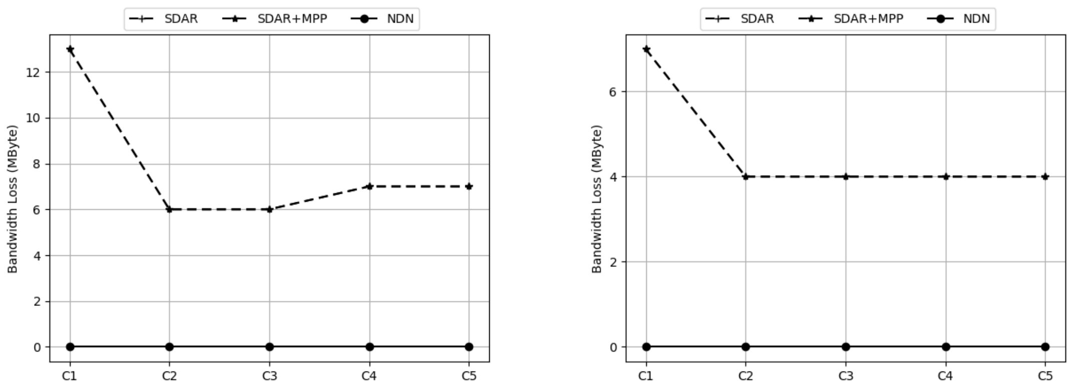
around $22MByte$ when the data priority is high whereas it is about $12MByte$ when the priority is low in topology 1 for the client C2. The amount of the loss is directly related to the size of segments and the features of the data. The tradeoff between data latency and bandwidth loss can be adjusted by changing the decision algorithm. The results for the second topology are shown in Figures 5.9a and 5.9a.

5.3 Performance of MPP Model

In this section, the evaluation of the Knowledge Sharing Based Forwarding method is presented. Firstly, three different performance metrics were used for both MPP and NDN to measure the performance. These metrics are the average RTT (Round Trip Time) for Interest/Data exchange, utilization for the router and the links between routers, and the overall bandwidth usage. The reason for having more than one metric in this section is that a new routing and forwarding strategy affects the multiple aspects of the network such as content retrieval delay and resource usage. Therefore, it is better to measure the performance of the purposed forwarding in different ways so that the distinction between MPP and NDN can be seen clearly.

There is one test case which was used to compute the performance in this part. Five randomly chosen distinct clients send Interest one by one to get the same data from a data provider. Then, for each client, the total content retrieval is calculated. After clients received the data, the router and the link utilization and the overall bandwidth usage are computed. The measurements are repeated two times and the results are the average of these two runs.

The first outputs are shown in Figure 5.10a and 5.10b for topology 1 and topology 2 respectively. The graphs show the average content retrieval time for the five clients mentioned above.



(a) When data priority is *High* (b) When data priority is *Low*

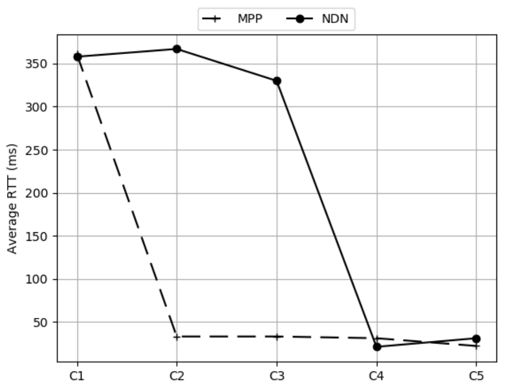
Figure 5.9: SDAR vs. NDN: Bandwidth Loss when the consumer does not want to get the rest of data segments: Topology 2

As can be seen on both topologies, some clients such as C2/C3 in topology 1 and C4/C5 in topology 2 experience the benefits of the MPP method by having a low latency than NDN. However, some clients have similar data delay with NDN.

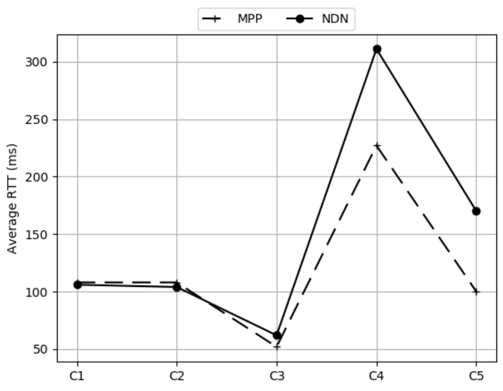
The utilization of the routers and links is another important performance indicator to understand the benefits of the MPP strategy. Figures 5.11a and 5.11b present the utilization results for both topologies. The results are quite promising in topology 1 because of the resource usage for both routers and links. In this case, the utilization difference is about 30% in the router usage and the link usage. In the second topology, this difference is not as big as in the first one, but MPP still has an advantage over NDN.

The last performance metric, overall network bandwidth usage, is shown in Figure 5.12. The result is given as percentage value for both topology 1 and topology 2. To collect the bandwidth usage, NDN traffic analysis tool which is *ndndump* has been used. The outputs of the test show that the MPP design significantly reduces the bandwidth over the entire network according to results in Figure 5.12. The decrease is 30% in the first topology and 19% in the second topology. These bandwidth savings might be really valuable for specific networks like real-time communication.

Although the virtually created data is used for sharing the statistical information between the router, the cost for this operation is not too much in terms of bandwidth usage. The bandwidth need for sharing the statistical information, first of all, depends on the size topology and the size of the shared name prefix. For the topologies having less number of routers, the MPP design will not consume too much bandwidth to share the information between routers. However, it may cause an important amount of bandwidth usage for the bigger topologies. In our cases, for both topology 1 and 2, the bandwidth consumption is about 20 KByte in average during the entire test.

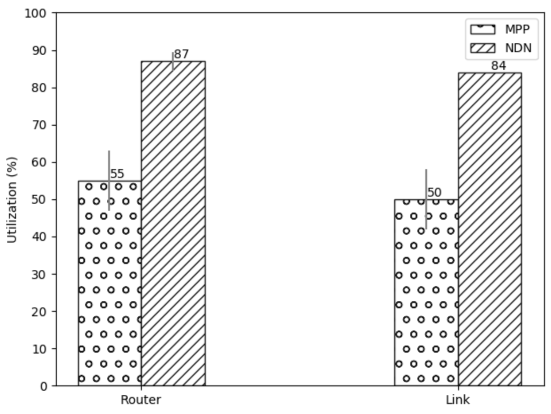


(a) Topology 1

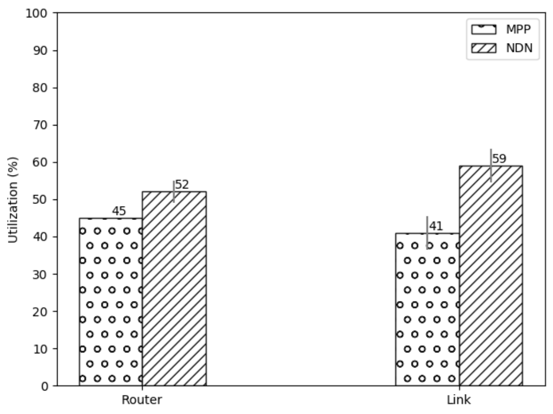


(b) Topology 2

Figure 5.10: MPP vs. NDN: Average Round Trip Time For Content Retrieval



(a) Topology 1



(b) Topology 2

Figure 5.11: MPP vs. NDN: Router and Link Utilization Percentage

The additional test has been done in this section to cover the failure case. The MPP design must be capable of handling exceptional cases. For example, it should handle the failure of a router on the network when a NACK or Timeout case occurs. The purposed KSBF has already the necessary functionality to take care of the NACK and Timeout use cases. It switches to the Best Route Strategy when it receives a NACK/Timeout message. After successful communication for data, MPP becomes available again. Figure 5.13 shows the failure scenario results. The first topology was used to perform the failure test case.

In this scenario, the router R1 has already received multiple distinct contents from the data provider router R18. Then, the other user R2 wants to get the same data over time. The MPP table of the router R2 already contains the necessary information about those contents. Therefore, When R2 presents Interests to receive the same contents, MPP will forward Interests towards R3 and then R4 in the case of topology 1. Initially, R2 receives the content from

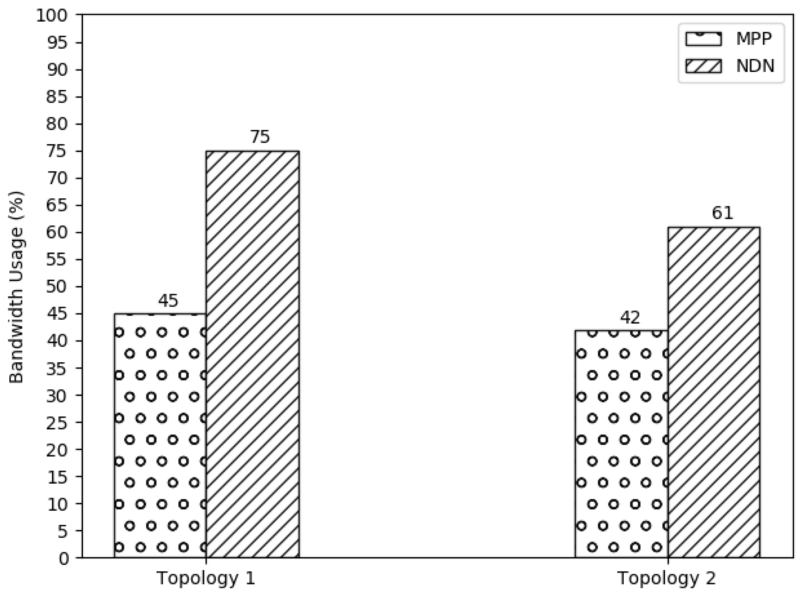


Figure 5.12: MPP vs. NDN: Bandwidth Usage Percentage

the nearest hop which is R4 thanks to the MPP forwarding unit. However, the router R4 is killed during the time frame between 10 and 20 seconds.

As can be seen in Figure 5.13, during this time, MPP fails to get the content and it gives the forwarding decision to the Best Route Strategy. This takes more time to retrieve the contents because Interests have to go to the original data provider by using the router R3's best path. Once we activate the node R4 again in 20 seconds, MPP starts to forward the interests toward R4 with less delay. As a result, MPP handles the NACK and Timeout cases dynamically.

5.4 Results Discussion

In this section, the overall performance of the purposed method will be summarized. First of all, the distance-based caching strategy performance results are showed in Table 5.1 which presents the percentage decrease in caching activity between HDAC and NDN. The colum label gives whether MPP support is provided or not. The rows represent topologies and the last row is the average of two topologies. As can be seen on the table, 56% (without MPP support) and 62% (with MPP support) less caching activity occurs when the purposed caching policy implemented into the current NDN design. That is, This caching operation decrease in overall network reduces processing overhead as well as power consumption needed for memory operations. The advantage of this caching policy especially can be seen on a bigger topologies. There might be many close routers in terms of link-cost on a big topology. For example, there are 10 routers having 1 ms link-cost between each of them during the return of a large content. In current NDN design, NFD module will cache the content in all those routers one by one

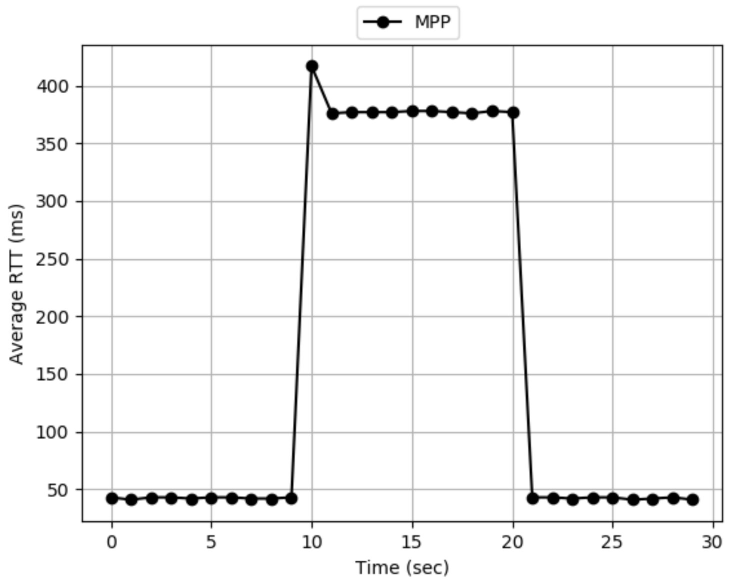


Figure 5.13: MPP Node Failure Test Case, Topology 1

although the content may not be requested from most of the routers. However, the purposed caching policy only caches the content in one of the routers and thanks to MPP if someone request the same content, the related Interest will be forwarded to near node to get the content in efficient way.

Table 5.1: Distance-based Caching Performance Overview, Percentage of Reduced Caching Activity

	Without MPP	With MPP
Topology 1	66	85
Topology 2	46	40
Average	56	62

Secondly, the SDAR performance is briefly listed in Table 5.2. The values showed on below table are maximum and minumum delay difference (in seconds) between SDAR and NDN. In other words, SDAR method may reduce data retrival delay by maximum 3.5 seconds and minimum 0.15 seconds for the first topology. On average, maximum delay reduction might be 1.82 seconds and 1.87 seconds. Number of segment and size of each segment directly influence the amount of delay decreased. These time savings can be significant for large content such as video streaming and FTP applications.

Lastly, MPP performance is highlighted in Tables 5.3 and 5.4. First table shows reduced resource utilization values for both router and link usage. When we look at the average values, MPP may result in 19% less router and 26% link usage on average. This causes less

Table 5.2: Segmented Data Policy Maximum and Minimum Delay Reduction in Seconds

	Maximum	Minumum
Topology 1	3.50	2.97
Topology 2	0.15	0.77
Average	1.82	1.87

Table 5.3: MPP Resource Utilization Decrease

Util. Reduction (%)	Router	Link
Topology 1	32	34
Topology 2	7	18
Average	19	26

processing on the router processing logic and gives them more time to handle other tasks. This decrease in resource usage might be useful especially for individual routers. Because traffic load managing is done by each router individually by controlling the Interest forwarding frequency. If a router is overloaded with incoming Interests, then it reduces the speed of forwarding Interest rate. In this manner, the purposed MPP method directly helps to reduce this overload on a router and increases the overall network quality.

Table 5.4: MPP Overall Saved Bandwidth Consumption Percentage

	Topology 1	Topology 2
Bandwidth Usage (%)	30	20

Second table presents overall bandwidth consumption which is one of the most important quality-of-service requirements. According to given results, MPP dynamic forwarding mechanism consumes around 30% less bandwidth for Topology 1 and 20% for Topology 2. This bandwidth reduction might be very significant especially for content intensive network applications such as Netflix and YouTube. Netflix is one of the most popular video streaming applications and it consumes 15% of the total bandwidth volume of traffic globally [25]. In fact, similar contents are watched on Netflix from multiple user again and again. The dynamic forwarding mechanism of MPP might be very significant for these kind of application to reduce the overall bandwidth consumption.

Chapter 6

Conclusion

In this chapter, the important points of the purposed system are summarized. Subsequently, the future work section describes what can be done in the future to reduce the overheads of the purposed system and also to extend the testing capability.

6.1 Conclusion

Routing-Forwarding and in-network caching are key factors in the NDN implementation. The performances of these two modules are very important to realize an efficient NDN design. In the current implementation of NDN, the routing and forwarding unit can be improved by providing different method on the top of the default routing logic. Moreover, caching strategy can be changed to use the resources efficiently.

In this thesis, I have designed and implemented the Hop Distance Aware Caching and the Knowledge Sharing Based Forwarding method to increase the efficiency of the current NDN implementation in terms of resource usage, bandwidth consumption and data retrieving latency. There are three major contributions of this thesis which are a $distance - aware\ caching$, $segmented\, data\, policy$, and $dynamic\, forwarding$. The important features of these three contributions are highlighted below paragraphs.

The first method which have been implemented is a new caching policy. In this method, the data transferred to the user on the reverse path is cached only in specific routers which are far enough from the previous router in terms of data latency. The objective of this method is to reduce the caching activity, processing overhead and possibly the energy consumption caused by those caching operations. According to evaluation part of the thesis, the caching activity is reduced up to 40% compared to the current NDN caching policy. In fact, this value can be more improved by MPP support. The caching activity might be around 60% less than the caching strategy in the current NDN implementation.

A segmented data policy is the second method which aims to decrease the data retrieval time for only segmented data. A large data can be stored as number of segments in NDN and all segments are identified with their unique name prefixes. In order to get the entire segments, the consumer application must send number of Interest which equals to the number of segments. However, the new policy developed in this thesis sends remaining segments back to the user without having been asked for after processing the first requested segment. As mentioned in evaluation part, he results shows significant drop in data retrieving delay when the segmented policy applied while it might cause bandwidth overhead if the user does not want to get the rest of the segments.

The last method is a dynamic forwarding unit. The current routing policy in NDN depends on partly static shortest-path information. It does not consider the dynamic changes on the network topology. For example, a content might be obtained by a neighbor router and thanks to in-network caching capability in NDN the neighbor router saves that content for you. But still if that neighbor router is not on your shortest path, your Interest packet for the same content has to go along the shortest-path to get the same data for you. However, the purposed dynamic forwarding method eliminates this overhead by sharing the knowledge with its neighbors about the content which it has processed already. This policy is called *Most Probable Path* running on the top of the route list. Thus a router first use the MPP method to forward an Interest packet before trying to use the Best Route strategy. The performance results proof that the MPP policy uses 25% low bandwidth in overall and 20% low resource usage in average of two given topologies. It also reduces the total round trip time for some consumers who want to get the similar content with its neighbors.

In conclusion, these three methods, $distance-aware\,caching$, *segment locality*, and $dynamic\ forwarding$ will play an important role to have better efficiency in NDN in terms of resource usage, bandwidth consumption, and data retrieval latency.

6.2 Future Work

In this section, the possible future works for the mentioned methods will be described. First of all, to demonstrate the capability and the limits of the purposed system, the tests could be run on different topologies having huge amount of nodes. This may reflect better understanding to see the effects of the purposed methods. Secondly, the decision algorithm which is used in the segment locality method can be extended to deal with the overheads of the policy. Lastly, the MPP method has to use NLSR as a default routing policy and the further studies can be made to change this dependency. This might save NDN from intensive routing information calculation.

Bibliography

[1] V. Jacobson, D. K. Smetters and N. H. Briggs, *Networking Named Content.* Palo Alto Research Center CA USA, 2009

[2] V. Lehman, M. Hoque, Y. Yu, L. Wang, B. Zhang, L. Zhang, *A Secure Link State Routing Protocol for NDN.* ACM SIGCOMM Workshop on Information-Centric Networking, 2013

[3] E. Bourtsoulatze and J. Saltarin, *Content-Aware Delivery of Scalable Video in Network Coding Enabled Named Data Networks.* IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 20, NO. 6, JUNE 2018

[4] J. Eduardo and S. Arco, *NETWORK CODING ENABLED NAMED DATA NETWORKING ARCHITECTURES.* Universität Bern, Bern, 15 September 2017

[5] L. Zhang, *NDN Architecture Overview.* ACM ICN 2016, University of California, Los Angeles, September 26, 2016

[6] NIST, *Information Centric Networking Program.* [online] Available at: https://www.nist.gov/programs-projects/information-centric-networking-program [Accessed 22 Mar. 2019].

[7] Named Data Networking, *Named Data Networking: Motivation & Details* [online] Available at: https://named-data.net/project/archoverview/ [Accessed 22 Mar. 2019].

[8] C. Bian, T. Zhao, X. Li, and W. Yan, *Boosting named data networking for data dissemination in urban VANET scenarios.* School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China, 21 February 2015.

[9] Virtual Reality, *EVERY DAY BIG DATA STATISTICS* [online] Available at: http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/ [Accessed 26 Mar. 2019].

[10] NDN Project Team, *NDN Technical Memo: Naming Conventions.* NDN, Technical Report NDN-0022, July 21, 2014.

[11] I. Moiseenko, *Fetching content in Named Data Networking with embedded manifests.* NDN, Technical Report NDN-0025, September 25, 2014.

[12] PubNub, *Human Perception and Technology* [online] Available at: https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology/ [Accessed 8 April 2019].

[13] C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang *MUCA: New Routing for Named Data Networking.* ISBN 978-3-903176-08-9 c 2018 IFIP, 2018.

[14] A. Abane and P. Muhlethaler *Towards evaluating Named Data Networking for the IoT.* Proceedings of the 5th International, Volume 56, Pages 73–83, 2018

[15] A. Afanasyev and J. Shi, *NFD Developer's Guide.* NDN, Technical Report NDN-0021, 2016

[16] Omnicore, *YouTube by the Numbers* [online] Available at: https://www.omnicoreagency.com/youtube-statistics/ [Accessed 14 April 2019].

[17] Mini-NDN, *Mini-NDN Github Source Code* [online] Available at: https://github.com/named-data/mini-ndn [Accessed 15 April 2019].

[18] J. Dehart, *NDN Testbed*, Computer Science & Engineering Washington University, 2014

[19] *Named Data Networking Project, 2011* [online] Available at: https://named-data.net/publications/ [Accessed 18 April 2019]

[20] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas, *Distributed Cache Management in Information-Centric Networks*, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 10, NO. 3, SEPTEMBER 2013.

[21] E. Hemmati and J.J. Garcia-Luna-Aceves, *Making Name-Based Content Routing More Efficient than Link-State Routing*, ISBN 978-3-903176-08-9, 2018.

[22] W. Shang, Y. Yu, and R. Droms, *Challenges in IoT Networking via TCP/IP Architecture*, NDN Technical Report NDN-0038, 2016.

[23] T. Refaei, L. Zhang, and A. Afanasyev, *A Data-Centric Battlefield: Leveraging Named Data Networks in Tactical Networks*, NDN Technical Report NDN-0038, 2017.

[24] C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang, *A Fast and Memory-Efficient Trie Structure for Name-based Packet Forwarding*, Department of Computer Science, The University of Arizona, 2018

[25] *Fortune* [online] Available at: http://fortune.com/2018/10/02/netflix-consumes-15-percent-of-global-internet-bandwidth/ [Accessed 24 April 2019]

Declaration