Reducing Redundant *Data* in *SDN*-based *NDN* using Single-State *Q*-learning

Kingshuk Dhar

Department of Computer Science and Engineering

United International University

A thesis submitted for the degree of

MSc in Computer Science & Engineering
United International University
Dhaka, Bangladesh

February 2024

Declaration

I, Kingshuk Dhar, declare that this thesis titled,**'Reducing Redundant *Data* in *SDN*-based *NDN* using Single-State *Q*-learning'** is the outcome of the research carried out by me under the supervision of Dr. Shahid Md. Asif Iqbal and co-supervision of Professor Dr. Mohammad Nurul Huda. I confirm that:

- This work was done wholly or mainly while in candidature for a MSc degree at United International University and any part of this thesis has not been previously submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed. if I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help. The thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Kingshuk Dhar
ID: 012213056
Department of Computer Science and Engineering,
United International University, Dhaka-1209, Bangladesh.

Certificate

I do hereby declare that the research works embodied in this thesis entitled **'Reducing Redundant *Data* in *SDN*-based *NDN* using Single-State *Q*-learning'** is the outcome of an original work carried out by **Kingshuk Dhar, Student ID: 012213056** under our supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of MSc in Computer Science and Engineering to be awarded by United International University (UIU).

1. **Supervisor:**

   Signed: _____

   Dr. Shahid Md. Asif Iqbal
   Professor & Chairman
   Department of Computer Science and Engineering,
   Premier University, Chittagong, Bangladesh.

2. **Co-Supervisor:**

   Signed: _____

   Dr. Mohammad Nurul Huda
   Professor & Chairman
   Department of Computer Science and Engineering,
   United International University, Dhaka-1209, Bangladesh.

Abstract

Software-defined networking (*SDN*), a cornerstone of future-generation networks, is adopted in Named *Data* Networks (*NDN*) for large-scale deployment. The forwarding strategies proposed for *SDN*-based *NDN* primarily use the centralized controller to optimize *Interest* forwarding and *Data* delivery. The nodes direct the *Interest*s to the controller to discover the content source/s and suppress the sub-optimal responses. To support such content discovery and delivery, the controller experiences frequent path calculation and trades excessive control messages to install the paths in the nodes due to rapid cache admission and replacement. Besides, the typical *NDN* forwarding solutions are not viable to realize or need considerable modifications in *SDN*-based *NDN*. To that end, the proposed strategy optimizes *Interest* forwarding and *Data* delivery using a Single-State Q-learning-based technique, namely *SDN-Q*. In *SDN-Q*, each content source learns to suppress the sub-optimal responses, with the learning task offloaded to the controller. The controller communicates the learning decision to the nodes. Each node only retains the action (decision) to entertain an incoming *Interest*. Once an *Interest* hits, the source either replies with the *Data* or remains silent and sends the *Interest*'s information (meta-data) to the controller for the learning task. Thus, *SDN-Q* enables the *NDN* nodes to remain light-loaded. Each node can instantly answer an *Interest* request without redirecting it to the controller. Additionally, Interest forwarding using a hop-based scoped-flooding approach has been optimized. The proof-of-concept (POC) implementation reveals that the proposed system outperforms the competing strategies by reducing the traffic load, latency, and

control messages in *SDN*-based *NDN* at most by 40%, 7%, and four times respectively, without negotiating packet delivery ratio.

Published Papers

The outcome of this research work is recognized in the following esteemed journals:

1. Kingshuk Dhar, Shahid Md Asif Iqbal, Mohammed Nurul Huda, Nazma Akther and Asaduzzaman, 'Optimizing *Data* Delivery in *SDN*-based *NDN* using Single-State Q-learning' (Under review)
2. Nazma Akther, Kingshuk Dhar, Shahid Md Asif Iqbal, Mohammed Nurul Huda and Asaduzzaman, '*Interest* forwarding strategy in Named *Data* Networks (NDN) using Thompson Sampling', Journal of Network and Computer Applications, Elsevier, Vol. 205, 2022, pp. 103458.

Acknowledgements

Above all, I am thankful to God, the Most Gracious and Powerful, for bestowing upon me the knowledge, skill, aptitude, and well-being required to pursue an MSc in Computer Science and Engineering. His kindness has made it possible for me to finish my thesis effectively.

Next, I would want to sincerely thank Dr. Shahid Md. Asif Iqbal for being the most polite, supportive, and helpful supervisor I have ever had. Although it has been a drawn-out and difficult journey, I am grateful to have such a wonderful mentor, advisor, and colleague. I was able to do this amazing research on the topic of Future Internet Architecture, termed *SDN*-based *NDN*, where NS3-based NDN Simulator technology is new to adopt, thanks to his invaluable assistance and research ideas. Additionally, I would like to thank Dr. Muhammad Nurul Huda for allowing me to follow my own ideas during my MSc studies. Moreover, the path to the MSc in CSE would have been far more difficult without his encouragement and support. Once more, I want to express my gratitude to them for their evaluations and edits, which enabled me to publish my work and finish this excellent thesis.

I am appreciative of Premier University for enabling me to enroll part-time in my MSc program in Computer Science and Engineering. That has greatly benefited my growth as a skilled scholar and informed teacher.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Evolution of *NDN* and *SDN*-based *NDN*

The TCP/IP Internet was basically conceived to enable resource sharing among a tiny pool of communicating devices [1, 2]. The last decade experienced massive digital content growth, transforming the Internet into a content hosting and delivery network rather than merely connecting hosts. Communication in the current Internet is host-centric, requiring locating the information host (source) first. Locating a particular item in this vast catalogue of stacked content using the current solutions is not scalable [3], limited and complicated [4]. In addition, there are proven problems in the typical Internet regarding content security, availability [5], user anonymity, and so on. To overcome the stated limitations, a drastic modification of the legacy Internet is a must. Information-centric networking (*ICN*) appeared as a hopeful and clean-state networking architecture proposed for modelling the next-generation Internet. *ICN* drives away from the host-centric approach to a content-centric paradigm, where each object is directly and uniquely addressable. The basic elements in this paradigm are the objects (contents) identified by their unique names rather than the machines (for example, servers, routers) identified by their IP address. *ICN* has several benefits such as location-free naming, name-based routing, content hiding, multicast, native data security [6], user anonymity, etc. Hence, *ICN* networks can efficiently deliver better quality contents to the users and improve the network capability management [2].

Among the *ICN* proposals, Named *Data* Networking (*NDN*) is versatile for its design, maintenance, continuous development, and broad developer forum [7]. In the *NDN* principle, each router's forwarding plane is considered stateful to support intelligent forwarding and on-path caching [8]. These unique characteristics enable content finding using hierarchical names and shorten the item retrieval time. *NDN* commutes two distinct kinds of packets: *Interest* and *Data* to solicit and retrieve an item, respectively. Any *Data* item is generally divided into several small, equal-sized, and cachable chunks or pieces. An *NDN* node is equipped with three distinct data structures, Content Store (*CS*), Pending *Interest* Table (*PIT*), and Forwarding Information Base (*FIB*), which are consulted in turn to forward *Interest*s and reply *Data* during communications [5, 9].

Each *Interest* is uniquely tagged with the desired object's name and a random nonce value by the consumer app application. A *Data* packet carrying the desired object digitally signed by the content provider, binding the name with the content payload. The (*FIB*) contains name prefixes and paths to reach those prefixes used to forward an *Interest* packet. The forwarding strategy pre-installed in each node is responsible for making forwarding decisions. A Pending *Interest* Table (*PIT*) entry tracks the forwarded (and unanswered) *Interest*s and waits for the corresponding item to return. The waiting table aggregates *Interest* forwarding and is used as bread-crumb trails to deliver the returned *Data* packet. Since the nonce value uniquely identifies an *Interest*, the *PIT* can detect and stop *Interest* looping. By its name, a Content Store (*CS*) holds replicas temporarily to entertain future requests early [9].

A consumer application uniquely tags each *Interest* with the desired object's name and a random nonce value. A content provider digitally signs the *Data* packet carrying the desired object, binding the name with the content payload. The (*FIB*) contains name prefixes and routes to reach those prefixes used to forward *Interest*s. The forwarding strategy pre-installed in each node is responsible for making forwarding decisions. A *PIT* entry tracks the forwarded (and unanswered) *Interest*s and waits for the corresponding item to return. The *PIT* aggregates *Interest* forwarding, and its entries are used as footprints to deliver the returned *Data*. Since the nonce value uniquely identifies an *Interest*, the *PIT* can detect and prevent *Interest* looping. A consumer transmits an *Interest* advanced upward by

the intermediate nodes until the *Interest* strikes a content source. A node may use flooding or single-path forwarding [10] to push the *Interest* upward. The solicited item is returned to the solicitor along the reverse path, using the *PIT* footprints. The *CS* retains content copies temporarily to serve forthcoming *Interest*s quickly [9].

Software-defined networking (*SDN*), a hot topic in the *NDN* community too, is transforming the network design and management by decoupling the control plane from the data plane. *SDN* comprises a collection of innovative technologies that dissociate the network resources from the physical aspects of the network to dramatically enhance the programmability, flexibility, orchestration and virtualization of those resources [6, 11, 12].

Optimal *Data* replying is formulated as a Singe-State Q-learning problem in *SDN*-based *NDN* here to minimize the traffic load and content retrieval delay by suppressing the sub-optimal replies.

The Singe-State Q-learning [13, 14] assumes a single state for an agent. The agent is asked to perform an action, and it receives a reward for that action. The reward is used to estimate the Q-value for that action. The agent selects the action that yields the highest Q-value. Since rewards are unknown, an agent can learn the Q-values by experimenting with the actions. As an action is performed over time, the corresponding Q-value is updated, resulting in a more accurate estimation. The objective of learning here is to maximize the Q-value over a given period.

1.2 Motivation

The previous results, namely *SRAB*^{3} and *SDN-SRP*, motivate the current study. However, there are basic differences between *SDN-Q* and *SDN-SRP* and *SDN-Q* and *SRAB*^{3}. *SRAB*^{3} requires a consumer to direct control packets (*AckNode* messages) to the content sources, used to determine the payout for learning (Thompson Sampling) purposes. The content sources solely carry out the learning task. In contrast, *SDN-Q* requires the content custodians to send control messages to the controller when an *Interest* strikes, saving the computation burden of *NDN* nodes. As the control messages use an out-of-band connection, *SDN-Q* results in

a relatively reduced communication cost (overhead). Again, *SDN-SRP* requires the exchange of control messages between the controller and nodes to serve every *Interest* (analogously suppress redundant replies), which may adversely affect the performance, especially in large-scale *SDN*-based *NDN*. For instance, the magnitude of control packets may grow extremely large. *SDN-Q* doesn't invoke the controller in real-time to conceal the suboptimal custodians.

1.3 Research Problem

The limited scalability of the current *NDN* forwarding strategies challenges their feasibility in large-scale *NDN* deployment [15]. In the last decade, considerable research work has been attempted to enrich and scale *NDN* network functionalities using *SDN*.

Introduction of an *SDN* controller in *NDN* [6, 16–19] enables the unicasting of *Interest* and *Data* packets in a more optimized path than the traditional *NDN* forwarding schemes. The optimization reduces network traffic and improves *Data* delivery efficiency [20] at the cost of rapid path computation at the controller. Later, the path installation in the nodes generates tremendous control packets [21]. Furthermore, many of these solutions require a node to inquire the controller for paths to the content sources, in the absence of such information, in its *FIB*, increasing the retrieval delay. Again *SDN-SRP* demands real-time controller involvement to answer an *Interest* from an optimal source. To that end, this study proposes a content retrieval strategy in *SDN*-based *NDN* that suppresses redundant replies; however, avoiding real-time controller involvement and path computation at the controller and utilizing fewer control messages.

This research assumes the optimal *Data* replying strategy by a content custodian as a Single-State *Q-learning* model, denoted as *SDN-Q*. The model considers a content custodian as an agent asked to react with one of the two actions, namely *Mute* or *Reply*, once an *Interest* arrives. An agent receives a reward for playing an action and uses it to compute a Q-value; it selects the action that yields the higher Q-value. Initially, an agent is ignorant about the Q-values of the actions and learns the efficiency of actions by responding to an *Interest* and inspecting the rewards. The Q-values are used to select actions for the next rounds (successive

*Interest*s). *SDN-Q* aims to optimize the traffic load while decreasing the download delay in *SDN*-based *NDN*.

1.4 Contribution

In this paper, we primarily devise a *Data* replying episode in the *SDN*-based *NDN* paradigm to retrieve a content from an optimal custodian minimizing traffic load and retrieval delay. A custodian assumes a Single-State Q-learning model to learn the optimal response (action) to entertain a requesting *Interest*. Furthermore, we optimize the *Interest* forwarding (flooding) episode freeing a thread of an *Interest* like [22], called *Ray*, to look for a content source in the entire network while restricting the other branches to a certain distance (hops). A node forwards the *Ray* through the interface ranked highest. The interface ranking method and flooding scope selection are completely different from $SRAB^3$. The key contributions are as follows:

- **Enhanced Data delivery strategy:** The research introduces a refined approach within Software-Defined Networking *(SDN)*-based Named Data Networking *(NDN)* that efficiently manages traffic load and reduces content retrieval delay by eliminating sub-optimal replies.
- **Interest flooding optimization:** A novel method is proposed to enhance Interest flooding scope within *SDN*-based *NDN*, leveraging optimal custodians to improve network efficiency.
- **Single State Q-learning:** This study introduces Single State Q-learning for content retrieval in *SDN*-based *NDN*, simplifying controller data structures and outperforming competitors in latency, traffic load, and control messages.

1.5 Organization of the Thesis

The remaining portion of this thesis is organized as follows. The second chapter begins with a brief overview of notable *ICN* projects. Since the focus of our research contributions is the *SDN*-based *NDN* tenet, it also provides a comprehensive

overview of the *NDN* architecture. Later in the chapter, the literature concerning present solutions for *NDN* and *SDN*-based *NDN*.

The Chapter 3 describes the preliminaries and the proposed *Interest* forwarding and answering in *SDN-Q*. This chapter demonstrate the methodology regarding *Interest* Forwarding and Answering in *SDN*-based *NDN*. Moreover, chapter 3 represents a prominent reinforcement learning algorithm called Singe-State *Q*-learning for *Data* response (*Mute* or *Reply*) strategies in *SDN*-based *NDN*.

The chapter 4 describes the simulation execution in the famous ndnSIM simulator. Furthermore, performance analysis are demonstrated based on transmission overhead, latency, cache hit ratio, cache eviction rate, and combined significance.

The chapter 5 concludes the research with contributions, constraints and remarks on the future research needs for forwarding in *NDN* and *SDN*-based *NDN* architectures.

Chapter 2

Background and Related Work

2.1 Background

Information-Centric Networking (*ICN*) would concentrate on content attributes rather than the precise location of host-centric *Internet* communication, marking a significant departure from traditional approaches. Prioritizing "what" content is over "where" it comes from, *ICN* aims to revolutionize the conventional host-centric architecture. By adopting a content-centric approach, content could be detached from physical network locations, transportation modes, storage types, and application layer programs. This design enhances resilience in challenging network environments, ensuring adaptability in dynamic scenarios. Additionally, *ICN* offers improved scalability in bandwidth demand and enhances content retrieval efficiency, promising a more robust and efficient communication framework for the future.

This chapter covers the popular future Internet architectures that have been proposed in the literature, with a focus on *Named Data Networking* (*NDN*) based on *SDN*, where our *Interest* forwarding technique is suggested and implemented.

2.2 *ICN* Fundamentals

In-network storage is used by *ICN* systems for replication-based multiparty communication and caching. With the *TRIAD* serving as the pioneer architectural

project, the *ICN* paradigm was introduced in 1999 [23]. Since then, a number of intriguing architectures have been discussed in the context of particular projects while utilizing the same paradigm. These include Content-Centric Networking (*CCN*) [24], *PURSUIT*(*PSIRP*) [25], *NDN* [26, 27], *DONA* [28],*CONVERGENCE* [29],*SAIL* [30],*COMET* [31], *NetInf* [32], and many others. Important ICN suggestions are briefly summarized in this section.

2.2.1 Data-Oriented Network Architecture (*DONA*)

DONA uses information routing at high levels of inter domain architecture as its technique. It entails a total redesign of name resolution and Internet naming. Consequently, an information-focused layer called Resolution Handlers (RHs) is built above the Internet. The order of *DONA*'s names is based on principals. Every service and other named entity (hosts, domains, etc.) has a principal and a pair of public and private keys associated with it. The names are input by a principal into the resource handler (RH) locally. Name resolution service is provided by a content router, or RH, which is hierarchically connected to other RHs. The network propagation of the producers' REGISTER messages updates all RHs with the location of a particular content object. Interested parties can send a FIND message to the associated RH to retrieve specific information. The relevant resource handler is instructed to handle the FIND request by the RH. After that, the content object is returned to the client that made the request either directly or via the FIND message's reverse path.

On-path caching in *DONA* is implemented via the RH network. When a RH decides to cache an item, it modifies the source network address of the FIND message to its own self-network address and passes the message to the next RH in line. Since the requested material is now being provided by the current RH, it has the chance to accept the *Data* in its cache.

2.2.2 The Publish-Subscribe Internet Routing Paradigm (*PSIRP*)

The goal of the *PSIRP* project is to develop and evaluate an information-centric Internet architecture for the future. The objective is to create a new kind of

internetworking schema that not only offers the required functionality, flexibility, and performance but also enables creative applications, fresh commercial prospects, availability, security, and mobility.This represents a joint effort between the internet routing paradigms of *PSIRP* and *PURSUIT*. In continuation of the *PSIRP* [25, 33, 34],the *PURSUIT* [35] is a 2010-started EU FP7 funded project. In *PURSUIT*, the conventional send-receive-based Internet communication is replaced with the publish-subscribe protocol stack.

Publishers in the *PURSUIT (PSIRP)* architecture provide content objects to the PSI network. Subscribers are customers who send subscription request messages to indicate their interest in a certain product. An essential part of the architecture is the Rendezvous Point (RP), which carries out the mapping from a subscriber interest to the corresponding content item. Similar to *DONA*, *PURSUIT* names are flat and are recognized by a collection of characteristics (scope ID, rendezvous ID). The specific information component is represented by the rendezvous portion of a name, while the reach or availability of a content object is shown by the scope section. A content's scope can be logical, like only accessible to a particular user group, or physical, like only available at a specific campus location. Other examples of scope include authorization, access rights, reachability, availability, and so on. RPs have the ability to hold subscription requests for unpublished content for a predetermined amount of time. Customers can express subscription curiosity for *data* items that have not yet been added to the network by using this subscription option, which spatially decouples the subscription and publication operations.

2.2.3 Network of Information (*NetInf*)

Named Data Objects (*NDO*) can be retrieved using two distinct techniques in this architecture: name-based routing and name resolution. Named data objects (*NDO*s) are a first-order networking primitive that can be accessed via the *NetInf* networking approach. The primary job of NetInf nodes is to transmit (*NDO*) requests and the associated objects. PUBLISH, GET, and SEARCH for (*NDO*s) are supported via the message-based NetInf protocol, which offers requests and replies. A source notifies the contents' availability via the PUBLISH message in

accordance with the model configured in the local network, making sure that the name resolution service (NRS) can register the appropriate name-location binding or that the routing protocols can disclose the object's routing information. The name-based routing technique is quite easy to use. The request (GET message) for the required content is sent directly to its source locations in accordance with the routing protocol. The contents are sent from the sources to the requester end using name-based routing as well.

In contrast, the name resolution approach suggests that users should investigate the available finders of the identified object by expressing their interest (via a SEARCH message) to the NRS. Subsequently, NRS presents a compilation of search tools that the user can utilise to locate the artefact from the primary source. The new *NetInf* model enables a seamless shift from the traditional location-dependent Internet architecture to a fully name-based Internet architecture. *NetInf* can be implemented by either launching the two models individually or by integrating them into a hybrid model that enables hop-by-hop flipping between the two models [36].

In contrast to previous suggestions from *ICN*, *NetInf* allows for the caching of both request messages and objects. Upon receiving a GET message, a node has the option to either conduct a per-request NRS lookup or aggregate the request by utilising a waiting table. Alternatively, a registered copy of a data object can be obtained either directly from the NRS or by employing alternative means to replicate the object within a node. Furthermore, it is possible to obtain a duplicate by utilising a cache-aware *NetInf* transport protocol when journeying to a destination where duplicates are known to be stored [37].

2.2.4 Content-Centric Networking (*CCN*)

In a *CCN* architecture, a name represents a data object. To obtain a particular object, a client can send a *Interest* message with the name of the desired data item. The client transmits the *Interest* message to the network, which is then routed and forwarded until it reaches a node that possesses a replica of the object associated with the *Interest*. The originating node, which possesses the duplicate, generates a *Data* packet containing the replica of the requested item and transmits

it to the requester. The *Data* packet traces the path of the *Interest* to reach the client. An intermediate router located along the path may retain a duplicate of the *Data* while transmitting the packet back to the client. This is done to fulfil future requests *(Interests)* from other clients for the same *Data*.

The Content-Centric Networking (*CCN*) initiative, initially introduced by Van Jacobson in 2006, provided the basis for the development of *NDN* [26, 27]. This is a component of a substantial collaborative research endeavour conducted by multiple academic and industrial entities. Nevertheless, both approaches have several core principles in common, including the retrieval of content based on its name, the use of a *Interest* to indicate a request, and the delivery of content through a *Data* packet. Although the *NDN* architecture and *CCN* architecture initially had common design patterns, the *NDN* architecture has now diverged from the *CCN* architecture and is evolving in a unique direction. When an *NDN* router forwards *Interest* packets and searches for matching content in caches, it looks for the longest prefix match [38]. On the other hand, in order to facilitate the process of making forwarding decisions, *CCN* utilises precise matching for route lookup. The *CCN* and *NDN* proposals in the literature are notable for their extensive and passionate research community as well as their open-source availability, making them stand out among other *ICN* proposals.

2.2.5 Named Data Networking (*NDN*)

Though entirely novel in terms of network architecture, *NDN* was designed with the perfection of the modern Internet as its primary goal. Named Data Networking (*NDN*) is one of the five projects funded by the American National Science Foundation (*NSF*) Future Internet Architecture (*FIA*) Program [26, 27]. Investigating Jacobson's groundbreaking proposal of a drastic move away from the conventional host-centric network design (*IP*) and toward a content-centric network architecture (*NDN*) is the aim of the *NDN* project. The TCP/IP Internet's hourglass shape features a slim waist at the network (*IP*) layer, where universal connectivity can be established with only the bare minimum of capability.The Internet's rapid expansion as a communication network, which separated the advancement of lower- and upper-layer technology, was largely attributed to

the narrow waist theory. However, IP was created to provide an end-to-end communication link where the communication endpoints' *IP* addresses are used to name and route packets.

At the moment, the Internet is more important than its analogue in serving as a more widespread distribution network. Using a point-to-point communication network to address distribution issues is difficult and prone to errors. The goal of *NDN* is to increase the ubiquity of the slender waist role, allowing packets to be labelled and routed based on the name of the object instead of communication endpoints.The *NDN* architecture's hourglass shape is retained, but the narrow waist is altered to focus on the data itself rather than its location. More specifically, *NDN* changes the semantics of network communication so that data identified by a given name is retrieved instead of a packet being sent to a specific destination node. Fig. 2.1.

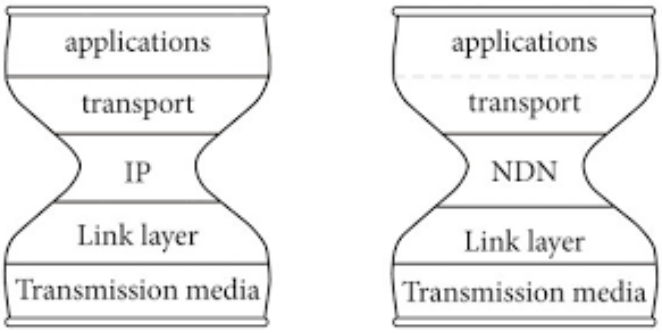


Figure 2.1: *IP* and *NDN* hourglass architectures

2.3 *NDN* Architecture Overview

The network-layer protocol *NDN* has a receiver-driven architecture. Like *IP*, the slender waist is the primary focus of the *NDN* architecture. *NDN* provides a new set of minimum functionalities because it delivers and retrieves material using data names rather than *IP* addresses. There are notable distinctions between *IP* and *NDN* data delivery procedures introduced by the new set. The different *NDN* components and aspects are introduced in this section along with their functions within the overall architecture.

2.3.1 *NDN* Packets

Using the specified name, a piece of data is identified and retrieved in compliance with the *NDN* services' semantics. Any exchange of information between two entities in *NDN* is receiver-driven, and any network entity can be referenced by the name. For *NDN* communications, there are two main packet types: *Interest* and *Data*. The two distinct packet types' packet formats are shown in Figure 2.2. The names of the individual data items that can be sent in a single *Data*

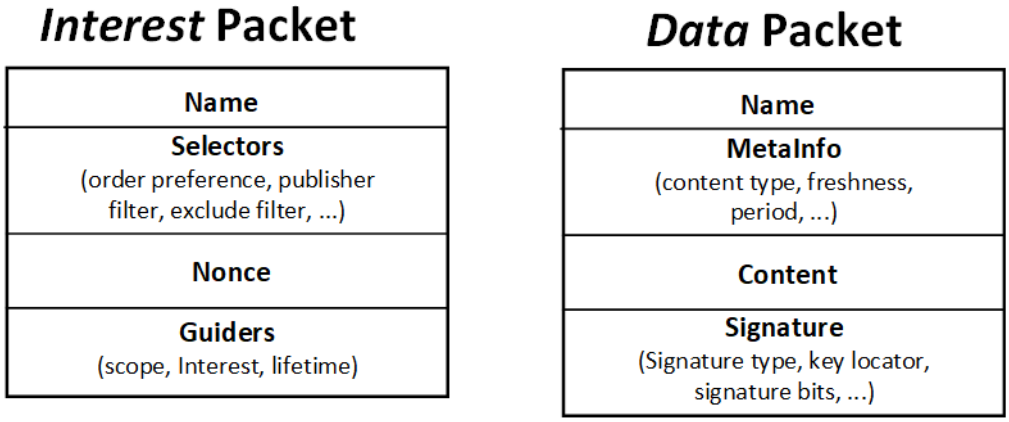


Figure 2.2: Packets in *NDN* architecture

packet are assigned to each type of packet. The name of the data a user wishes to receive must be included in a *Interest* packet that they submit to the network. To request the desired item, a customer sends out a *Interest* packet, which is a little envelope that looks like the TCP acknowledgement messages. Once the preferred data name is entered, the consumer creates a *Interest* packet and sends it across the network. An *Interest* packet just needs the name; however, it also includes other important data that aid in *Interest* matching and forwarding. Only the message name and the randomly generated value in the nonce field can be used to identify a *Interest* message. It is primarily intended to detect and prevent *Interest* looping when using a nonce value. More details about the requested data are provided in the selected area. The amount of time left until the *Interest* is timed out and dropped is indicated by the lifetime and scope fields.

A copy of the requested item on a network node can generate the *Data* packet once the corresponding *Interest* happens. The node can serve as a router for

caching or as a producer. The source node creates and sends a *Data* packet in response to the matching asking *Interest*. A *Data* packet can be cached by an NDN router to fulfil incoming requests. The name, some arbitrary data, some optional data (such as meta-data), and the cryptographic signature that encrypts the data and connects it to the content are all included in the *Data* packet. A *Data* packet's name is the first field in order to simplify packet processing. Given that the cryptographic signature computation takes into account all components that came before it, the signature comes at the conclusion of the packet.

2.3.2 *NDN* Naming

The routers utilise *NDN* data names in an *Interest* to steer the request towards potential content providers. *NDN* names are considered opaque to the network, indicating that routers are merely cognizant of the divisions between the different parts of a name, but not its actual significance. As a result, naming schemes can evolve separately from the network, allowing each application to choose the name scheme that most effectively meets its needs. Each individual component of a data item that is included in a single *Data* packet [39] can be uniquely identifiable by a name. The name of each *NDN* consists of many constituent elements that are organised in a hierarchical manner. A name component is usually a string of variable length that represents text in a way that can be easily read by humans. The forward slash "/" is used both to separate and combine the parts of a name. The hierarchy of a name component is also defined by the same delimiter. The current binary encoding and application-layer rules for content naming are displayed in Figure 2.3.

The version creator is represented as RS followed by a numerical version number denoted as v. The segmentation marker is set as 00, followed by an integer number that represents a specific frame number, block number, or byte number for a segment of the video content[39]. Moreover, the application has the ability to integrate all contextual information and data element relationships into the names.

The hierarchical naming structure enables any application to establish its own preferred name format, tailored to its specific requirements. This structure also

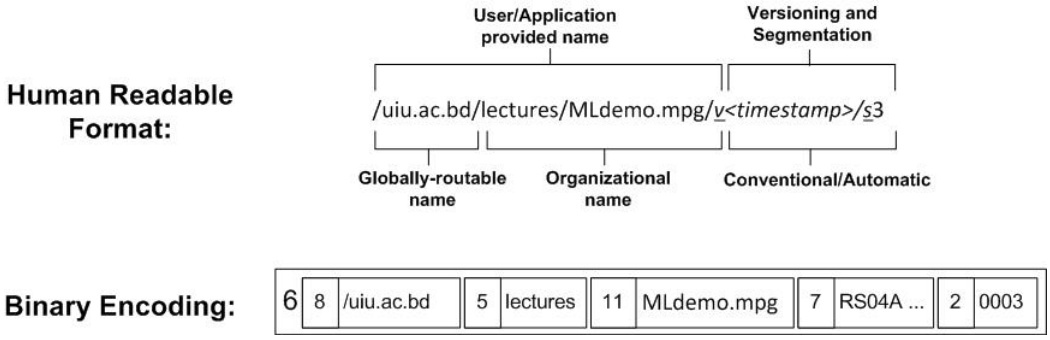


Figure 2.3: Example *NDN* name format, from [39].

facilitates the independent evolution of naming schemes. The application can additionally incorporate any relevant information and connections between the data pieces in the names. The autonomous content hosting system is depicted by the hierarchical name tree shown in Figure 2.4.

To fetch any arbitrary data, consumers must really be able to generate the name of the required data chunk without having to know the name of the content beforehand. Deterministic algorithms can be used by the producer and the consumer to generate the same name based on shared knowledge. Otherwise, the desired component can be obtained in one or more rounds by utilizing *Interest* selectors together with the longest prefix matching [36]. Zhang *et* al. [27] claim that only a basic set of selectors and a limited quantity of naming information are needed to retrieve a data packet. A section of the name tree connected to the convention depicted in Fig. 2.3 is represented by the hierarchical tree in Fig. 2.4.

As an illustration, a student who wants to access the latest version of the MLdemo.mpg video lecture from UIU can make a request for /uiu.ac.bd/lectures/MLdemo.mpg/v with the *Interest* selector set to "leftmost child". In response, they will receive a *Data* packet named /uiu.ac.bd/lectures/MLdemo.mpg/v1/s1, which identifies the first segment. The client can submit additional requests that include information from the initial *Data* packet and adhere to the naming convention employed by the publishing application [27].

To obtain data on a global scale, only names that are unique worldwide are necessary. With the exception of these names, there is no requirement for other names to be unique worldwide. Local communication names are important inside

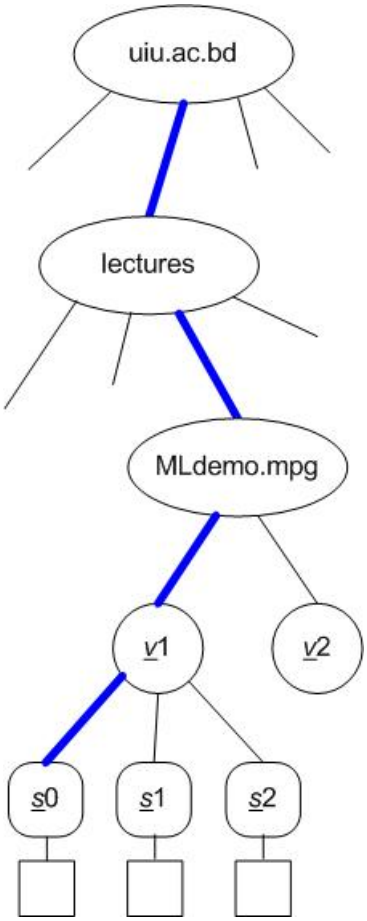


Figure 2.4: *NDN* node data structure, from [39].

a specific area and only involve limited routing or local broadcast to get the corresponding data. Data naming is the paramount element of the NDN paradigm. Thanks to *NDN*, it is possible to support a range of network functionalities, such as simultaneous multi-casting of the same *Data* to multiple users, distributing the same content at different times, accommodating user mobility, and retrieving content even when there is intermittent connectivity (delay-tolerant networking).

2.3.3 *NDN* Data Structures

Each *NDN* router utilizes three primary data structures to handle *Interest* and *Data* packets: the Content Store (*CS*), the Pending Interest Table (*PIT*), and the Forwarding Information Base (*FIB*) [40–42]. The three structures are essential

elements of the *NDN* forwarding engine. Figure 2.5 illustrates the data structures and forwarding engine.

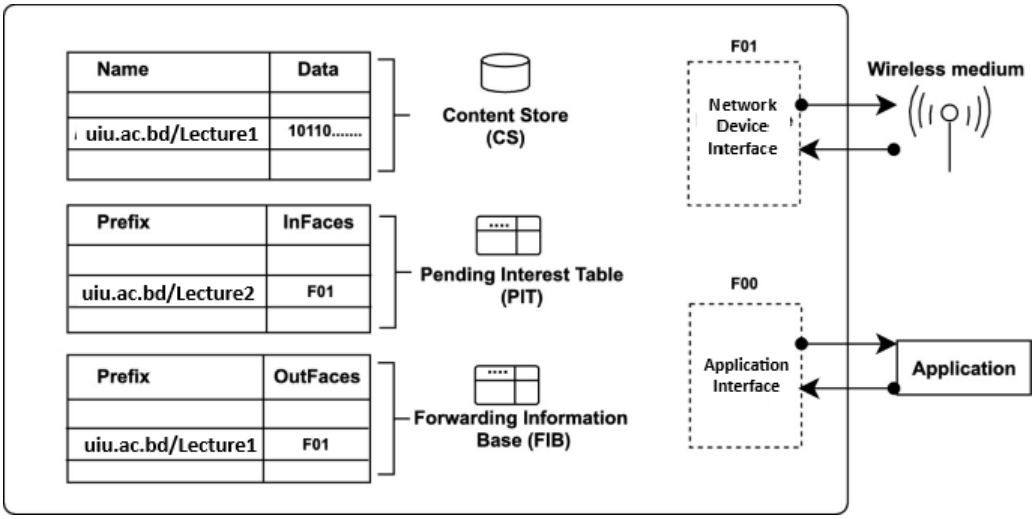


Figure 2.5: *NDN* forwarding engine model, from [39].

2.3.4 Content Store (*CS*)

The Content Store of a node is the designated storage location where *Data* packets are briefly held before to being transmitted to the subsequent node for processing incoming requests. Within an *NDN* node, the sole component that allows for caching within the network is a *CS* (Content Store). Usually, the *CSs* can only retain a small fraction of the vast quantity of content that is accessible. The term "content name" is utilized to index and retrieve an entry in the *CS* database.

2.3.5 Pending *Interest* Table (*PIT*)

Users have the ability to anonymously request and obtain content using *NDN*. This implies that neither the user's address nor their name is included in the Interest packet, which is used to identify or locate the requesting consumer. Instead, the consumer obtains a response to the returned *Data* packet by tracing the trails of the routers' Pending Interest Table (*PIT*). When a "Interest" is transmitted and awaiting the answer *Data*, it is documented in a *PIT* item. A router constantly

checks incoming interfaces for pending or unsatisfied *Interests*, which are attempts to push forward and reach possible data source(s) but have not been fulfilled yet. Upon receiving matching data, these records are utilized to transmit the *Data* packet to the consumer(s) making the request. The *Data* identifier is utilized to reference an entry within this waiting table. An individual *PIT* entry is generated for every name that is requested. Each entry in the record consists of a forwarded Interest's name, a list of incoming interfaces, and a list of outgoing interfaces. This record is represented as a three-element tuple. A *PIT* element additionally contains a roster of nonce values that have been uncovered for that particular name. Due to the presence of these tuples, the *PIT* acquires statefulness, enabling the forwarding of similar *Interests* to occur just once.

Users have the ability to anonymously request and obtain content through the usage of *NDN*. This implies that neither a user's address nor their name is included in the *Interest* packet, which is used to identify or locate the requesting consumer. Instead, the consumer obtains a reply to the returned *Data* packet by tracing the trails of the routers' Pending Interest Table (*PIT*). When a *Interest* is transmitted and awaiting the response *Data*, it is documented in a *PIT* item. A router consistently checks incoming interfaces for pending or unsatisfied *Interests*, which are attempts to push forward and reach possible data source(s) but have not been fulfilled yet. Upon receiving matching data, these records are utilized to transmit the *Data* packet to the consumer(s) making the request. The *Data* identifier is utilized to reference an entry within this waiting table. An individual *PIT* entry is generated for every name that is requested. Each entry in the record consists of a forwarded *Interest*'s name, a list of incoming interfaces, and a list of outgoing interfaces. This record is represented as a three-element tuple. A *PIT* item contains a record of nonce values that have been found for that particular name. As a result of these tuples, the *PIT* gains statefulness, which enables the forwarding of similar *Interests* to occur just once.

2.3.6 Forwarding Information Base (*FIB*)

Although the *FIB* of the *NDN* router shares many similarities with the *FIB* of the IP router, it varies from it in two significant areas. Unlike an *NDN FIB*, an IP

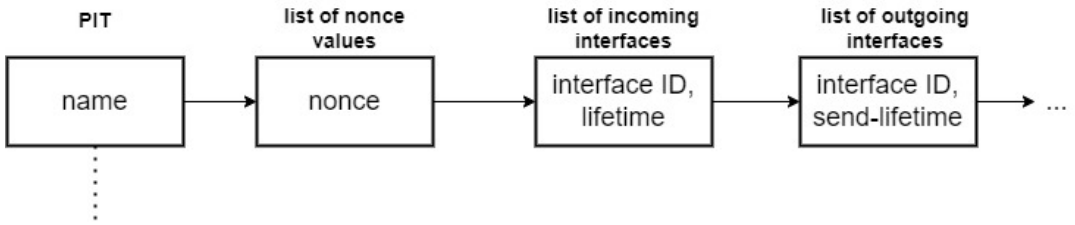


Figure 2.6: Forwarding state in *PIT*, from [43].

FIB entry typically only stores a single optimal next-hop instead of a prioritized list of numerous interfaces. Secondly, an IP Forwarding Information Base (*FIB*) entry simply includes the next-hop information, but an *NDN FIB* entry enables adaptive forwarding decisions by gathering information from both the routing and forwarding planes. When a name prefix is reported during routing, a new entry is added to the Forwarding Information Base (*FIB*), as illustrated in Figure 2.7. Every *NDN* node contains a Forwarding Strategy module that determines the destination of an incoming Interest packet by analyzing information from the Content Store (*CS*), the Pending Interest Table (*PIT*), and the Forwarding Information Base (*FIB*).

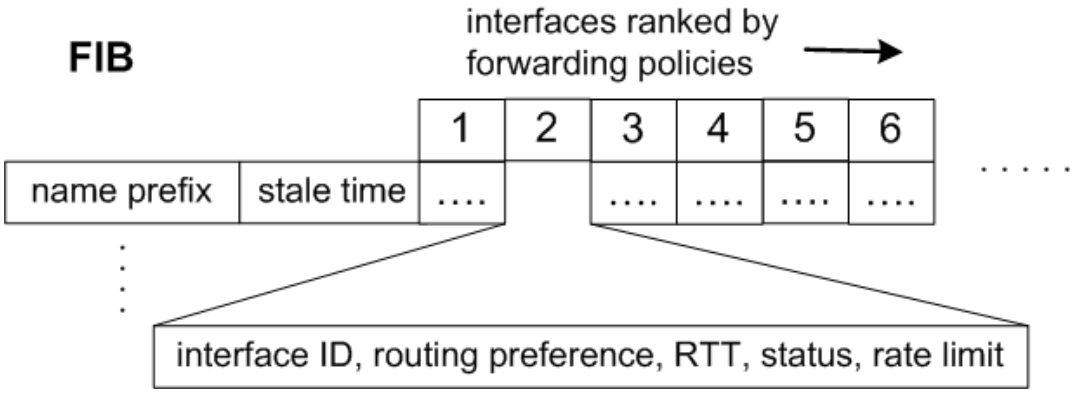


Figure 2.7: Forwarding state in *FIB*, from [43].

2.3.6.1 *NDN* Forwarding Process

In *NDN*, each time an *Interest* or a *Data* packet reaches an intermediate node, the node processes the packets and decides whether to forward them in accordance

with the procedure shown in Fig. 2.8 In the *NDN* architecture, when an *Interest* or *Data* packet reaches an intermediate node, the node examines the packets and determines whether to advance them based on the mechanism illustrated in Figure 2.8. . Upon the arrival of a *Interest* packet, a router initiates a search

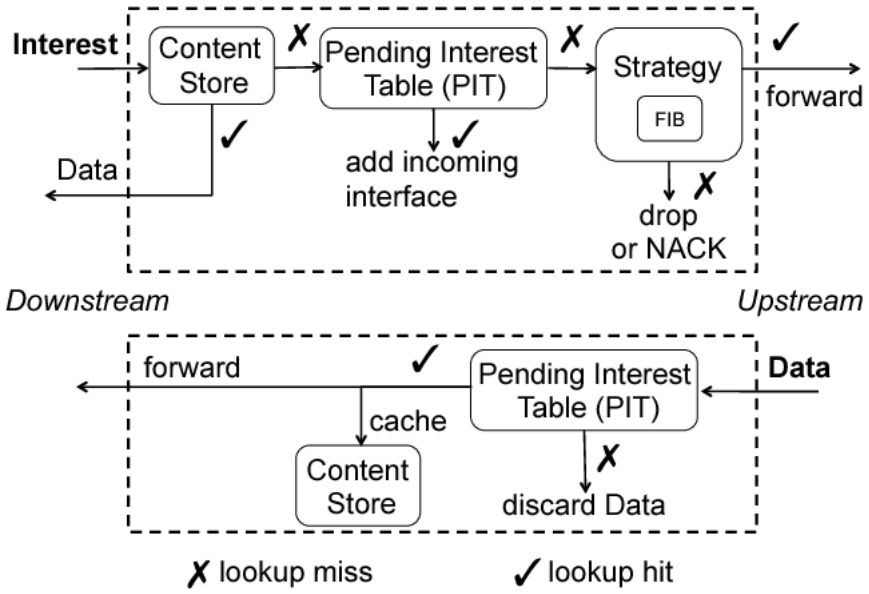


Figure 2.8: Forwarding state in *FIB*, from [27].

in the *CS* to find a suitable match. If a lookup hit occurs, signifying that the *Interest* can be satisfied from the local cache, the intermediary node transmits the required *Data* over the incoming interfaces of the *Interest*. If the router cannot fulfill the request within its own network, it conducts a *PIT* lookup. If there is a corresponding *PIT* item, the *Interest* is discarded if the same *Interest* is sent back over a different interface. On the other hand, if the *Interest* is generated by a different client using a different interface, the interface through which the *Interest* arrives is included in the list of requesting interfaces in the corresponding *PIT* item, and the *Interest* is subsequently suppressed. When a router encounters a *PIT* lookup failure, it generates a new *PIT* entry for the incoming *Interest*. This failure occurs when the *Interest* name is not found in the *PIT*. The router thereafter does a Forwarding Information Base (*FIB*) query in order to transmit the requesting *Interest*. If the name does not relate to any items in the *FIB*, the *Interest* is disregarded. Alternatively, as decided by the forwarding strategy

module, the requested *Interest* is advanced based on the interfaces specified in the corresponding *FIB* entry.

Upon receiving a *Data* packet, an intermediate router immediately does a *PIT* lookup. If the router detects an unsatisfied *Interest* with the same name already present in the *PIT* table, it sends the *Data* packet to all the interfaces through which the interest originated, marks the *Interest* as satisfied, and removes the corresponding *PIT* entry. Consequently, the matching *Data* packet always takes the identical path as the requesting *Interest*, but in the opposite direction. If there are no entries that match the arriving *Data* packet, the packet is suppressed since it is regarded to be an unsolicited packet. Every *Interest* issued by a consumer has a specific duration, and if a timeout happens before it is fulfilled, any *Interest* that is still pending in the *PIT* table is removed. Upon receiving a *Data* packet, a router places it into the network cache based on its caching strategy. Because in-network caching is integrated into *NDN*, it accelerates the process of retrieving content by allowing subsequent content requests or re-transmitted *Interests* to be fulfilled straight from the router's cache instead of having to travel back to the producer.

2.3.7 IP and *NDN* Routing

In a TCP/IP architecture, a router directs a *Data* packet by considering the source and destination Internet Protocol addresses. An IP router consists of a data plane and a routing plane. The intelligent routing plane has complete authority over the dissemination of IP addresses for both broadcasting hosts and associated networks across the network. The routing plane is responsible for calculating the path(s) to networks based on the received ads. The aircraft disseminates routing notifications throughout the entire network by employing routing protocols like as Distance Vector or Link State, BGP, OSPF, and others. By utilizing the incoming ads, every router chooses the most optimal route (next hop), saves the routes in its *FIB*, and subsequently calculates the next hop. In addition, the address-based routing system must possess the capability to identify the address space, conceal or expose public or private addresses, and effectively manage network address translation (NAT) issues, among other functionalities.

Consequently, the IP Routing Information Base (RIB), which bears resemblance to the NDN FIB table in terms of look, contains a solitary optimal next-hop and pertinent data. Hence, it is imperative to update the RIB table anytime there are modifications in the network, such as changes in topology or policy. The responsibility for this lies with the IP routing plane. Therefore, the routing plane is both capable of maintaining state and adjusting to changes. Although the forwarding plane is limited to unidirectional motion and lacks intelligence, it successfully performs all the essential functions. Additionally, the data plane utilizes a single interface to transmit packets that adhere precisely to the RIB table [43]. In IP networks, the intelligent plane has full responsibility for packet delivery.

During a network change, such as a route failure, the router exchanges routing updates to recover from the disruption and establish consistency. The time period from when network failure is detected to when all routers receive an update is referred to as the routing convergence time. Once the network has regained its functionality, the transmission of packets can recommence promptly, resulting in a minimal convergence time and mitigating the risk of packet loss. However, achieving rapid convergence in extensive networks is challenging, especially when considering routing stability and scalability.

Scalability is essential for supporting extensive networks consisting of multiple nodes or routers, IP prefixes, and links. Routing stability is crucial for managing applications that are impacted by fluctuations in round-trip time (RTT). Thus, achieving simultaneous rapid convergence, stability, and scalability is a complex task. Distance-vector routing techniques ensure more scalability but have slower convergence by utilizing the hop distance between routers (without considering the network topology) as a path measure for routing and forwarding packets. Conversely, link-state routing algorithms exhibit rapid convergence but suffer from limited stability and confined scalability due to each router being aware of the entire network topology.

Due to its lack of native support for multi-path routing and the need for additional routing to assure loop-free pathways, IP is not the ideal choice for multi-path routing. The routing plane is responsible for guaranteeing loop-free

pathways and quick convergence in the presence of looping packets, which requires significant resources and consumes CPU power.

2.3.7.1 *NDN* Routing Plane

NDN resolves the problems associated with IP addresses by directing and transmitting a *Data* packet based on its name. Due to the unlimited nature of the *NDN* namespace, there is no issue of address depletion. Moreover, there is no problem with NAT traversal because the routers transmit data to the requesters by following the paths in the PIT, instead of requiring a host to disclose its private address in order to provide content. Ultimately, within a Local Area Network (LAN), the need for address management and allocation is eliminated. Unlike IP, which declares address prefixes that encompass the content the router is ready to provide, *NDN*'s routing plane declares name prefixes. The content providers initiate the announcement of name prefixes, and the other routers calculate the routes to those prefixes directly. A routing protocol is employed to disseminate the announcement over the network, and each router that receives it constructs its own Forwarding Information Base (*FIB*) [39]. The routing plane is responsible for populating and updating the Forwarding Information Base (*FIB*). Link-state, OSPF, and other routing protocols designed for IP should also be compatible with *NDN*. The routing plane of *NDN* should be redesigned to accommodate its intelligent and adaptive data plane, which will be discussed further. The existing routing plane exclusively facilitates the forwarding plan by populating the Forwarding Information Base (FIB).

2.3.7.2 *NDN* Forwarding Plane

The data plane in *NDN* is responsible for two things: first, it sends consumer *Interest* around the network, and second, it pulls pertinent *Data* down the same path but in the other direction. The routers forwarding the *Interest* preserve an entry in their *PIT* in order to maintain the forwarding state-full. Consequently, by symmetrically sharing interest-data and preserving a forwarding state at routers, *NDN* aids in the implementation of adaptive forwarding [43, 44]. By examining the *Data* packets that are returning and the forwarding state data (*PIT* entries),

a *NDN* router can evaluate packet delivery performance, including round-trip time and throughput. A router can detect problems that result in packet losses as well as link failures or congestion. Let's say, for example, that the *Data* request is not received over the same interface that sent the *Interest*, or that it is not received in a timely manner. If so, the router notices that as a possible problem: a connection being lost due to congestion, link failures, loops, and other reasons.

Decisions regarding *NDN* forwarding are adaptable to the network's condition. The forwarding plane has the ability to choose among multiple alternative paths in order to bypass the problematic interface. The rapid detection and recovery capability of the forwarding layer ensures that *NDN* routing does not require fast convergence to reflect network modifications. In addition, the scalability and stability of routing can be significantly improved by resolving network faults at the forwarding layer, so relieving the routing plane from temporary difficulties.

The concept of a *Nack* message was initially introduced by Yi *et* al. [43, 44]. In this concept, a node generates and sends a *Nack* message to the downstream node when it refuses an incoming *Interest*. Alongside sharing the same name as the incoming *Interest*, *Nack* has the capability to promptly alert about network issues by transmitting a code along with a descriptive message detailing their origin. The forwarding plane can promptly take local action to recover from network problems, without the need to wait for routing convergence [45]. The forwarding plane has the capability to prevent looping of *Interest* packets by assigning a nonce field (a random number) to each *Interest* packet. This nonce, combined with the *Interest* name, ensures that each *Interest* packet is uniquely identified. In addition, the equivalent *Interest* cannot form a loop either, as the *Data* packets follow the same paths but in the opposite direction. As a result, an *NDN* router is capable of forwarding an Interest over several interfaces without creating loops. The Forwarding Information Base (*FIB*) in the Named Data Networking (*NDN*) architecture includes a prioritized list of several interfaces for each entry, taking advantage of *NDN*'s built-in capability for multipath forwarding. The forwarding strategy module selects the interface(s) to forward an incoming *Interest* based on the forwarding information, routing information, and forwarding policies set by the operator [36].

2.3.7.3 Establishing the *FIB* table

In order to ensure that the material saved on the producer application can be easily accessed in the *NDN* (Named Data Networking) system, the producer application registers the prefix of the content with the local forwarder on the same host computer. Subsequently, the content producer notifies the availability of its content to remote *NDN* forwarders for the purpose of requesting and downloading. Methods such as manual configuration, dynamic name announcement protocols (such as *NLSR*), or opportunistic data discovery techniques (like the access strategy) can be used to announce the availability of material [36]. The trade-offs of each strategy differ based on aspects such as usability, implementation difficulty, and communication costs.

Jacobson and his colleagues introduced the Automatic Prefix Propagation (*APP*) protocol, as described in their work cited as Back69. This protocol is utilized on the producer computer to enable the local *NDN* forwarder to immediately transmit local prefix registrations to the distant forwarder of the gateway router. Propagation of local prefix registration will commence if two criteria are fulfilled: (1) the presence of an operational remote *NDN* gateway; and (2) possession of a private key and the related *NDN* certificates by the local forwarder that either match or encompass the locally registered namespace. Figure 2.10 illustrates the process. If there are numerous choices for the keys/namespace, the key with the shorter namespace is selected. The forwarder is able to consolidate several local registrations into a single remote operation, resulting in a reduction in communication and bookkeeping expenses. The Forwarding Daemon (*NFD*) periodically updates the registration following the first distribution of the prefix, as long as the prefix needs to be registered locally or there are *NDN* gateways set up.

2.3.8 *NDN* Forwarding Strategies

The majority of existing *NDN* forwarding techniques can be classified into two main categories: flooding-based strategies and single-route-based strategies. Flooding-based strategies involve the dissemination of packets to all neighboring nodes, while single-route-based strategies involve forwarding packets along a predetermined

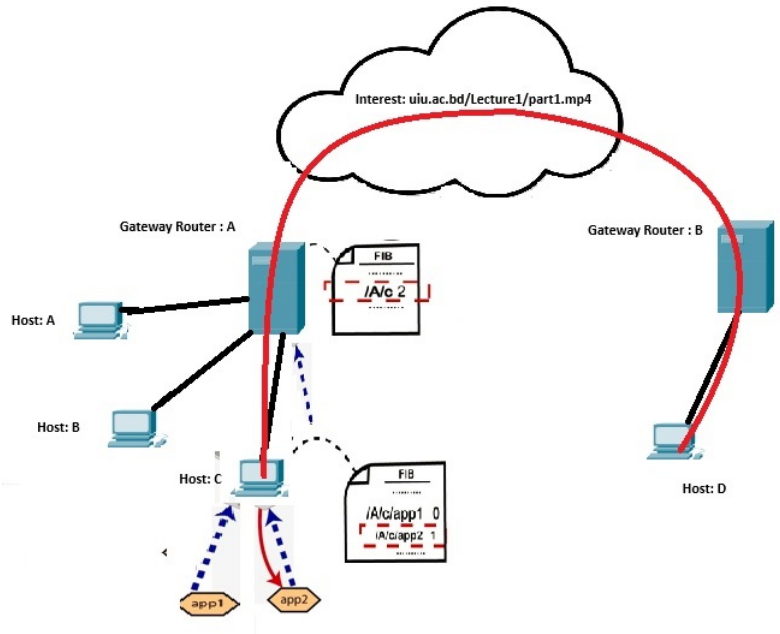


Figure 2.9: Advertising a producer application's *Data* to *NDN* forwarders, from [46]

path. These categories have been extensively studied and documented in various research papers, such as the ones cited: Kerrouche et al. (2016) for flooding-based strategies, and Kerrouche et al. (2016) and others for single-route-based strategies. The flooding-based tactics employ several routes to target many sources, and the information is sent back to the requester in the opposite direction [16, 47–54]. The guarantee of information retrieval is ensured by forwarding Interest to all accessible outgoing interfaces. Consequently, the person making the request gets more than one duplicate of a certain piece of content. However, except for the initial copy, the further copies are disregarded because they are deemed unneeded. The strategy is straightforward, but the excessive duplicates result in substantial transmission overhead, depleting valuable network bandwidth, depleting node resources (energy), causing network congestion, and diminishing network performance, especially in dense networks [55, 56].

2.4 Software-Defined Networking

The fact that *NDN* is a distributed network design presents a serious drawback that restricts the potential remedies to its scalability issues. However, a number of Software-Defined Networking (*SDN*) advancements that attempted to divide the control plane from the data plane were advantageous to the IP design. Next, a logically centralized component with a global network perspective implements the control plane. As a result, the controller is able to implement effective network management algorithms that address numerous of the previously described routing scalability issues thanks to this global view. To limit the size of the FIB, for example, only a portion of the most frequently used routes should be stored in the switches; additional routes can then be queried from the controller. Likewise, with the controller's assistance, alternate routing algorithms that don't depend on substantial *FIB*s in each switch—like source routing—can be implemented. We give some background information on *SDN* and several research ideas that combine *NDN* with SDN in the next section.

The modern Internet was intended to be a distributed system, with multiple distributed algorithms operating on each switch and router to provide packet forwarding. Though this distributed architecture offers a number of dependability benefits, network management becomes more difficult as a result. For example, anytime a new network behavior is required, a network operator must configure each switch separately, typically utilizing vendor-specific interfaces. Overall, this increasing complexity led to a slowdown in network layer innovation and made the network less able to adjust to new requirements. The switch from IPv4 to IPv6 is a good example; although IPv6 was designed some decades ago, the Internet has not yet fully incorporated it. In this context, SDN principles were introduced to provide more flexibility and creativity at the network layer.

The centralization of the network control plane is the fundamental concept of *SDN* techniques. The data plane and the control plane are the two fundamental layers of distributed network switches and routers. Fast packet forwarding logic, such as destination-based IP forwarding used in conventional IP routers, processes each packet according to a set of straightforward rules and is contained in the data plane. However, the various router algorithms that configure the data plane (such

as the various routing algorithms that populate the switch forwarding table) make up the control plane. Separating the control and data planes and centralizing the control plane into a single component—the controller—is the first action taken by *SDN* techniques. After that, the various switches are reduced to basic forwarding devices that are set up by the controller via the southbound interface, a defined API. Thus, enabling the control plane to be programmable improves network flexibility. As a result, the controller serves as a network operating system, giving various *SDN* applications that use network management techniques an overall, abstract picture of the network. Thus, updating an *SDN* application in the controller to incorporate a new network behavior becomes straightforward. Furthermore, the controller's global network view can help a number of algorithms function more effectively. For instance, *SDN* routers do not need to keep the graph of the entire network topology and then determine the shortest paths to every destination, in contrast to distributed OSPF routers. Rather, the controller learns the topology of the network, computes the best routes using an SDN routing application, and then updates the routing tables of every managed router inside the domain.

Fig. 2.10 shows an overview of a generic *SDN* architecture.

The southbound interface is a significant element in *SDN* designs. Indeed, it dictates the functionalities of the data plane, the architecture of the packet processing pipeline, and the methods by which it can be customized. The primary interface for southbound communication is OpenFlow. This statement describes both the logical structure of the packet processing pipeline and the network protocol that is employed to configure this pipeline. Each OpenFlow switch revolves around a customizable flow table. The latter refers to a match-action table that includes flow rules that specify a forwarding action for certain packets. To be more explicit, a flow rule is used to match a series of packets, known as a flow, based on the contents of packet header fields. The OpenFlow specification defines the range of potential forwarding operations and the header data that can be utilized for matching. Although the initial OpenFlow standard had limited support for header fields and forwarding actions, the newest version now enables complicated behaviors across several protocols, such as metering and flow table chaining.

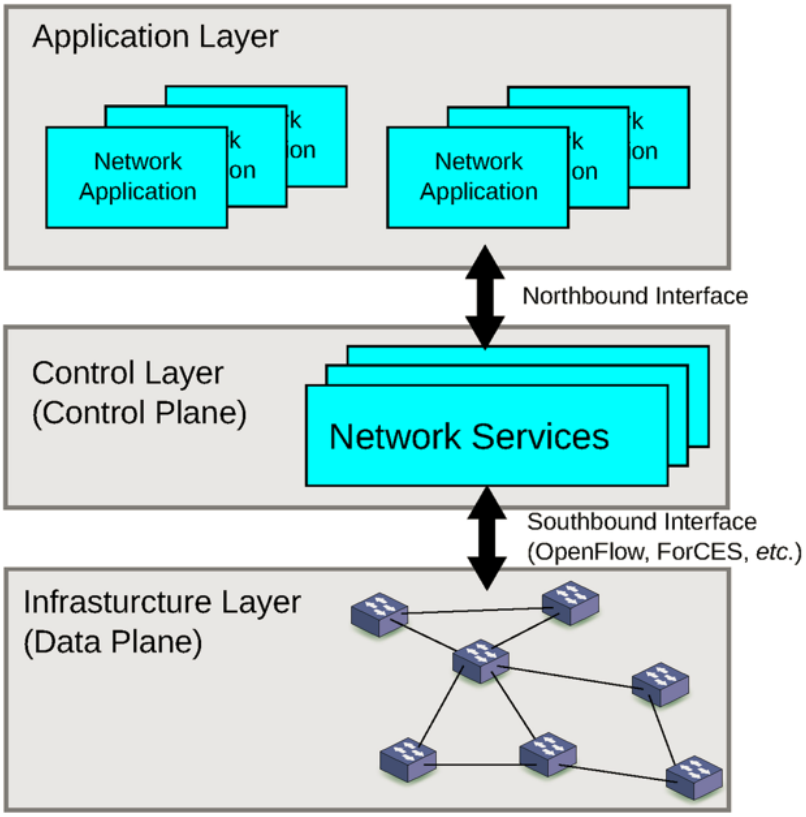


Figure 2.10: Main components in an *SDN* architecture, from [57]

Although OpenFlow is the primary southbound interface utilized in *SDN* architectures, there exists a wide range of controller network operating systems. These controller designs vary in their components, the abstract network view they offer to *SDN* applications, their ability to be deployed in a distributed manner, and the northbound interface used for programming. Nevertheless, a substantial classification of network operating systems comprises network hypervisors. Their primary objective is to partition the physical network into several virtual network perspectives that may be individually controlled by separate controllers. The introduction of SDN has resulted in the creation of a wide range of SDN applications that cater to different network management requirements such as traffic engineering, monitoring, and security.

2.4.1 SDN-based Approaches for NDN

Several research proposals aim at applying SDN concepts to NDN, especially for routing. Like in the current Internet, instead of having every node in the autonomous system recover the whole AS topology and content availabilities, an SDN controller can be used. In SDN-based approaches, routing and content-related information are logically centralized in a single controller that calculates the optimal routes and fills the routers' FIBs. SDAR [58] and SRSC [59] are two examples with SDN-based routing.

These SDN techniques mostly depend on creating a protocol similar to OpenFlow, utilizing NDN packets, and establishing a naming system for communication between the controller and the routers. Initially, each node would communicate with the SDN controller by providing information about its neighboring nodes and the data it contains. The SDN controller will choose the most efficient paths by implementing algorithms commonly found in name-based link-state routing protocols, such as Dijkstra's algorithm or the heuristics described in [60]. Subsequently, it will transmit these routes to the routers with the purpose of populating their Forwarding Information Bases (FIBs). Alternative methods employ Software-Defined Networking (*SDN*) to enhance the efficiency of caching Data Packets (Dpkts) in the Content Servers (CSs).The papers referenced are [61–63]. Nevertheless, these ideas primarily prioritize the cache management algorithm rather than the *SDN* architecture itself.

2.4.2 Singe-State *Q*-learning

Reinforcement learning (*RL*) is a highly effective approach for teaching agents to make sequential choices in intricate situations. The Q-learning algorithm is a fundamental method for learning optimum policies in the field of *RL*. Nevertheless, when the complexity of environments increases, conventional *RL* methods may encounter difficulties in terms of scalability and computational efficiency. Single-state Q learning (SSQL) presents a viable solution to address these challenges by streamlining the learning process and minimizing the computing burden.

SSQL builds upon the principles of Q-learning, aiming to streamline the decision-making process by aggregating Q-values at the state level. Unlike tra-

ditional Q-learning, which maintains Q-values for each state-action pair, SSQL condenses this information into a single Q-value per state. This aggregation simplifies the learning process, reducing the computational complexity associated with updating and storing Q-values in large state spaces. By focusing on the expected cumulative reward from each state, SSQL provides a more efficient framework for learning optimal policies.

According to [64], the SSQL model is defined as a Markov decision process, where an agent is installed in an environment and equipped with a set of distinct environment states $S = \{s_1, s_2, s_3, ...s_m\}$ and a set of actions $A = \{a_1, a_2, a_3, ...a_n\}$. Fig. 2.11 depicts the Q-learning model.

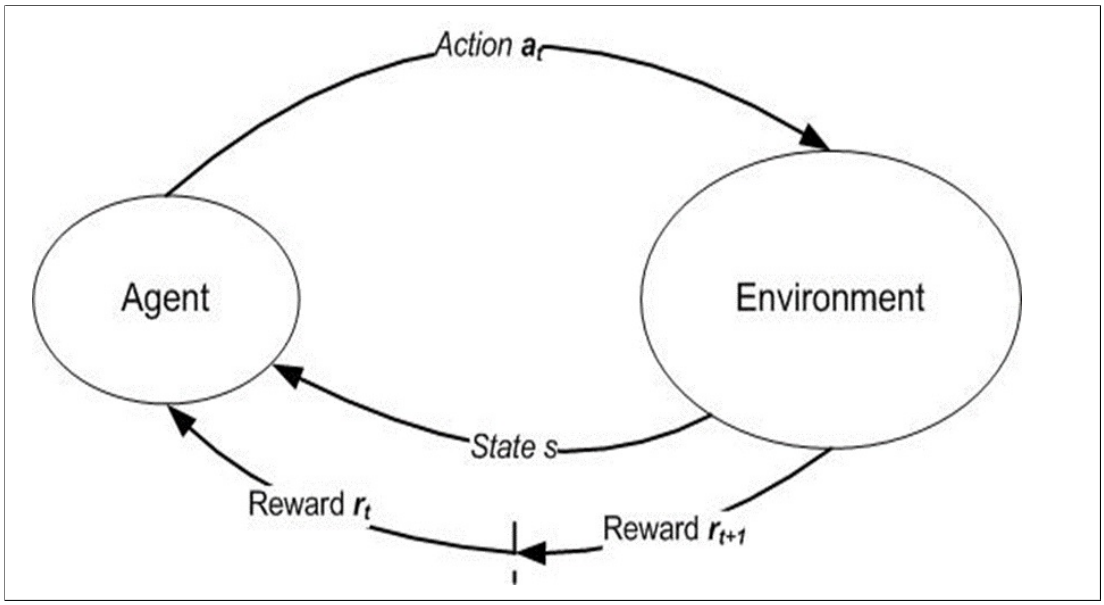


Figure 2.11: Standard Q-learning model

At any discrete time t, the agent perceives the environment state $s_t \in S$ and selects an action $a_t \in A$ based on the s_t, By performing a_t, the environment state changes or switches to a new or another state s_{t+1}, and the agent receives a reward $r_t = r(a_t s_t)$. Thus, the learning model typically consists of the following elements [65]:

1. S: a finite set of environment states

2. A: a finite set of actions

3. r: $S \times A \longrightarrow R$: a reward function

4. P: $S \times A \longrightarrow P(S)$: a state transition function

The objective is to find an optimized policy ϕ^* and maximize its accumulated reward. The optimized value function $\Phi^*(s)$ under ϕ^* is defined as [65]:

$$\Phi^*(s) = \max_a (r(s,a) + \gamma \sum_{s' \in S} P\Big(s'|s,a\Big)\Phi^*(s')) \tag{2.1}$$

where $P\Big(s'|s,a\Big)$ is the transition probability from s to s' when a is the chosen action, γ is the learning rate ($0 \leq \gamma \leq 1$), $r(s,a)$ is the cumulative reward of state s, and $P\Big(s'|s,a\Big)\Phi^*(s')$ is the expected feedback from the successor state s'. Once the $\Phi^*(s)$ is calculated, we can derive the optimized policy as [65]:

$$\phi^* = \operatorname*{argmax}_a (r(s,a) + \gamma \sum_{s' \in S} P\Big(s'|s,a\Big)\Phi^*(s')) \tag{2.2}$$

SSQL is used to compute $\Phi^*(s)$ when the installed agent has an information shortage regarding state transition probability $P\Big(s'|s,a\Big)$ and the estimated reward $r(s,a)$. The optimized value computed using Q-learning is the Q-value which allows an agent to select the optimal action from the action set A. According to [64], for a policy ϕ, the Q-value of action $a \in A$ in a particular system state $s \in S$ can recursively be obtained as:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha\Big(r(s,a) + \gamma \max_a Q(s'a) - Q(s,a)\Big) \tag{2.3}$$

Mapping state-action pair is essential to apply SSQL techniques to any system. However, the proposed system model considers the content source as the agent that selects between actions *Respond* or *Mute*. Since the agent's only objective is to suppress redundant or sub-optimal responses, which is achieved through the two actions, the state-action pair is less of an issue here. We propose a single-state Q-learning model where an agent finds the usefulness of selecting an action in the next round to effectively model an agent's behavior.

The authors in [66] originally proposed the single-state Q-learning to solve stateless computer science games [65]. The authors reform the typical Q-learning

algorithm to such that the Q-values of actions effectively are estimates of the usefulness of the actions in the next episode of the learning process. An agent computes the Q-value for each available action and selects the action for the next round entirely based on its Q-value. The Q-value of the chosen action is updated by receiving the immediate reward. The function to update the Q-value is defined as:

$$Q(a) \longleftarrow Q(a) + \gamma(r(a) - Q(a)) \tag{2.4}$$

Where γ is the learning rate ($0 \leq \gamma \leq 1$) and $r(a)$ is the immediate reward of choosing action a.

SSQL finds applications across various domains where efficient decision-making is paramount. In robotics, SSQL enables autonomous agents to navigate complex environments with reduced computational overhead, making it suitable for real-time control systems. In finance, SSQL can be employed to optimize trading strategies by learning from historical market data and adapting to changing market conditions. Additionally, SSQL holds promise in healthcare, where it can assist in personalized treatment planning and resource allocation. By simplifying the learning process, SSQL extends the reach of reinforcement learning to domains with stringent computational constraints.

Several advantages distinguish SSQL from traditional Q-learning approaches. Firstly, SSQL reduces computational complexity by aggregating Q-values at the state level, making it well-suited for environments with large state spaces. This reduction in complexity leads to faster convergence and more efficient decision-making. Secondly, SSQL requires less memory overhead, as it stores only one Q-value per state rather than maintaining a Q-table for each state-action pair. This memory efficiency enables SSQL to scale to environments with high-dimensional state spaces. Thirdly, SSQL offers enhanced interpretability, as the Q-value for each state represents the expected cumulative reward from that state onwards. This interpretability facilitates a deeper understanding of the agent's decision-making process and aids in troubleshooting and optimization.

The evolution of Single-State Q learning represents a significant advancement in the field of reinforcement learning, offering a pragmatic solution to the challenges of scalability and computational efficiency. As research in this area continues to

evolve, future directions may explore extensions and refinements of SSQL to address its limitations and expand its applicability to diverse domains. Additionally, interdisciplinary collaborations between researchers in reinforcement learning, artificial intelligence, and domain-specific fields can unlock new opportunities for leveraging SSQL in real-world applications. By harnessing the principles of simplicity and efficiency, Single-State Q learning paves the way for intelligent decision-making systems capable of navigating complex and dynamic environments, driving innovation and efficiency in various domains.

2.5 Related Work

The majority of the current *NDN* forwarding strategies use single-path strategies such as best-route [8], *ASF* [67], *SAF* [68], *INFORM* [69], and others, which consumes precious resources such as network bandwidth, CPU cycles, etc., [70]. The strategies gather prior (forwarding) information to find a better path, either issuing periodic or event-triggered control messages or probing the alternative interfaces periodically or opportunistically. The interval of such control messages or probing rate limits the applicability of the strategies when large-scale *NDN* deployment is assumed. *SDN-NDN* integration can scale well to address such difficult scenarios.

A common practice in *SDN*-based *NDN* paradigm [16, 17, 19, 71–76] is to maintain tables for the underlying cache state and network topology at the controller. The controller utilizes the table information to compute the best path to a source, which is sent on-demand to a forwarder or installed proactively in each forwarder. The tables are updated every time a content is admitted or replaced. Such proposals experience computation burden at the controller and enormous control packets exchange between the controller and forwarders, especially in rapid cache ingress and egress moments.

SDN-driven multipath forwarding strategy for *NDN* is proposed in [16] that utilizes the global network information at the *SDN* brain to distribute *Interest*s efficiently. The work successfully reduces latency for content retrieval and avoids congestion on a single path. Increased communication overhead is apparent here as the scheme retrieves content from multiple sources. In contrast, the

previously proposed *SDN-SRP* suppresses redundant responses (partially) to reduce transmission overhead; however, the *SDN* controller is actively involved in the data communication process to suppress the sub-optimal responses. The suppression requires trading significant control packets between the controller and the lower layer nodes. In the present work, we optimize the quantity of control messages exchanged to eliminate redundant replies.

In [6, 19], the network is divided into domains or clusters, and the *NDN* switches undertake bloom filter-based *FIB*, *PIT*, and *CS* to improve their name lookup performance. However, the proposed work fails to achieve the objective cost-efficiently as it demands the *SDN* controller and *NDN* switches to implement three and four additional complex data structures or tables, respectively. Here, when a forwarder is unable to fetch the desired object, the *Interest* is forwarded to the *SDN* controller to initiate a search and retrieve the content from another domain. In that case, the content is retrieved through the control channel, limiting the scheme's practicality.

Aubry *et* al. [17, 77] proposed *SRSC* that avoided sending path information to the *NDN* switches; however, only at the cost of a new type of control *Data* packet containing the entire path (list of nodes) to the nearest source. The size of such control packets can become unmanageable in an actual network topology. *SRSC* exhibits a better cache-hit rate and fewer control messages assuming infinite *FIB* memory size. In a memory-constrained environment, the consumers may experience a longer content download time if the *SDN* controller is rapidly consulted for on-demand path information. Ascigil et al. [78] proposed a slightly different strategy where a local service of each domain collects and maintains path information of that domain. The *SDN* controller in *CRoS-NDN* proposal [71, 79] implements a similar approach for routing service. However, the *Cros-NDN* has the same problem as *SRSC* as it engages the controller (requesting on-demand routes) to guide an *Interest* to the nearest content source.

Jung-Hwan et al. [80] urged that variable-length names used in *NDN* for longest prefix matching should be transformed to fixed-size match fields in flow table entries used in the *SDN* paradigm. The authors addressed this non-trivial problem utilizing a dual naming scheme where the alphanumeric and variable content name is used for routing naming space, and an IPv6 address is used

for *SDN* forwarding. Like [17, 71], a forwarder instantiates a query to the *SDN* controller, asking for the best path to the source when no information is available in the flow table, hence increasing the content retrieval delay.

Son et al. [72] introduced cluster-based distributed *SDN* controllers and packet structure modification to propose a novel forwarding scheme. Although the proposed scheme uses a single path to reach the content source, like [17, 71, 80], it invokes the controller while searching for the desired content.

Mahmood *et* al. [81] proposed an unique *SDN-NDN* integration which completely deviates from the common practice. The authors avoid cache state and network topology maintenance at the controller, resulting in negligible control messages being exchanged for data communication. Here, *Interest* and *Data* packets are commuted in overlay UDP packets. The proposal does not consider transmission overhead reduction, and its performance is not contrasted against any relevant proposals.

Amadeo et al. [82] identified a new horizon to advance the state-of-the-art *SDN-NDN* integration by examining the impact of controller decisions on *FIB* management. *NDN*-Fab's proposed in [83] provides an *SDN-NDN* integration of path-based (location-based) and content-centric networking models. Salsano et al. [84] extended the OpenFlow [85] protocol for *SDN*-based *CONET*.

Apart from the *SDN* and *NDN* integration proposals for wired networks, a considerable research effort has been devoted to *SDN-NDN* integration for wireless networks, and others, such as MANET [86], VANET [87–89], WSN (IoT) [90, 91], etc. These proposals need to address two main challenges: mobility and broadcast storm.

Numerous deep and machine-learning approaches for legacy *NDN Interest* forwarding have been reported in the literature, including exploring Q-learning. However, the application of these methods incurs significant computational overhead, more overhead packets or an increase in the size of standard *Interest* and *Data* packets [69, 92–99], due to the utilization of deep and machine learning techniques. *DTABB* and $SRAB^3$ proposed in [21, 22] employ straightforward reinforcement learning *RL* algorithms to optimize *Interest* forwarding and *Data* delivery at the cost of considerable control (acknowledgement) packets. The content sources use the acknowledgements to learn the optimal response type.

The volume of control packets contributes significant network overhead, especially in large-scale networks. In the proposed scheme, the sub-optimal responses are suppressed using Q-learning; however, no acknowledgement packets are utilized for the learning purpose in the content sources, making it different from the $SRAB^3$ and *DTABB*. Moreover, the learning task is offloaded to the *SDN* controller from the *NDN* routers. The Q-learning task installed in the centralized controller requires routers to exchange control packets with the controller.

2.6 Research Gap

In $SRAB^3$, the customer must return an acknowledgment to each content source available in the network to complete the learning process. The acknowledgment packets consume network bandwidth. Moreover, the content sources incur significant computational overhead since the learning is executed within the sources. In *SDN-SRP*, a substantial volume of control messages is necessary to suppress the suboptimal replies, and the sources must remain connected to the controller at all times; otherwise, the scheme reverts to the default scheme. However, the proposed solution spares the client from sending any acknowledgment to the sources as the learning takes place in the controller. The controller communicates with the sources using only a few out-of-band control messages. Besides, the scoped flooding technique in SRAB3 is not adaptive as it utilizes cache hits and misses information instead of content sources' distances. While SDN-SRP floods the *Interest* request in the entire network, leading to an increased communication overhead. In contrast, SDN-Q controls the flooding ball based on the content sources' distance information.

2.7 Summary

This chapter provides a concise overview of significant *ICN* architectures, including *PURSUIT*, *DONA*, *NetInf*, novel innovation *NDN* and *CCN*. All of the proposed designs employ a content-centric design paradigm, as opposed to a host-centric one, but they may utilize alternative methodologies. The study focuses on enhancing the *SDN*-based *NDN* concept and provides a detailed analysis of its architectural

components and guiding concepts. The following chapter examines the *SDN-Q* forwarding techniques.

Chapter 3

SDN-Q Forwarding Strategy

This chapter commences with the *SDN*-based *NDN* picture for large-scale *NDN* solution, followed by a formal problem statement that articulates the reaction of a content source to an incoming *Interest* request as a standard Q-learning model.

3.1 *SDN*-based *NDN* Architecture

The *SDN*-based *NDN* architecture dissociates a router's control plane from its data plane to transform the typical flat network into a hierarchical two-layer network architecture. A remote controller or server centralizes the control planes of the routers, referred to as the *SDN* controller. Fig. 3.1 depicts the *SDN*-based *NDN* architecture. The proposed tables or data structures, such as *Action* and Control-Q are described in section 3.4. The controller is considered the upper layer of the two-layer architecture. It is the "brain" of the model and is responsible for managing or administering underlying network resources and nodes' cache states. The decoupling transforms each *NDN* router into merely a forwarding device or switch. The lower layer consists of forwarding devices with data planes only that obey the programmed rules. Each forwarding node uses a direct link or control channel to connect to the controller. We assume that the lower layer forwarding nodes form a connected multi-hop network and their data planes support chunk-level operations. The terms node, device, and switch in this study have the same meaning. The network topology at the lower layer is represented as an undirected graph $G = (V, E)$, where V is the set of nodes

$v_1, v_2, ...v_i..., v_n$ and $E \subseteq (V \times V)$ is the set of edges/links between the nodes. An original provider (server) of a content and any $v_j \in V$ $(1 \leq j \leq n)$ retaining a replica of the original content are considered sources of the content. The latter is called a cacher, custodian, or holder and the set of custodians is represented as $H \subseteq V$. A $h \in H$ holding a copy of content p is represented as h^p and it may satisfy an incoming *Interest* based on the strategy's policy from its *CS*. Together, the server and the custodians are considered content sources. A consumer issues an *Interest* request for the desired content, forwarded in a hop-by-hop fashion until it hits a source. The corresponding *Data* packet travels downstream toward the consumer. A downstream node may retain a copy of the content according to the installed caching scheme. For simplicity, all items are assumed to be of equal size. However, the result of this study can be generalized to the cases where item size differs. Moreover, the lower layer forwarding planes support adaptive forwarding by ranking and selecting the interfaces for advancing those input *Interest*s that cannot be satisfied locally. The model uses southboundAPIs, for example, OpenFlow [85] for inter-communication between the two planes.

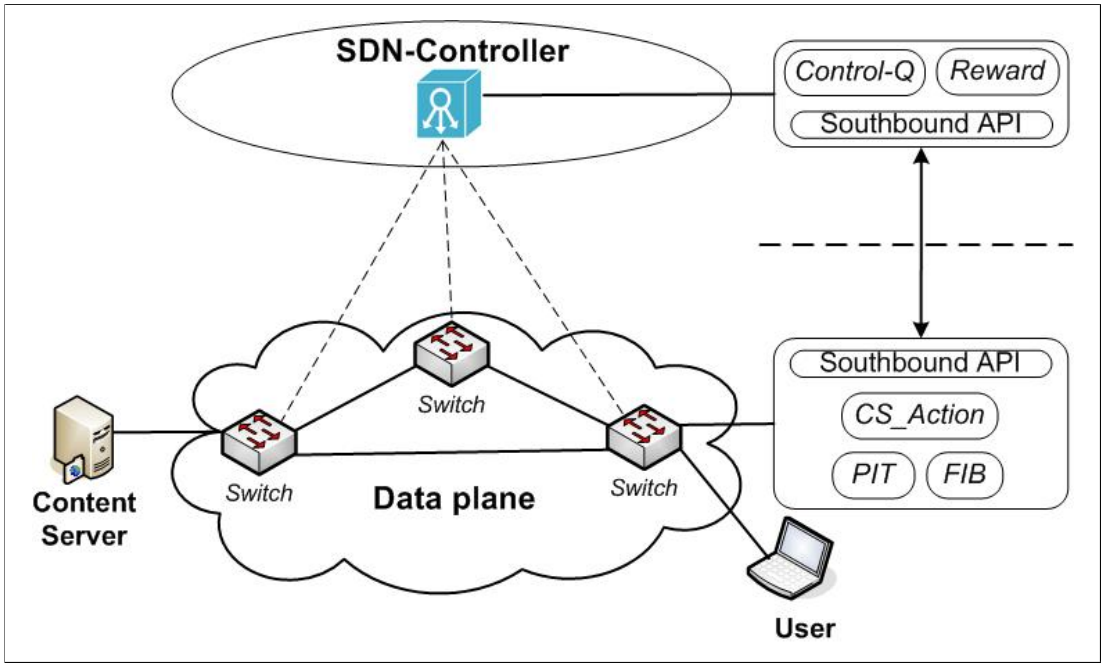


Figure 3.1: *SDN*-based *NDN* architecture

Since *Interest* flooding is a negligible contributor to the traffic load performance bottleneck, the controller mainly endeavours to optimize the content retrieval phase by suppressing redundant *Data* replies. The proposed mechanism devises a novel scoped flooding to direct *Interest*s to a limited number of content sources and utilizes *RL* to fetch the required *Data* primarily from the optimal source and suppress the suboptimal responses. Like *SDN-SRP* [21], and $SRAB^3$ [22], it eliminates the need for meta-data packets, for instance, *Offer* packets. Moreover, no control packets are communicated for *RL* among the lower-layer data plane, unlike $SRAB^3$, which uses acknowledgement packets for the said purpose. The proposed solution reduces the prolonged delay problem of $SRAB^3$ and the additional traffic load problem of *SDN-SRP*. Furthermore, the current scheme minimizes the volume of control messages traded between the upper and lower layers to ensure reliable and efficient content download.

We modify the default *NDN Interest* and *Data* packets for communication between the forwarding nodes and the *SDN* brain. The OpenFlow enables the controller to obtain and retain a complete and consistent view of the underlying network states, especially the cached contents at the nodes, and updates the forwarding planes directly as needed. Since we urge the content delivery phase is chiefly accountable for performance bottlenecks, for example, higher traffic load, we push the controller to optimize the sub-optimal responses by retrieving content mainly from the best source. The lower-layer forwarding nodes optimize the *Interest* forwarding, distinguishing the proposal from *SDN-SRP* that performs *Interest* flooding in the entire lower-layer network. Again, the controller executes the learning job to stop the sub-optimal redundant replies, differing it from $SRAB^3$.

Like, *SDN-SRP*, the *SDN* controller need not calculate any route for *Interest* forwarding needed by the data planes. However, the Q-learning task of all *NDN* nodes is now centralized in the controller, making it different from *SDN-SRP*. The centralized learning makes it unlike $SRAB^3$ too. The messages regarding learning use the control channel between the controller and the forwarding nodes, thereby eliminating the need for in-band control packets. The distinctive features permit *SDN-Q* to design only a few data structures and exchange fewer control messages to retrieve contents.

3.2 Justification for Q-learning

Single-state Q-learning is known for its simplicity and efficiency in decision-making processes. As the controller is burdened with executing the learning tasks of each content source, it needs a simple learning algorithm for faster decision-making. Single-state Q-learning is more efficient regarding communication overhead, as the controller renders the actions only, not the states. Since each node retains the action required for an incoming *Interest*, the Q-learning, in turn, can significantly reduce latency in content discovery and delivery. As the underlying caching state of the topology is dynamic and non-trivial to assume, the experience-based approach of Q-learning spares the agent (controller) from requiring prior knowledge about the cache states, for example, the controller can learn the underlying caching state as contents are admitted or evicted.

3.3 Problem Definition

The *CS* of each *NDN* node acts as a temporary custodian for a content and is considered a learning agent that lives in a single state and entertains each arriving *Interest* in precisely one of the two specified ways. A custodian assumes a discrete-time model for the hitting *Interest*s. For each incoming *Interest*, analogously, at any round, the custodian selects between reciprocating with one of the two actions, namely, *Respond* or Mute. The *Respond* action corresponds to the typical case where a node must reply with the matching *Data*. On the contrary, the *Mute* action resembles that the content holder prefers to ignore the *Interest* or restrain from sending the *Data* back to the requester. The node selects the former action if it is the optimal source for responding with the *Data*, while it selects the latter to suppress any sub-optimal *Data* reply. A *CS* can dare to stay *Mute* as the proposed strategy directs an *Interest* to multiple sources and guarantees item download at least from one of those discovered sources.

An agent's objective is to learn the appropriate action with the highest estimated Q-value by experimenting, mainly using trial-and-error discovery and delayed reward [64]. Therefore, a sub-optimal source may answer with the *Respond* action alongside the optimal source, causing redundant copies to arrive at

the consumer. On the other hand, an optimal source may stay *Mute* causing a sub-optimal reply to serve the purpose. However, only the first *Data* arriving at the consumer is meaningful; the consumer ignores any subsequent *Data* packets. Every time a requesting *Interest* hits a content holder, it reciprocates with one of the two actions and sends a message to the *SDN* controller, which the controller uses to determine the reward received for playing the action. The reward updates the estimated Q-values, which the holder uses to determine the appropriate action to entertain subsequent *Interest*s. The Q-values get converged eventually so that the agent can react with optimal action.

The requesting *Interest*s (I) are sequentially presented to the custodian. The custodian must react to each striking *Interest* i ($i \in I$) based on the Q-values of the actions. The action with the higher Q-value is picked. *Respond* and *Mute* are the two actions (a) the agent selects from such that $a \in A = \{R, M\}$, where R and M represents actions *Respond* and Mute, respectively. The action selected to entertain an incoming i at time t, is denoted by a_t, and when executed, the agent receives a reward $r_t = b$ for that action. An i may hit multiple custodians, and the custodians must react to i with R or M. After responding, each custodian sends a message to the controller where the reward for the corresponding action taken by a custodian is determined. In this study, the reward for any action is binary such that $b \in \{0, 1\}$. After that, the controller updates the Q-value of the selected action for the custodian as:

$$Q(a) \longleftarrow Q(a) + \gamma(b - Q(a)) \tag{3.1}$$

where $a \in \{R, M\}$ and $b \in \{0, 1\}$. To minimize the number of control packets traded between the controller and lower-layer switches and to make the lower-layer nodes light-loaded, the learning task of an agent is offloaded to the controller. The controller, on behalf of an agent, updates the Q-values after an action is taken and decides the action to entertain the next i. Next, the controller renders the decision to the agent. Any agent only stores the action (R or M) in its *Action* table to entertain the next i.

3.4 *SDN-Q* Methodology

First, this section introduces various data structures and messages used by the *SDN* controller and the *NDN* nodes for learning purposes. Later, it describes the principal concepts and algorithms used to commute *Interest*s and *Data* to retrieve a content. The proposed strategy utilizes scoped flooding to restrict *Interest* dissemination to a few content sources. The proposed design suppresses the sub-optimal sources using Q-learning to reduce the traffic load.

3.4.1 Data Structures, Control Messages, and *NDN* Packet Extensions

3.4.1.1 Switch Tables

An *NDN* forwarding device comprises the following data structures:

- *CS_Action*: This table is almost the same as the content store installed in typical *NDN* architecture, and table admission and eviction are controlled locally by the data plane. Each entry of the table is extended by adding a field *Action* to denote the action to respond with whenever a corresponding *Interest* arrives. The *Action* field (D_t^P) is updated whenever the controller instructs the h^p via an *Action(p)* message. The *CS_Action* table is given in Table 3.1.

Table 3.1: *CS_Action* Table

Prefix(p)	***Data***	***Action*** (D_t^p)
audio/one	$\cdots$	R
audio/two	$\cdots$	M
audio/three	$\cdots$	R

- *PIT*: Similar to the *PIT* installed in the standard *NDN* and populated and updated locally by the forwarding switch.

- *FIB*: The *FIB* holds the forwarding information, such as each entry in the table corresponding to an interface to reach a content source. The

FIB is capable of ranking its interfaces locally and uses the ranking to control *Interest* flooding. The structure also contains an entry regarding the interface to communicate with the controller.

3.4.1.2 Controller Tables

The controller contains two novel tables named Control-Q and *Reward*.

- *Control-Q*: The table contains cache states of lower layer switches linked to the controller. Each entry in this table refers to a content prefix (p), the node name (ID), Q-value of action R (Q-R), Q-value of action M (Q-M), and selected action. A table entry is created when a node caches a new item and is updated when an i hits a custodian. Fig. 3.2 portrays the *Control-Q* table.

Table 3.2: *Control-Q* Table

Prefix(p)	***Node (ID)***	***Q-R***	***Q-M***	***Present Action*** (D_t^p)
audio/one	1	0.7	0.1	R
audio/one	2	0.1	0.8	M
audio/two	1	0.6	0.0	R
audio/three	3	0.3	0.1	R
audio/three	4	0.1	0.4	M

- *Reward*: Each entry in this table contains a content prefix, an ID, and a status (*isSatisfied*). An entry is inserted, and the status is initialized when the first custodian is hit. Fig. 3.3 portrays the *Reward* table.

Table 3.3: *Reward* Table

Prefix(p)	***InterestID***	***IsSatisfied***
audio/one	100	*False*
audio/two	105	*True*
audio/three	110	*False*

3.4.1.3 Control Messages

The control messages are traded between the controller and the switches to update the *Control-Q* table, determine the binary reward, and select the appropriate action to entertain an incoming *Interest*. We modify the default *Interest* and *Data* packets to design the control messages. Table 3.4 lists the messages and their purposes.

Table 3.4: *Control Messages*

Sl. No.	***Message***	***Type***	***Purpose***	***Sender***	***Receiver***	***Format***
1	*C-Adv*	*Interest*	Controller advertisement	Controller	$v_j \in V$	*<ControllerID>*
2	*NodeReg*	*Data*	Node registration	$v_j \in V$	Controller	*<NodeID>*
3	*Insert(p)*	*Interest*	Update *Control-Q* on admission of p in a h	A h^p	Controller	*<NodeID>*
4	*Reward(p)*	*Interest*	Update *Control-Q* upon request served for p	A h^p	Controller	*<RequestID, NodeID>*
5	*Action(p)*	*Data*	Action suggestion for p to a h^p	Controller	A h^p	*<NodeID, a>*

3.4.1.4 Header Extension

SDN-Q extends the default *Interest* packet header to control the scope flooding and suppress redundant replies. Three new fields, named *RandomID*, *Ray*, and *Radius*, are added to the default *Interest* header. The first field assigns a random unique *ID* to each *Interest* a consumer issues. The controller uses the field to determine the reward for performing an action. The second field denotes the instance of the *Interest* permitted to forward outside the scope. The *Radius* field controls the scope (or the number of hops) of *Interest* flooding, analogous to the TCP/IP's time to live *TTL* value. The *RandomID* field is never changed, while the rest of the extensions are initialized by a consumer and updated by the *NDN* switches as the *Interest* moves frontward.

SDN-Q extends the default *Data* packet header, by adding a field *Hops*, to find the hop distance between a content source and the user. The source initializes $Hops = 0$, which the downstream switches update as $Hops = Hops + 1$. However, if a switch decides to cache the *Data* packet, it reinitializes $Hops = 0$.

3.4.2 Content Retrieval Strategy

3.4.2.1 Node Registration

The network starts with a lower layer node registering itself with the *SDN* controller. Initially, the switches are ignorant of the controller's presence. The controller literates its presence to the lower layer by advertising a *C-Adv* message. Upon receipt of the advertisement, a node binds the incoming port or interface in its *FIB* to communicate with the controller. Then, it sends back a *NodeReg* message to register itself to the controller via that interface. A node ignores the message if already been enlisted by the controller.

3.4.2.2 *Control-Q* Update

Every time a h^P inserts P to its *CS*, it explicitly informs the controller. This is obligatory for the controller to obtain a complete view of the cache states to suppress the suboptimal responses. A h^P informs the insertion by sending an *Insert(p)* message where the *NodeID* identifies the h^P and p is the name prefix of P. The controller uses the cache insertion messages to update the *Control-Q* table.

In addition, A h_P sends a $Reward(p)$ message to the controller whenever a i_P strikes where the *RequestID* identifies the particular *Interest*, *NodeID* identifies the h_P, and *isRay* identifies whether it is the i_P^Y. The controller uses the *Reward* message to determine the binary reward for performing an action and update the *Control-Q*.

3.4.2.3 Example Scenarios

Figs. 3.2 and 3.3 refer to *Interest* dissemination and content retrieval in *SDN-Q*. A consumer C transmits an *Interest* i_P to fetch item P disseminated to a certain region (green-dotted arrow) in the lower layer network, initiating a contour around C. A special trail of i_P (brown-dotted arrow), called *Ray* (denoted as i_P^Y for P), is allowed to surpass the ring until it hits a h^P. Any h_P stroke by i_P^Y must respond with action R, thus sending back P to C. The sources (h_ps and any server) within the region react to i_P with the action as learned according to the Q-learning.

Though the sources may decide between actions R and M, the presence of *Ray* ensures content retrieval via i_P.

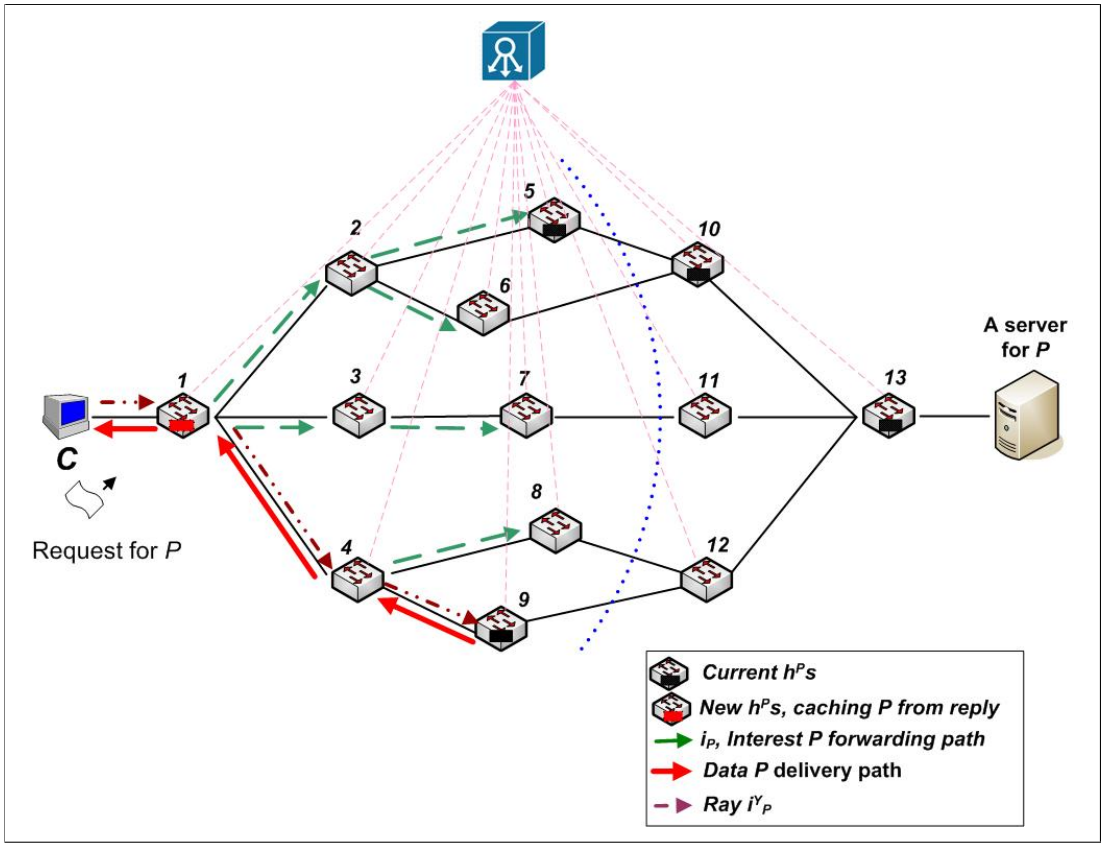


Figure 3.2: Optimizing Content Retrieval in *SDN-Q*- an optimal scenario

Fig. 3.2 corresponds to the scenario when the *Ray* strikes an optimal custodian inside the contour. Fig. 3.3 resembles the case when the *Ray* hits a sub-optimal source (the server) and an optimal holder within the ring responds with *Data* (red-dotted arrow).

3.4.2.4 Reward Determination and Action Selection

A custodian stores the action to perform for each content it caches. In any h^P, the Q-learning decision to entertain an i_P for content P at time t is represented as D_t^p, and hence, $D_t^p = R$ or $D_t^p = M$. D_t^p is stored in the *CS_Action* table in a h^P and also in the *Control-Q* table in the controller. Since the learning task

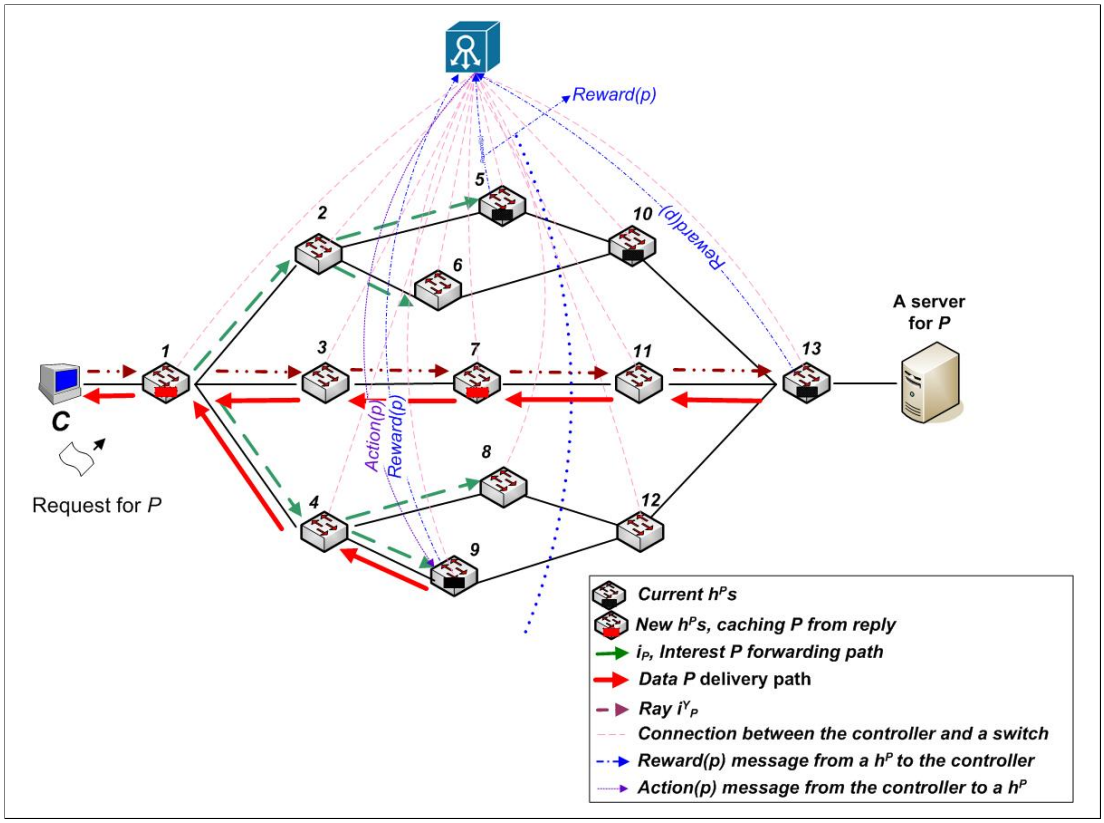


Figure 3.3: Optimizing Content Retrieval in *SDN-Q* - a sub-optimal scenario

is offloaded to the controller, the controller decides, stores, and communicates the D_t^p. A h^P only needs to store the value of D_t^p. If $D_t^p = M$, a h^P remains *Mute* when an i_P is encountered; however, the h^P must respond with *Data* if the i_P is i_P^Y, despite the stored action being *Mute*. The h^P sends the information of the striking i_P to the controller using a *Reward(p)* message. In contrast, a h^P responds with the *Data* (action R) when an i_P hits if $D_t^p = R$. This time too the h^P using a *Reward(p)* message sends the i_P's information to the upper layer. A *Reward(p)* message contains the *RequestID* of the i_P, the *NodeID* of the h^P, and *isRay* value of the i_P.

D_t^p for p in each h^P is initialized to M, mainly to minimize the traffic load. A consumer initializes the *RequestID*, *Ray*, and *Radius* fields while issuing an i_P. *RequestID* is a randomly generated number identifying the i_P. *Ray* field is initialized to 1. These two fields are used to determine the reward for performing

an action. An i_P may strike multiple h^Ps, thus, the controller may receive multiple *Reward(p)* messages for the i_P. When a controller receives the first *Reward(p)* message for an i_P from a h^P, it inserts an entry for (p) and h^P in its *Control-Q* table with $Q_R^p = 0.0$, $Q_M^p = 0.0$, and $D_t^p = M$. The controller also inserts an entry for p for the first *Reward(p)* message in its *Reward* table. The *Reward* table entry is populated using the fields of *Reward(p)* message such as $InterestID = ID$ of i_P, $CustodianID = ID$ of h^P, and $Ray = isRay$. The lifetime of an entry in this table is similar to an *Interest*'s lifetime. We assume an identical lifetime for each *Interest*, and the controller is pre-installed with the knowledge. The *Reward* table entry for the first *Reward(p)* message is used to determine the reward for the action taken by each h^P stroke by the i_P. For instance, when a controller receives a *Reward(p)* message for prefix "audio/one" from a custodian with $RequestID = 100$ and $isRay = false$, it inserts an entry in the *Reward* table as $p = audio/one$, $InterestID = 100$, $isSatisfied = false$. Next, it locates the D_t^p for the corresponding h^P in the *Control-Q* table. D_t^p can be R or M. If $D_t^p = M$ for the h^P (from *Control-Q* table), the reward (penalty) for the $D_t^p = M$ is $b_t^p = 0$. This corresponds to suppressing the i_P by the h^P was a sub-optimal action as it was the first h^P hit by the i_P. Thus, for the h^P, responding with *Mute* last time was unsuccessful. The controller sets $isSatisfied = false$ for the corresponding entry in the *Reward* table and updates $Q_M = Q_M + \gamma(0.0 - Q_M)$ in the *Control-Q* table. After updating Q_M, if $Q_R \geq Q_M$ update $D_t^p = R$ and send an *Action(p)* message to the h^P to update or synchronize its $D_t^p = R$. On the contrary, if $D_t^p = R$, then $b_t^p = 1$, corresponding to p served by an optimal h^P. This action corresponds to a successful action; the controller marks $isSatisfied = true$ and updates the Q_R value. However, if $isRay = true$, the controller only marks the corresponding Reward table entry $isSatisfied = true$ as the h^P must send back the corresponding *Data* packet when stroke by the *Ray*. No entry is inserted for the same *Reward(p)* messages for the same i_P. The successive *Reward(p)* messages received at the controller with the same *RequestID* but from different h^Ps are used to determine the reward received by those h^Ps.

For the second and or any later *Reward(p)* message received from a h^P, the controller queries the corresponding *isSatisfied* field (of the *Reward* table) and D_t^p (of the *Control-Q* table). We need to determine the rewards when $isRay = false$.

If $isSatisfied = false$ and $D_t^p = R$, then $b_t^p = 1$. As the request is yet to serve, the *Reply* action corresponds to the "successful" action, and the h^P is the optimal custodian. The controller marks $isSatisfied = true$, updates the Q_R value, and sends an *Action(p)* message to the h^P if needed. In contrast, if $isSatisfied = true$ and $D_t^p = R$, then $b_t^p = 0$. As the requested *Data* is already sent back, the *Reply* action corresponds to the "unsuccessful" action. The h^P is the suboptimal custodian sending redundant replies contributing to the traffic load. The controller keeps $isSatisfied = true$ and updates the Q_R value. However, when $isRay = true$, the controller only marks the corresponding entry $isSatisfied = true$, if not already marked, as the h^P must respond with the *Reply* for the *Ray*.

On the other hand, for any second or later Reward(p) message received, if $isSatisfied = false$ and $D_t^p = M$, then $b_t^p = 0$. As the request is yet to serve, remaining silent with the *Mute* action corresponds to the suboptimal action as it would delay the content retrieval. The controller keeps $isSatisfied = false$ and updates the Q_M value. In contrast, if $isSatisfied = true$ and $D_t^p = M$, then $b_t^p = 1$. As the requested *Data* is already sent back, the *Mute* action suppresses any suboptimal response that would otherwise contribute to the traffic load. The controller keeps $isSatisfied = true$ and updates the Q_M value. However, $isRay = true$, the controller only sets $isSatisfied = true$ as the h^P must respond with the *Respond* for the *Ray*. Whenever the controller updates its Q-values for any h^P, it may change the action selected to entertain the next i_Ps and communicate the change of action to the h^P. The controller only sends an *Action(p)* message if the update of Q-values recommends the present action to change, upon receipt of an *Action(p)* message, a h^P updates the corresponding D_t^p in its *CS_Action* table. Fig. 3.4 depicts the reward determination and action selection process.

3.4.2.5 Optimizing *Interest* Forwarding

A user opting for P issues a i_P directed toward the gateway node. The user initializes the new fields $RequestID = arandomnumber$, $Ray = True$ and $Radius = 0$. If the gateway is a h^P, it immediately responds by returning the *Data* P without caring about the D_t^P field of its *CS_Action* table. The gateway (h^P) doesn't send

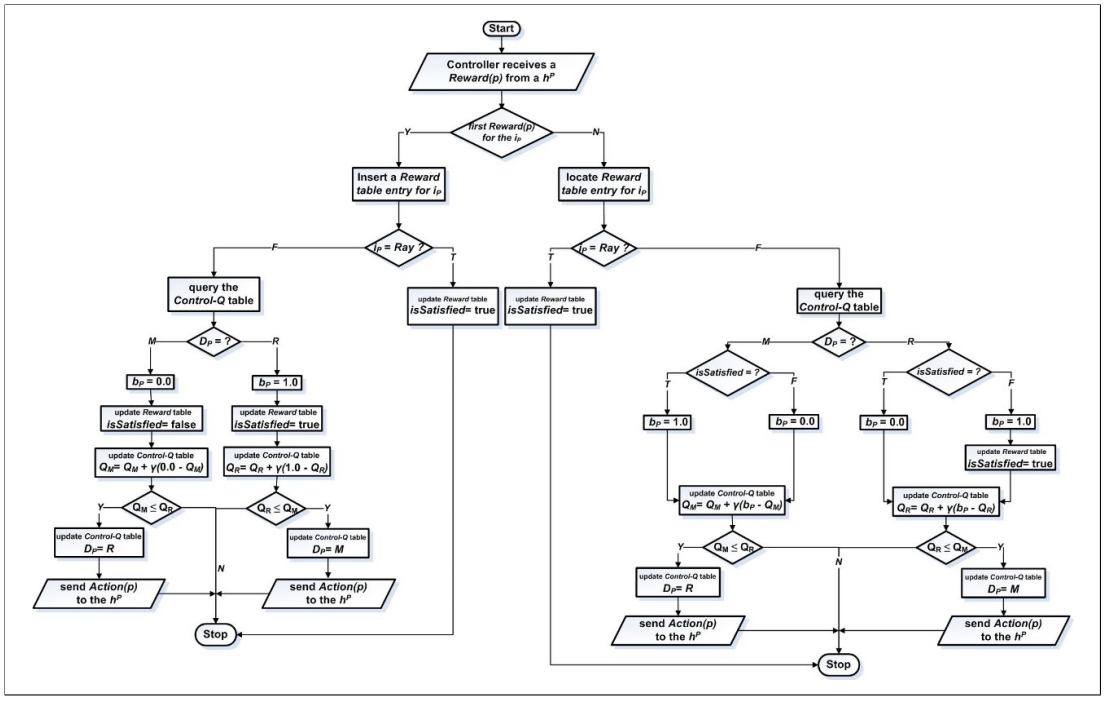


Figure 3.4: Reward determination and *Action* selection process

any *Reward(p)* message to the controller too. If the gateway is not a h^P, it sets the *Radius* field of the i_P as eq. 3.2. At any time t, any gateway node sets the *Radius* as:

$$Radius = \min_f \bar{X}_f^t \tag{3.2}$$

Each node finds the hop distance (*Hops*) of an incoming *Data* packet (from the source) received on each interface. For interface *f*, it is denoted as X_f and $X_f = Hops$. After that, the node finds the average hop distance of the *Data* packets, denoted as $\bar{X}_f^t$, received on *f*. At any time *t*, $\bar{X}_f^t$ is obtained as EWMA of the X_fs:

$$\bar{X}_f^t = (1-\sigma)X_f + \sigma\bar{X}_f^{t-1} \tag{3.3}$$

where:(i) parameter $\sigma \in (0,1)$ is set to 0.125 to avoid large fluctuations in the estimation and assign more weight to the historical values in front of the instantaneous ones, (ii) $\bar{X}_f^{t-1}$ is the previous value of $\bar{X}_f^t$. $\bar{X}_f^t$ is used for interface ranking such that the interface with a lower $\bar{X}_f^t$ value has a higher rank since contents are retrieved from or cached at a closer source.

After that, the gateway finds the outgoing interfaces from the *FIB* to move the i_P forward. The gateway sets $Ray = True$ for the *Interest* i_P thread outgoing through the interface having lowest $\bar{X}_f^t$, this thread is the i_P^Y. While $Ray = False$ for other threads of the i_P. If $\bar{X}_f^t$ is the same for all f, an interface is arbitrarily selected with $Ray = True$ for the outgoing thread through that interface and $Ray = False$ for all threads outgoing through other interfaces. The gateway suppresses the non-*Ray* threads if $Radius = 0$.

An intermediate node (which is not a h^P) receiving the i_P^Y thread updates the *Ray* field of the i_P as the gateway node. It sets $Ray = True$ for the thread (i_P^Y) outgoing through the best interface and $Ray = False$ for all other threads of the i_P. The intermediate node also updates *Radius* as $Radius = Radius - 1$. If $Radius = 0$, the non-*Ray* i_Ps are suppressed. On the contrary, if the node is a h^P, it finds the D_P^t decision from its *CS_Action* table. If the i_P thread hitting is the i_P^Y, the h^P responds with the matching *Data* and sends a *Reward(p)* message to the controller disregarding D_P^t. If the thread is a general i_P and $D_P^t = R$, the custodian returns the corresponding *Data* packet to the consumer and informs the controller by sending a *Reward(p)* message. A h^P initializes $Hops = 0$ every time it returns back a *Data* packet. The downstream switches update it as $Hops++$. The field is initialized to zero if the *Data* packet gets cached on-path. However, if i_P is general and $D_P^t = M$, the h^P remains silent, performing the *Mute* action but sending a Reward(p) message to the controller. The *Ray* ensures content retrieval as it is free to travel the entire network, and a custodian must respond to it with the matching *Data*; thereby, counteracting the *Mute* action of the custodians. Notably, any intermediate node receiving a *Ray* thread bypasses the *PIT*, if that entry is due to a non-*Ray* thread, forwards the *Ray* thread upward.

On the other hand, if a non-*Ray* strikes and the node is not a h^P, the node updates *Radius* as $Radius = Radius - 1$. It drops the i_P if $Radius = 0$, Otherwise, it keeps $Ray = False$ for the i_P and frontwards the thread.

Any server for p hit by an i_P must return back the corresponding *Data* packet to the user. *Interest* and *Data* communication in *SDN-Q* is depicted in Fig. 3.5.

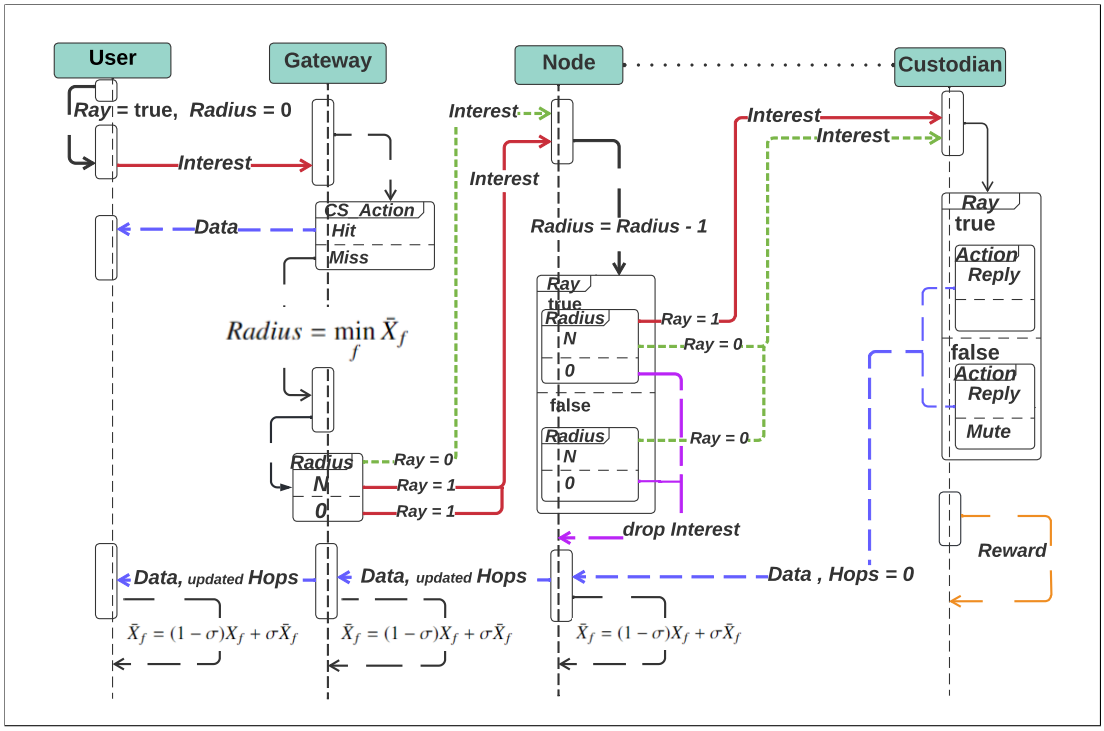


Figure 3.5: *Interest* forwarding and answering in *SDN-Q*, where N is any positive integer

The *Interest* and *Data* update operations performed at the nodes are trivial. Therefore the nodes are not loaded with computation burden, except computing $\bar{X}_f$ at the nodes. However, the learning task is offloaded to the controller saving a lot of *CPU* cycles compared to the strategies, such as *DTABB* and *SRAB*[3], where the nodes are responsible for performing the learning task.

3.4.2.6 Single-State *Q*-learning Algorithm for *Data* Replying

This study assumes a *RL*-based approach, namely a Single-State Q-learning algorithm, to solve the *Data* answering problem of a custodian. The proposed strategy intends to (i) minimize the traffic load by suppressing (*Muting*) the suboptimal custodians and (ii) reduce the retrieval delay by obtaining (Replying) the *Data* packet from an optimal holder. Q-learning is an online learning algorithm that records the reward values (successes and failures) for each action performed to estimate the Q-values. The algorithm selects the action that produces the best estimate. Once a i_P strikes at time t, a h^P responds with an action (a_t) and sends

a *Reward(p)* message to the controller. The controller determines the b_t^p for a_t using as in section 3.4.2.4. Next, he controller estimates the Q-value for action R at step t as:

$$Q_R^t = Q_R^{t-1} + \gamma(b_t^p - Q_R^{t-1}) \tag{3.4}$$

Similarly, the estimated Q-value for action M at tth time slot can be calculated as:

$$Q_M^t = Q_M^{t-1} + \gamma(b_t^p - Q_M^{t-1}) \tag{3.5}$$

The algorithm selects action R if $Q_R > Q_M$; otherwise selects action M. If the selected action differed from the last action, the controller updates D_t^p in the *Control-Q* table and sends down an *Action(p)* message to reflect the change in the h^P's *CS_Action* table. The Q-learning algorithm for *SDN-Q* problem is given in Algorithm 1.

Algorithm 1 Q-learning for *SDN-Q* (R, M, b_t^p, *Reward(p)*)

0: # in the controller
0: **for** $t = 1, 2, ...$ **do**
0: # estimate Q-values:
0: **for** $a = R, M$ **do**
0: Estimate $Q_a^t = Q_a^{t-1} + \gamma(b_t^p - Q_a^{t-1})$
0: **end for**
0: # select action as:
0: $a_t \leftarrow \underset{a}{\text{argmax}}\ Q_a^t$
0: **if** $D_{t-1}^p \neq a_t$ **then**
0: update *Control-Q* as $D_t^p = a_t$
0: send *Action(p)* message to the h^P
0: **end if**
0: **end for**
0: # in a custodian
0: update *CS_Action* as $D_t^p = a_t$
0: react with a_t to the next i^P =0

3.4.3 Theoretical Analysis of Optimizations

To justify the optimizations achieved in *Data* delivery strategy and *Interest* flooding in *SDN-Q*, we use asymptotic notations to examine the behavior of the performance metrics of the strategies as the input size grows.

3.4.3.1 Traffic Load Analysis

SDN-SRP: This strategy floods the entire network to find the optimal content source. The traffic load increases significantly with the number of clients and network nodes.

- Asymptotic Traffic Load: $O(\hat{C} \cdot |V|)$, where $\hat{C}$ is the number of clients and V is the number of nodes in the network. The traffic grows linearly with the number of clients and the network size due to the extensive flooding. It must be noted that the number of content sources increases with the number of nodes. The increased sources generate additional traffic and control messages.

BestRoute: Uses a single optimal route for content retrieval, resulting in the lowest traffic load.

- Asymptotic Traffic Load: $O(\hat{C})$. The traffic load increases linearly with the number of clients as only one path is used for each request.

SRAB[3] and *SDN-Q*: Both use a combination of localized search and some form of learning to optimize data retrieval, leading to moderate traffic loads.

- Asymptotic Traffic Load: $O(\hat{C} \cdot log|V|)$. The localized search reduces the overall network traffic compared to full flooding, leading to logarithmic growth with the network size.

3.4.3.2 Latency Analysis

SDN-SRP: High control message overhead due to network-wide *Interest* flooding and suppression of sub-optimal responses.

- Asymptotic Latency: $O(log|V|)$, The search for the optimal source increases logarithmically with the network size due to efficient retrieval from the closest source.

BestRoute: Experiences higher latency since it always uses the single best-known path.

- Asymptotic Latency: $O(|V|)$. Latency increases linearly with the network size as the single path might not always be the closest.

$SRAB^3$ and *SDN-Q*: These strategies balance between search overhead and path optimization.

- Asymptotic Latency: $O(log|V|)$. Intelligent learning and partial network exploration allow for logarithmic latency growth.

3.4.3.3 Control Message Overhead Analysis

SDN-SRP: High control message overhead due to network-wide interest flooding and suppression of sub-optimal responses.

- Asymptotic Overhead: $O(\hat{C} \cdot |V|)$. Due to extensive communication with the controller, control messages grow linearly with the number of clients and network size.

SDN-Q: Uses centralized learning to minimize control messages.

- Asymptotic Traffic Load: $O(\hat{C} \cdot log|V|)$. Overhead grows with the number of clients and the network size due to the combination of centralized learning and partial network exploration.

Chapter 4

Simulation and Results

SDN-Q is implemented in ndnSIM [**?**] modifying the built-in *Interest* and *Data* packets [100] and *CS* structure. The simulation scenarios are analogous to the literature, especially [21, 22, 70, 101], concerning workload, resource availability, and network topology for ease of implementation and comparison. The rest of the section describes the performance metrics, simulation scenarios, essential assumptions and parameter selection and finally analyzes the experiment results.

4.1 Performance Metrics

The experiments find the credentials of *SDN-Q* in reducing traffic load and retrieval delay by countering its performance against flooding (*Multicast* [100]), single-route (*BestRoute*, scoped-flooding ($SRAB^3$), *SDN*-based content retrieval strategy (*SDN-SRP*). Among the counterparts, *Multicast* and *Best-Route* are built in ndnSIM [100]) and we implemented $SRAB^3$ and *SDN-SRP*. The strategies are compared primarily based on typical performance metrics, such as traffic load, cache hit ratio, latency, cache-eviction rate, packet delivery ratio (*PDR*) [22, 95, 96, 102, 103], and combined significance [21, 104]. In addition, we also compare the control packets incurred by *SDN-Q* and *SDN-SRP*. The performance metrics are defined below:

- **Traffic load**: it corresponds to the magnitude of *Interest* and *Data* packets,

and relevant messages if any, for instance, *AckNode* messages in $SRAB^3$, communicated to retrieve the requested objects, measured in megabytes(MB).

- **Cache hit ratio**: it is the ratio of the consumer requests answered from routers' *CS*s to the entire requests that arrived at the routers.
- **Latency**: it refers to the seconds between a consumer sending a request and downloading the desired object, averaged over all downloaded packets.
- **Cache replacement rate**: it denotes the ratio of the number of cache replacements to the number of cache misses.
- ***PDR***: is the ratio of the number of objects downloaded to the number of objects solicited by the customers.
- **Control messages**: it refers to the number of control packets exchanged between the *SDN* controller and *NDN* nodes.
- **Combined significance**: it is obtained as the ratio of PDR and product of latency and traffic load. The combined metric reveals the productivity of any scheme; the user experience a scheme provides for the cost it incurs; the higher the value, the better the productivity. We do not consider the control message volume in the equation due to the use of out-of-band communication. It is defined as:

$$Combined\ significance = \frac{PDR}{traffic\ load \times latency}$$

Since we propose to optimize *Interest* flooding and suppress sub-optimal *Data* replies to reduce traffic load and latency, the selected performance metrics can measure the magnitude achieved by *SDN-Q*. *PDR*, latency, and cache hit ratio assist in determining the effectiveness of a scheme and can impact user experience. Whereas traffic load, control messages, and cache eviction rate aid in gauging the efficiency, resource consumption, or resource friendliness of the participating policies. Moreover, higher cache hits resemble a lower workload on the original content servers or producers.

4.2 Network Topology

Add more in ndnSIM ndnSIM stands out as a known open source network simulator specially crafted for Named Data Networking (*NDN*). *NDN*, a data networking architecture prioritizes fetching content based on names instead of traditional location based *IP* addresses. Lets delve into some aspects of ndnSIM. ndnSIM serves as an extension module for the *NS3* network simulator. *NS3*, known for its discrete event simulation capabilities acts as a platform for developing network protocols and exploring their performance across network settings. By integrating with *NS3* ndnSIM enhances support for *NDN* functionalities. Within the *NS3* framework ndnSIM effectively implements the *NDN* protocol stack encompassing components like *Interest*/*Data* exchange mechanisms, forwarding strategies and naming conventions. Users have the freedom to design custom network topologies and define traffic patterns using ndnSIM. This empowers researchers to simulate network scenarios involving sizes, topologies and traffic loads. With ndnSIMs configuration features users can fine tune aspects, like cache policies, forwarding strategies and content store sizes related to *NDN* operations. This flexibility facilitates exploration of how different configurations influence network performance and behavior.

The generate identical *NDN* topologies like [21, 22, 70] according to the Barabasi-Albert model realized in *BRITE* [105] Internet topology generator. Each arbitrary topology resembles a graph of 50 vertices categorized into two domains, with each domain comprising 25 distinct vertices. The inter- and intra-domain link bandwidths are randomly distributed between 10-100 Mbps. The default Barabasi-Albert model connects the two domains using only a single physical link; we added two additional 100 Mbps links between the domains to form multiple paths between the portions. Furthermore, each vertex is connected (out-of-band connection) to an *SDN* controller. A leaf node is a vertex connected to its domain (network) via only a physical link. A node is a non-leaf node if it is connected to the rest of its domain via two or more links. Each bar in the performance charts is obtained by repeating the corresponding experiment ten times and then averaging all runs for accuracy. In every run, an arbitrary *NDN* topology with the same

number of nodes but a different connectivity configuration among the nodes is assumed.

4.3 Traffic Generation

A content (or object) is hosted in five distinct servers to allow an *Interest* to reach the object via different paths. *Interest* flooding control can be applied in such conditions. We consider a catalogue of 500-2000 chunks or pieces for the hosted content, and the clients assume the famous Zipf-Mandelbrot popularity model [106, 107], to generate requests for the hosted chunks. Each client and each server is linked to an arbitrary leaf node (gateway node). The servers please consumer *Interest*s following the built-in ndnSIM producer application.

In each experiment, the clients request content pieces at a constant rate, and we alter the *Data* payload size and client volume to produce diverse traffic loads to compare the proposed and reference strategies. A client sends *Interest* for chunks from the first second of every experiment, and each experiment is resumed for 100 seconds. After the first 100 seconds, the magnitude change of the performance metrics is insignificant. Each bar in the performance chart is achieved through the mean and standard deviation of 10 randomized simulation runs.

4.4 Caching Policy and Capacity

We consider Random Caching and *LRU* as the cache insertion and replacement policies, respectively. Each router is equipped with a given maximum capacity (size), and initially, the caches are vacant. Unlike [21, 22], we prefer Random Caching to leave copy everywhere (*LCE*) as it admits and thus evicts contents less aggressively than *LCE*. *LCE* is mainly used to contrast the performance of a newly developed caching strategy and *LRU* is widely adopted in *NDN*. Since *SDN-Q* executes *Q*-learning only for cached items, a steady cache admission policy, such as *Random Caching* is picked to retain cached items for a longer span. Moreover, *Random Caching* doesn't require any packet or structure extension/modification. We vary the maximum cache capacity of a router to retain a percentage of the total catalogue size; detailed in the next sections.

4.5 Experiment Groups

We consider four experiment groups, each obtained by varying a different parameter, to quantify the performance of the proposed and the opponent strategies. The four parameters are client size, catalogue size, cache capacity, and payload size. In the first simulation group, we vary the client volume from 100-500 clients to gauge the strategies under changing traffic loads. However, the remaining three parameters are kept constant at catalogue size = 500 chunks, cache capacity = 100 items, and payload = 1024 bytes. The second group corresponds to performance against different catalogue sizes, for instance, 500-2000 chunks, while keeping other parameters constant, such as client size = 100, cache capacity = 100 items, and payload = 1024 bytes. The varying catalogue size aids in assessing the strategies when the client requests range from a narrow (common) to a broad (diverse) spectrum of the catalogue. The third experiment evaluates the schemes under different cache capacities or resource constraints. Each *NDN* node can cache a portion of the catalogue, 100-20 items or equivalently 20%-4% of the catalogue size. The rest of the parameters remain fixed at client size = 100, catalogue size = 500 chunks, and payload = 1024 bytes. Finally, the payload (*Data*) is varied from 1024-4096 bytes, keeping the other three parameters unchanged at client size = 100, catalogue size = 500 chunks, and cache capacity = 100 items. The payload size variation gauges the strategies regarding saving control messages. Table 4.1 provides different assumptions for the four simulation groups.

Table 4.1: Simulation parameters

Parameters	***Experiment Groups***			
	Varying Client Size	***Varying Catalogue Size***	***Varying Cache Capacity***	***Varying Payload Size***
Content popularity parameter (α)	0.7	0.7	0.7	0.7
Client Size	100-500	100	100	100
Data payload size (bytes)	1024	1024	1024	1024-4096
Catalogue size (chunks)	500	500-2000	500	500
Caching capacity (items)	100	100	100-20	100
Interests/sec	10	10	10	10
RadCntrl (*SRAB*[9])	3	3	3	3

4.6 Performance against Client Size Variation

For each client size, the simulation is repeated 10 times. Fig. 4.1 represents a random ndnSIM scenario when the client size is varied from 100 to 500 clients. The leaf nodes comprise consumer and producer nodes. The clients generate *Interest* packets to request specific content and the producers respond with *Data* packets when content is available. The nodes are distributed randomly across a simulated area. This experiment group examines the consequence of changing user magnitude on the forwarding schemes, depicted in Fig. 4.2; it is apparent that the traffic volume of the participating strategies surges proportionally as the client number rises from 100 to 500, Fig. 4.2a. *SDN-SRP*, where users flood requests in the entire network to retrieve the corresponding *Data* from the best source, experiences a higher traffic load than other schemes as a few sub-optimal sources respond in addition to the optimal one. Concurrently, the policy enjoys a lower retrieval delay as it is more likely to hit a closer replica. *BestRoute* confronts an opposite phenomenon; the lowest traffic load but higher content download time as it exploits a single route for content retrieval.

SRAB^3 and *SDN-Q* primarily use the *Ray* (*Beam*) to retrieve a content; however, explore a particular portion of the remaining network for a better (nearer) replica or copy. This phenomenon leads to *SRAB*^{3} and *SDN-Q* injecting slightly higher traffic than *BestRoute*, however much lower than *SDN-SRP*. *SDN-Q* realizes *RL*, Single-State Q-learning, with the aid of the controller. It increases the volume of out-of-band control messages indeed; at the same hour, it avoids the usage of any in-band additional control packets, for instance, *AckNode* message used by *SRAB*^{3}. Thus, *SDN-Q* pumps lower traffic, approximately 3-4.5%, than *SRAB*^{3} as it eliminates the *AckNode* messages. *SRP* discovers the entire network but conceals sub-optimal responses with the aid of the controller. Furthermore, *SDN-SRP* trades significantly higher control messages with the controller in trying to incite the optimal and suppress the sub-optimal sources. It has a higher traffic load, for instance, roughly 20-12%, than *SDN-Q*.

Again, the diversity of content requests increases with the client size causing the closer caches to retain only a tiny portion of the requested contents. As the cache admission policy is the same in the competing forwarding strategies, the

cache hit rates exhibit no such difference as exhibited in Fig. 4.2b. *SDN-SRP* is marginally better than the others as it explores the entire network, bringing and retaining more copies into the nodes' caches. However, the latency performances Fig. 4.2c of all strategies decline as more *Interest*s need to travel further to retrieve the desired contents. *SDN-Q* has approximately 5-7% and 0-6% lower retrieval latency than *BestRoute* and *SRAB*^{3}, respectively, as depicted in Fig. 4.2c. *SDN-Q* is slightly better regarding latency for higher client sizes as the *SDN* controller is involved in the learning process, enabling more accurate actions (responses) by the custodians. Nevertheless, the latency is higher than *SDN-SRP* by around 6-2%. *SDN-SRP* achieves it at the cost of much higher overhead, such as 20-12%, making *SDN-Q* more significant Fig. 4.2f than it, for instance, by approximately 11-10%. Again, *SDN-Q* is superior to *SRAB*^{3} by approximately 3-10%, and inferior to *BestRoute* by roughly 2-1%. However, the gap between *SDN-Q* and *BestRoute* diminishes for the higher payload size, from 2% to 1% for 100 client size, as depicted in Fig. 4.3f.

SRAB^3, *SRP*, and *SDN-Q* suppress sub-optimal replies to minimize the traffic load, which reduces the frequent cache eviction as a byproduct, resulting in a lower cache replacement rate. *BestRoute* does not suppress any reply *Data* packet; each *Data* packet must travel back to the consumer. Here, the downstream nodes suffer from frequent cache replacement. *SDN-Q* and *SRAB*^{3} discover a minor portion of the network compared to *SDN-SRP*, suppressing a lower number of *Data* replies. regarding. Fig. 4.2d portrays the cache replacement phenomena.

Notably, the *PDR* of the participating forwarding strategies is higher than 0.99, and hence, the combined significance is mainly determined by traffic load, control messages, and latency. This is maintained for all simulation groups and thus *PDR* is not plotted in the graphs. Finally, Fig. 4.2e displays that the control packets increase with the number of users, 100-500 clients, as there are relatively more cache hits and so as cache misses. The control message volume of *SDN-Q* is remarkably lower than *SDN-SRP*, for instance, by roughly three (3) times. *SDN-Q* upwards *Reward* messages to the controller only when hit by the *Interests*, and the controller downwards *Action* messages only when it needs to update a custodian's action. In contrast, *SDN-SRP* needs to trade multiple messages between the controller and *NDN* nodes to satisfy a request and suppress sub-optimal replies.

4.7 Performance against Payload Size Variation

Data payload size is the primary contributor to the traffic load generated in the network. The experiment group in Fig. 4.3 reveals that the traffic load gap between *SDN-Q* and counterparts shrinks as the *Data* packets piggyback more bytes. The volume of overhead incurred due to the header extension of the *Interest* and *Data* packets and exchange of control packets remain stationary despite the payload size increases. The decrease of overhead and the traffic load ratio allows the traffic load difference between *SDN-Q* and others to shrink. The difference between *SDN-Q* and *BestRoute* and *SDN-Q* and $SRAB^3$, depicted in Fig. 4.3a, reduces from approximately 7% to 5.5% and 3% to 1%, respectively, for payload size 1024-4096 bytes. *SRP* has much higher load due to network-wide *Interest* flooding. Suppression of sub-optimal *Data* packets by *SDN-Q*, *SRP*, and $SRAB^3$ releases the available network bandwidth to deliver other optimal *Data* packets faster, especially for larger payload sizes. The latency performance difference between *SDN-Q* and *SRP* as given in Fig. 4.3c minimizes from roughly 6% to 2%, as the *Data* size surges from 1024-4096 bytes.

The cache hit ratio Fig. 4.3b, cache eviction rate Fig. 4.3d, and *PDR* performances of all candidates remain unchanged as all other network parameters, apart from the payload size, are kept constant. Content retrieval latency Fig. 4.3c inclines proportionally to the payload size. Together traffic load and latency change bring down the combined significance from high to low. However, the relative difference between *SDN-Q* and other schemes reduce, making the proposed scheme comparatively more significant than the reference strategies. For example, *SDN-Q* is at least 2% ($SRAB^3$) and at most 15% (*SDN-SRP*) more significant than others. It's worth highlighting that the *SDN* controller and *NDN* switches exchange an equal volume of control packets, thereby not being affected by the payload size variation. This is unlike the magnitude of control messages that increased almost linearly to the client volume.

4.8 Performance against Cache Capacity

The *RL* agents in *SDN-Q* and *SRAB*^{3} execute the learning tasks only for cached contents. The learning task starts as soon as a content is admitted and is resumed until the content is evicted. The learning credentials, for instance, Q-values in *SDN-Q* are all useless once the content is kicked out. The learning task, and hence the credentials, start afresh next time the content is cached again. The learning span is significant, and therefore, a router's caching capacity is vital to the performance of the *SDN-Q* and *SRAB*^{3}. Fig. 4.4 represents the cache capacity's impact on the competing forwarding strategies' performances.

It is straightforward that in-network caching becomes effective as the cache capacity of each router increases from 20-100 items or 4%-20% of the catalogue size, as a higher caching capacity can accommodate more contents that serve user requests quickly.

When the cache capacity rises, *SDN-Q* maintains a 3% performance advancement compared to *SRAB*^{3} and 16% improvement than *SDN-SRP* regarding traffic load. While it exhibits an inferior performance (7%) when contrasted against *BestRoute*, as depicted in Fig. 4.4a.

The pictures suggest that *SDN-SRP* has the highest latency performance, around 6% compared to *SDN-Q* Fig. 4.4c. *SDN-SRP* achieves it with the help of the controller and enormous control messages. *SDN-Q* has a marginal performance advantage over *SRAB*^{3} as the *SDN* controller aids in responding with the optimal action and suspending the sub-optimal responses. The learning efficiency of *SDN-Q* becomes evident for cache constraint scenarios (lower cache capacities) when it shows a better cache hit performance than *BestRoute* (by closely 12-1% for cache capacities 20-80 items) and *SRAB*^{3} (by closely 1% for cache capacities 20-100 items), as given in Fig. 4.4b. *SDN-Q* responds with a comparatively higher number of *Reply* actions than *SRAB*^{3}. A higher number of *Reply* actions invites more *Data* packets being replied to the content solicitor. This phenomenon declines the cache replacement performance of *SDN-Q* by around 3% than *SRAB*^{3}, exhibited in Fig. 4.4d. Nevertheless, the higher number of *Reply* actions enables content retrieval from nearer sources, reducing the latency of *SDN-Q* than *SRAB*^{3}. *SDN-SRP* is the best performer in terms of cache hit and replacement as it explores

the entire network and mostly suppresses the sub-optimal responses. In contrast, *BestRoute* is the worst performer regarding the two metrics as it explores only the best path. Searching only through the best interface is not beneficial for a lower caching capacity. Again, the sources must respond with the *Data* packets, which compels more cache replacements in the downstream nodes.

Here, the volume of control packets decreases as the routers' caching capacity surges from 20-100 items, as depicted in Fig: 4.4e. The cache hit ratio increases with larger cache sizes, satisfying content requests from the nearest caches (gateway routers) and minimizing the number of *Reward* messages (in *SDN-Q*) and control messages (*SDN-SRP*) sent to the controller. Remarkably, the volume of control messages of *SDN-Q* is significantly lower, for instance, three (3) times, than *SDN-SRP*. The combined significance of *SDN-Q*, as exhibited in Fig: 4.4f, is better than *SDN-SRP* and $SRAB^3$ by around 11% and 4%, respectively, and the same as *BestRoute*.

4.9 Performance against Catalogue Size Variation

Fig. 4.5a demonstrates the declining performance of the forwarding strategies when the catalogue range widens. When the catalogue size broadens, on the one hand, the ratio of the cache size to the catalogue size decreases. On the other hand, diverse items are wanted by the user base, which essentially deteriorates the in-network caching advantage Fig: 4.5b. The traffic load performance deteriorates with the increasing catalogue size. Under such circumstances, *SDN-Q* stands out comparatively better than *SDN-SRP* and $SRAB^3$.

BestRoute generates the lowest traffic load (by virtue of single-path content retrieval) while *SDN-SRP* is the worst performer in terms of network traffic. *SDN-Q* contributes marginally higher, approximately 7-5.5%, traffic load than *BestRoute* but 3% lower than $SRAB^3$, as the catalogue size increases from 500-2000 chunks. *SDN-SRP* achieves the lowest latency as the controller ensures content delivery from the nearest (best) replica; however, at the cost of around 20-40% higher traffic load than the proposed strategy. *SDN-SRP* explores the

entire network searching for the desired content, ending up fetching distant copies in addition to the closest one. *SDN-Q*, *SDN-SRP*, and *SRAB*^{3} ignore a few *Data* packets from caching, leading to much lower cache-in and -out frequency than *BestRoute*. Regarding the cache hit ratio, *SDN-SRP* shows a considerable improvement of around 4-0% over *SDN-SRP* as the catalogue size widens. Since *SDN-Q* and *SRAB*^{3} can adapt the flooding ball, they locate the content sources more efficiently and hit closer caches (if any) via the non-*Ray* or -Beam threads. Notably, the latency of *SDN-Q* and *SRAB*^{3}, given in Fig: 4.5c, is never lower than the best-path strategy as they ensure a content download through the *Ray* or *Beam*. Their performances converge as the catalogue size grows. A wider catalogue injects diverse *Interests* into the network, causing frequent cache replacements. Since the two strategies' learning tasks are performed only for the cached items, their performances degrade. *SDN-Q* and *SRAB*^{3} suffer slightly lower latency performance, for instance, around 6% each, than *SDN-SRP*.

Regarding control packets, *SDN-Q* has a much lower overhead than *SDN-SRP* as given in Fig: 4.5e. The *Reward* messages sent to the controller reduce relatively as the number of cache misses inclines with the size of the catalogue. However, a higher number of cache insertions increases the *Insert(p)* messages sent to the *SDN* controller. As a result, the volume of the control message increases with the catalogue size. Remarkably, the volume of the overhead of *SDN-Q* is significantly lower than, for instance, around 3-4 times, *SDN-SRP*. The combined significance indicates that *SDN-Q* is almost as resource-friendly as *BestRoute*, depicted in Fig: 4.5f, despite pumping slightly more traffic than the latter.

4.10 Discussion

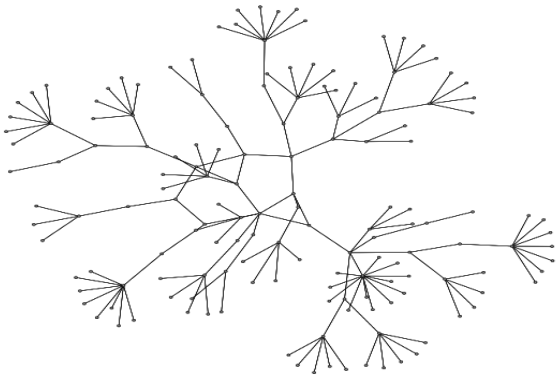
The performance of *SDN-Q* relative to other strategies such as *SDN-SRP*, *BestRoute*, and *SRAB*^{3} is notable in several aspects including latency, traffic load, and control messages. The superiority of *SDN-Q* can be attributed to its design and implementation of Single-State Q-Learning.

SDN-SRP exhibits the highest traffic load due to its network-wide request flooding, often resulting in responses from sub-optimal sources alongside the optimal ones. *BestRoute* achieves the lowest traffic load by using a single path for

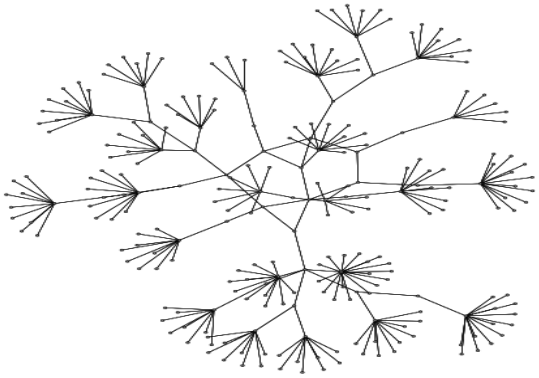
content retrieval, which results in higher content download times. However, neither strategy uses any learning algorithm. *SRAB³* and *SDN-Q* generate moderate traffic by exploring specific portions of the network to find closer replicas, leading to slightly higher traffic than *BestRoute* but significantly less than *SDN-SRP*. *SDN-Q* reduces the need for in-band control packets, further lowering its traffic compared to *SRAB³*.

SDN-Q performs better in terms of latency compared to *BestRoute* and *SRAB³*, thanks to its *Q*-learning-based optimization, which results in more accurate responses from the network. However, it still has a higher latency than *SDN-SRP* due to fewer control messages and a less exhaustive search. *BestRoute* and *SRAB³* exhibit higher latencies due to limited search scope and reliance on single or partially explored paths.

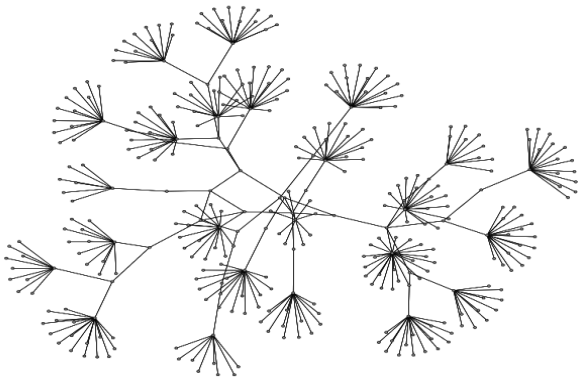
SDN-SRP incurs a high volume of control messages to manage optimal and sub-optimal sources, leading to significant overhead. *SDN-Q* balances control message volume and efficiency better, avoiding the extensive use of in-band control packets like *AckNode* messages, which *SRAB³* uses, thus pumping lower traffic. *SDN-Q*'s volume of control messages is notably lower than *SDN-SRP*, as it only exchanges messages with the controller when necessary.



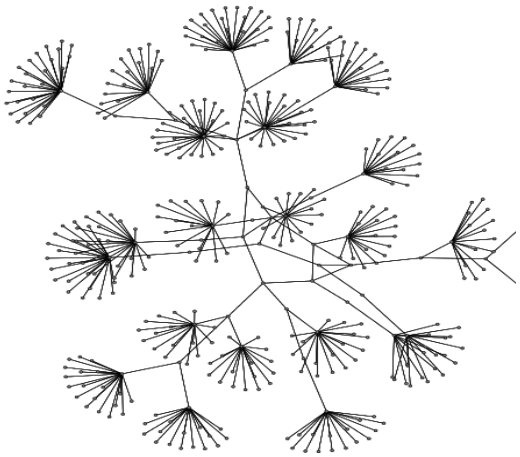
(a) 100 Clients



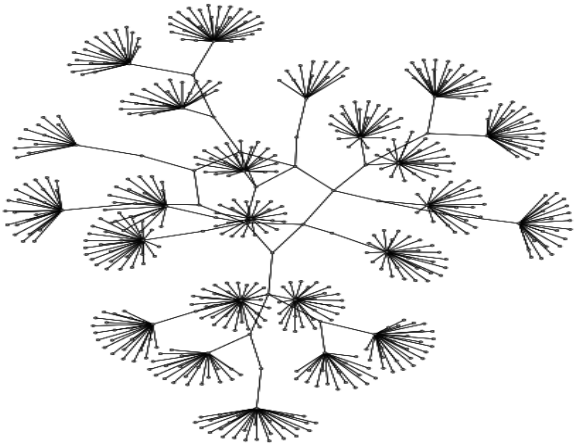
(b) 200 Clients



(c) 300 Clients

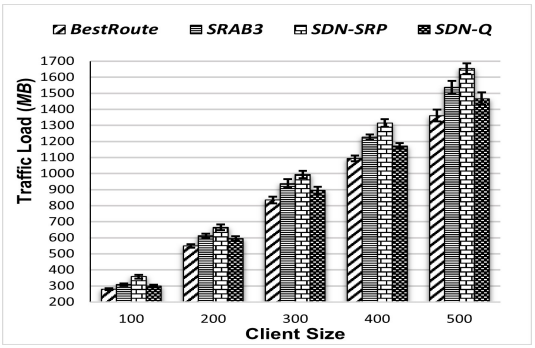


(d) 400 Clients

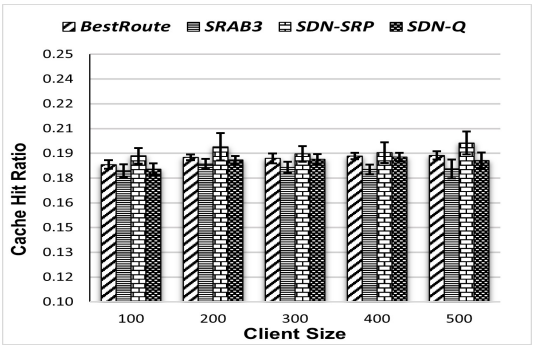


(e) 500 Clients

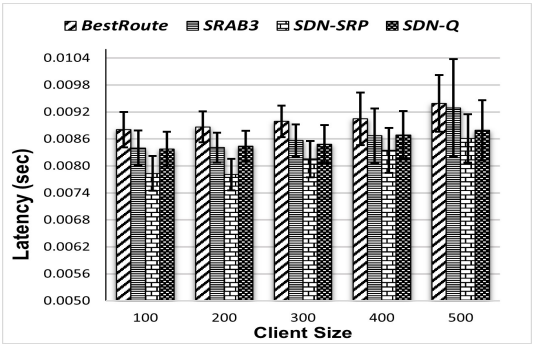
Figure 4.1: Simulation environment for 100, 200, 300, 400, 500 clients



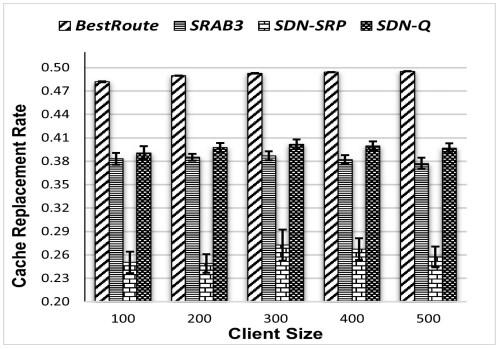
(a) Traffic load



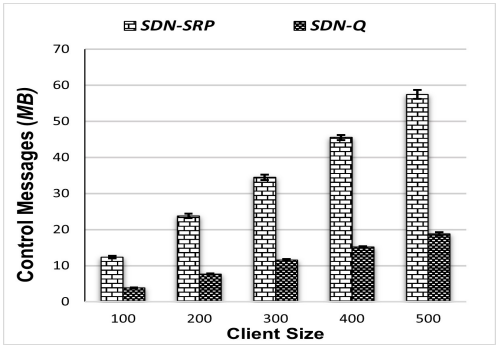
(b) Cache hit ratio



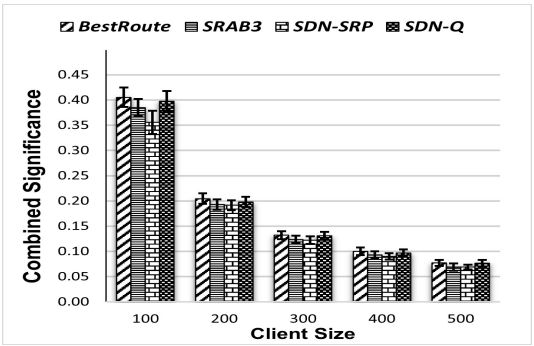
(c) Latency



(d) Cache replacement rate

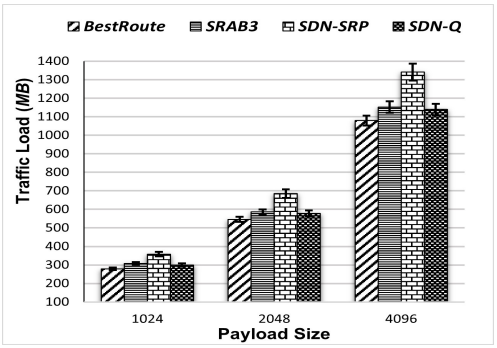


(e) Control messages

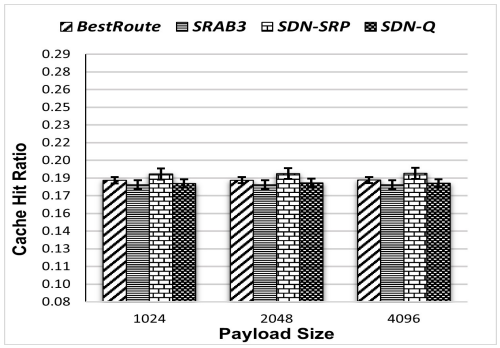


(f) Combined significance

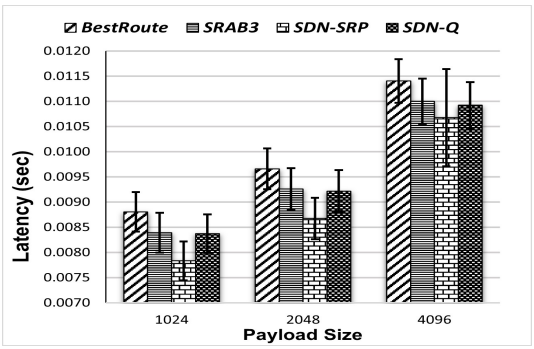
Figure 4.2: Performance assessment varying client size and unchanging catalogue size, cache size, and payload size



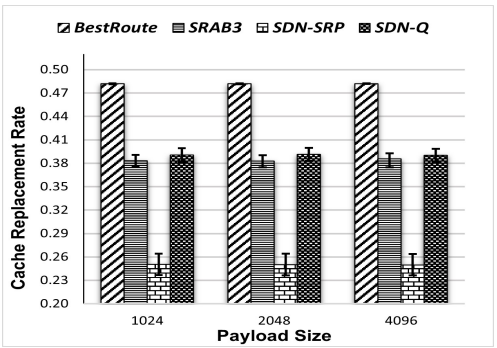
(a) Traffic load



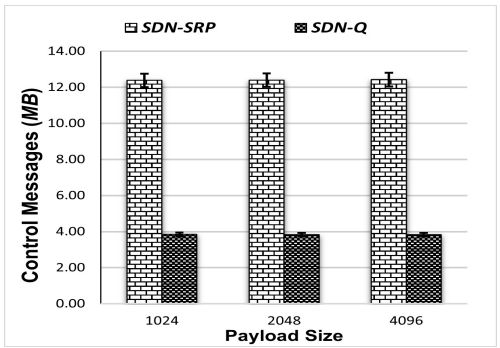
(b) Cache hit ratio



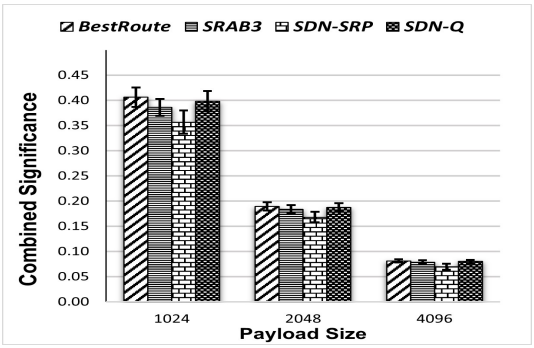
(c) Latency



(d) Cache eviction rate

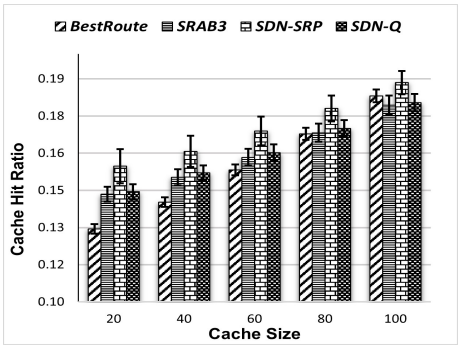


(e) Control messages

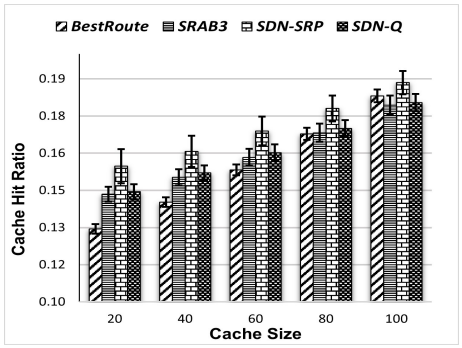


(f) Combined significance

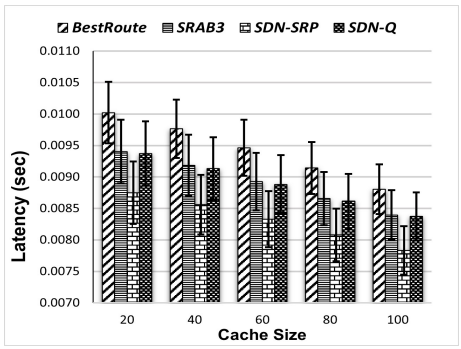
Figure 4.3: Performance assessment varying payload size and unchanging client size, catalogue size, and cache size



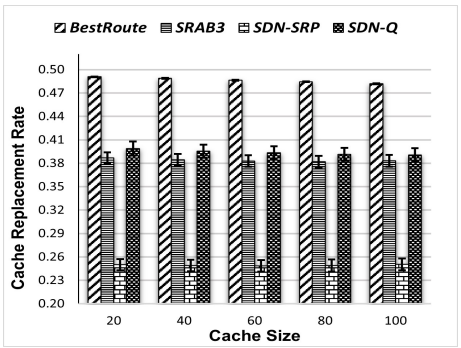
(a) Traffic load



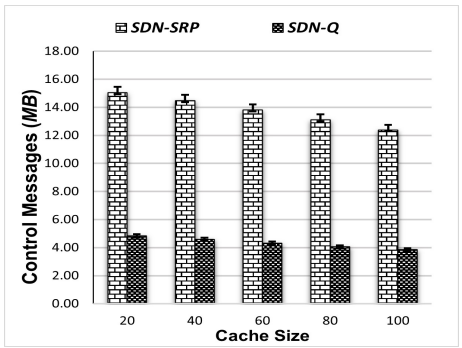
(b) Cache hit ratio



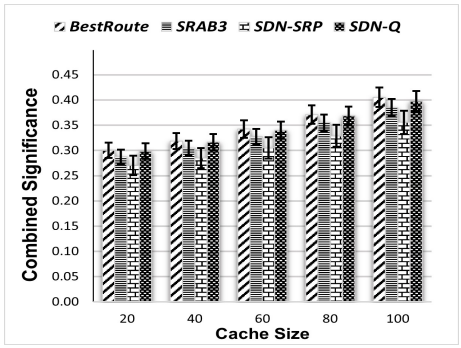
(c) Latency



(d) Cache eviction rate

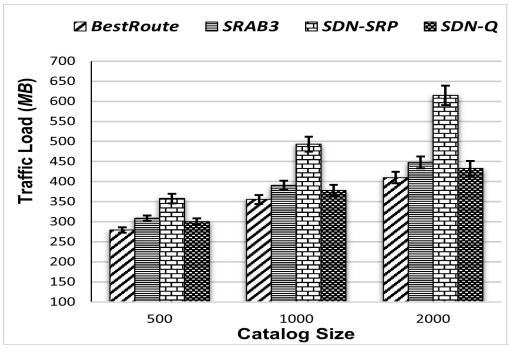


(e) Control messages

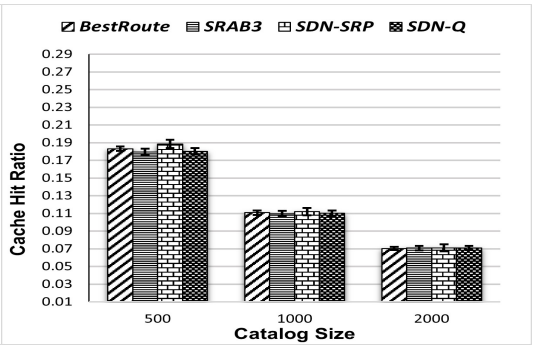


(f) Combined significance

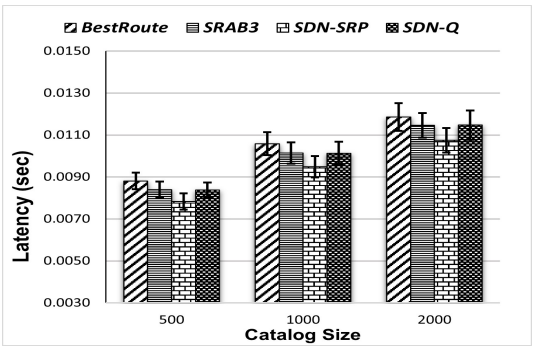
Figure 4.4: Performance assessment varying cache size and unchanging client size, catalogue size, and payload size



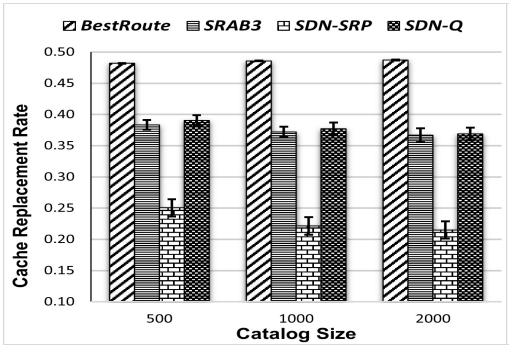
(a) Traffic load



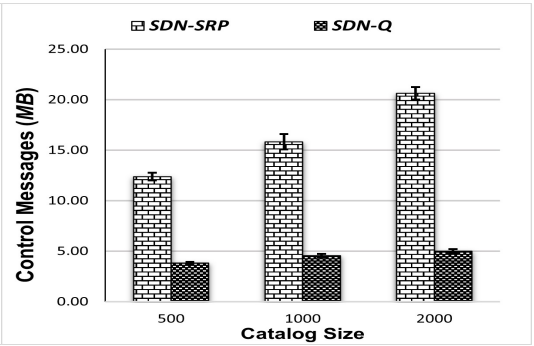
(b) Cache hit ratio



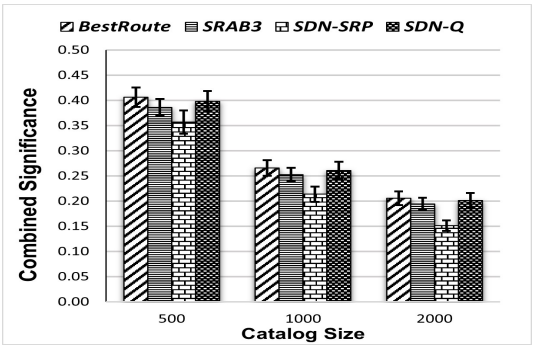
(c) Latency



(d) Cache eviction rate



(e) Control messages



(f) Combined significance

Figure 4.5: Performance assessment varying catalogue size and unchanging client size, cache size, and payload size

Chapter 5

Conclusions, Limitations and Future Works

5.1 Conclusions

This paper proposes *SDN-Q* that mainly optimizes the *Data* delivery phase in *SDN*-based *NDN*. Here, each *NDN* node disseminates an *Interest* through all outgoing interfaces; however, it ensures the corresponding *Data* replies at least through one interface and occasionally through multiple interfaces. The *Interest* instance ensuring the *Data* reply is the *Ray* instance and other threads are the non-*Ray Interest*s. Using a hop-based mechanism, the non-*Ray Interest*s are confined to a limited network region. Any content custodian answers the non-*Ray Interest*s selectively. Each custodian is considered a *RL* agent that utilizes Single-state *Q*-learning in association with the *SDN* controller to decide the answer type whenever an *Interest* strikes. The extent of non-*Ray Interest*s and correct replies by the *Q*-learning agents control the traffic load and *Data* retrieval delay in *SDN-Q*.

SDN-Q is contrasted against *BestRoute*, *SDN-SRP*, and $SRAB^3$. The performance of the strategies is assessed by varying four important network parameters: varying client size, payload size, caching capacity, and catalogue size.

Simulation results advise that when the client size varies *SDN-Q* consumes a lower traffic load, around 3-4.5% than $SRAB^3$, and 20-13% less than *SDN-SRP*. While it achieves 5-7% and 0-6% lower latency than *BestRoute* and $SRAB^3$,

respectively. Furthermore, the results also point out that when contents are solicited from a wider content library *SDN-Q*'s performance improves by 5% in terms of latency.

5.2 Limitations

Some issues are encounter by *SDN-Q* which are listed below:

- The nodes (content sources) need an uninterrupted connection to the controller. Once the connection between the controller and a source fails, the source retains the prior state before the disconnection and accordingly serves the subsequent incoming *Interest*s. This phenomenon may lead the source to respond with a sub-optimal action.
- The controller assumes a direct connection to each *NDN* node to render the learning decision promptly. For an extremely large-scale *SDN*-based *NDN*, where multiple controllers are arranged in a hierarchy, the *SDN-Q* solution may suffer a higher latency to forward the states to the nodes.

5.3 Future Work

In *SDN-Q*, the controller exchanges control messages with the content sources in real-time to update the actions' Q-values to optimize network performance, retrieve content from the best source to reduce latency and suppress the sub-optimal replies to reduce traffic load. However, this phenomenon may limit the scalability and efficiency of *SDN-Q*, especially in large and dynamic networks where control packets traded may become unmanageable.

By adopting centralized learning and decentralized execution, the network controller will train the model by aggregating behaviors (mute or reply actions) from different content sources. This centralized training process will allow the agents at the controller to make better-informed decisions (recommend actions) to the content sources. The controller will communicate the decisions to the sources concurrently instead of conveying the recommendations individually. Thus, the

future approach will reduce control message volume and eliminate the need for real-time communication to suppress redundant replies.

Bibliography

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012. 1

[2] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013. 1

[3] R. A. Rehman and B.-S. Kim, "Lomcf: Forwarding and caching in named data networking based manets," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 10, pp. 9350–9364, 2017. 1

[4] P. Wendell and M. J. Freedman, "Going viral: flash crowds in an open cdn," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 549–558. 1

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12. 1, 2

[6] A. Kalghoum and L. A. Saidane, "Fcr-ns: a novel caching and forwarding strategy for named data networking based on software defined networking," *Cluster Computing*, vol. 22, no. 3, pp. 981–994, 2019. 1, 3, 4, 35

[7] J. Li, R.-c. Xie, T. Huang, and L. Sun, "A novel forwarding and routing mechanism design in sdn-based ndn architecture," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 9, pp. 1135–1150, 2018. 2

[8] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013. 2, 34

[9] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014. 2, 3

[10] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, 2010. 3

[11] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014. 3

[12] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014. 3

[13] T. A. Kelley, "A note on the bernoulli two-armed bandit problem," *The Annals of Statistics*, vol. 2, no. 5, pp. 1056–1062, 1974. 3

[14] O.-C. Granmo, "Solving two-armed bernoulli bandit problems using a bayesian learning automaton," *International Journal of Intelligent Computing and Cybernetics*, 2010. 3

[15] J. Lv, X. Wang, M. Huang, J. Shi, K. Li, and J. Li, "Risc: Icn routing mechanism incorporating sdn and community division," *Computer Networks*, vol. 123, pp. 88–103, 2017. 4

[16] M. Alhowaidi, D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, "Multipath forwarding strategies and sdn control for named data networking," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2018, pp. 1–6. 4, 26, 34

[17] E. Aubry, T. Silverston, and I. Chrisment, "Srsc: Sdn-based routing scheme for ccn," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5. 34, 35, 36

[18] N. L. van Adrichem and F. A. Kuipers, "Ndnflow: Software-defined named data networking," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[19] A. Kalghoum, S. M. Gammar, and L. A. Saidane, "Towards a novel forwarding strategy for named data networking based on sdn and bloom filter," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2017, pp. 1198–1204. 4, 34, 35

[20] S. Eum, M. Jibiki, M. Murata, H. Asaeda, and N. Nishinaga, "A design of an icn architecture within the framework of sdn," in *2015 Seventh International Conference on Ubiquitous and Future Networks*. IEEE, 2015, pp. 141–146. 4

[21] S. M. A. Iqbal and M. M. Hoque, "A source-driven probabilistic forwarding and caching strategy in ndn and sdn-based ndn," *International Journal of Communication Systems*, p. e5093. 4, 36, 41, 58, 60, 61

[22] N. Akther, K. Dhar, S. M. A. Iqbal, M. N. Huda *et al.*, "Interest forwarding strategy in named data networks (ndn) using thompson sampling," *Journal of Network and Computer Applications*, vol. 205, p. 103458, 2022. 5, 36, 41, 58, 60, 61

[23] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013. 8

[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, 2007, pp. 181–192. 8

[25] S. Tarkoma, M. Ain, and K. Visala, "The publish/subscribe internet routing paradigm (psirp): Designing the future internet architecture." in *Future Internet Assembly*, 2009, pp. 102–111. 8, 9

[26] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, 2010. 8, 11

[27] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014. xi, 8, 11, 15, 20

[28] C. Tsilopoulos, G. Xylomenos, and Y. Thomas, "Reducing forwarding state in content-centric networks with semi-stateless forwarding," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 2067–2075. 8

[29] The convergence project. [Online]. Available:. [Online]. Available: http://www.ict-convergence.eu 8

[30] Sail. [Online]. Available:. [Online]. Available: http://www.sail-project.eu 8

[31] Comet. [Online]. Available:. [Online]. Available: http://www.comet-project.org 8

[32] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, "Design considerations for a network of information," in *Proceedings of the 2008 ACM CoNEXT Conference*, 2008, pp. 1–6. 8

[33] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233–245, 2012. 9

[34] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 60–67, 2012. 9

[35] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in *International Conference on Broadband Communications, Networks and Systems*. Springer, 2010, pp. 1–13. 9

[36] A. Kerrouche, "Routing named data in information-centric networks," Ph.D. dissertation, Université Paris-Est, 2017. 10, 15, 24, 25

[37] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012. 10

[38] Z. I. Kiss, Z. A. Polgar, C. Vinti, M. Varga, A. B. Rus, and V. Dobrota, "Network coding-based congestion control at network layer: Protocol design and evaluation," *International Journal of Computer Networks and Communications*, vol. 3, no. 1, pp. 119–138, 2011. 11

[39] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12. xi, 14, 15, 16, 17, 23

[40] Domain names - implementation and specification. [Online]. Available:. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1035 16

[41] H. Yan, D. Gao, W. Su, and H.-C. Chao, "A forwarding strategy of counteracting redundancy data in named data networking," *International Journal of Communication Systems*, vol. 28, no. 18, pp. 2289–2310, 2015.

[42] H. Qian, R. Ravindran, G.-Q. Wang, and D. Medhi, "Probability-based adaptive forwarding strategy in named data networking," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 1094–1101. 16

[43] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013. xi, 19, 22, 23, 24

[44] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *ACM SIGCOMM computer communication review*, vol. 42, no. 3, pp. 62–67, 2012. 23, 24

[45] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," in *Proceedings of the 1st ACM conference on information-centric networking*, 2014, pp. 27–36. 24

[46] V. Jacobson, J. Burke, L. Zhang, T. Abdelzaher, B. Zhang, P. Crowley, J. A. Halderman, C. Papadopoulos, and L. Wang, "Named data networking next phase (ndn-np) project may 2015-april 2016 annual report," *Named Data Networking (NDN)*, 2016. xi, 26

[47] Y. Zhang, J. Liu, T. Huang, J. Chen, and Y. Liu, "Multi-path interests routing scheme for multi-path data transfer in content centric networking," 2013. 26

[48] D. H. Lee, K. Thar, D. Kim, and C. S. Hong, "Efficient parallel multi-path interest forwarding for mobile user in ccn," in *2016 international conference on information networking (ICOIN)*. IEEE, 2016, pp. 390–394.

[49] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, vol. 2011, pp. 1–6, 2011.

[50] A. Udugama, S. Palipana, and C. Goerg, "Analytical characterisation of multi-path content delivery in content centric networks," in *2013 Conference on Future Internet Communications (CFIC)*. IEEE, 2013, pp. 1–7.

[51] X. Hu, S. Zheng, G. Zhang, L. Zhao, G. Cheng, J. Gong, and R. Li, "An on-demand off-path cache exploration based multipath forwarding strategy," *Computer Networks*, vol. 166, pp. 107 032–107 050, 2020.

[52] A. Udugama, X. Zhang, K. Kuladinithi, and C. Goerg, "An on-demand multi-path interest forwarding strategy for content retrievals in ccn," in *2014 IEEE network operations and management symposium (NOMS)*. IEEE, 2014, pp. 1–6.

[53] G. Zhang, H. Li, T. Zhang, D. Li, and L. Xu, "A multi-path forwarding strategy for content-centric networking," in *2015 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2015, pp. 1–6.

[54] S. Shi, J. Li, B. Han, H. Wu, Y. Ma, Q. Dong, and H. D. Schotten, "Multi-path forwarding strategy for named data networking based on pending interests and available bandwidth," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 731–737. 26

[55] A. Kerrouche, M. R. Senouci, and A. Mellouk, "Qos-fs: A new forwarding strategy with qos for routing in named data networking," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7. 26

[56] K. Lei, J. Wang, and J. Yuan, "An entropy-based probabilistic forwarding strategy in named data networking," in *2015 IEEE international conference on communications (ICC)*. IEEE, 2015, pp. 5665–5671. 26

[57] P. E. V. C. E. R. S. A. Kreutz, F. M. V. Ramos and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015. xi, 29

[58] Y. Liu and H. Wadekar, "Sdar: Software defined intra-domain routing in named data networks," *Proc. IEEE 15th Int. Symp. Network Computing and Applications (NCA)*, pp. 158–161, 2016. 30

[59] T. S. E. Aubry and I. Chrisment, "Srsc: Sdn-based routing scheme for ccn," *Proc. 1st IEEE Conf. Network Softwarization (NetSoft)*, pp. 1–5, 2015. 30

[60] K. Schneider and B. Zhang, "How to establish loop-free multipath routes in named data networking," in *NDN, Tech. Rep.* NDN, 2017, p. 0044. 30

[61] M. F. Z. N. Aloulou, M. Ayari and L. Saidane, "A popularity-driven controller- based routing and cooperative caching for named data networks," *Proc. 6th Int. Conf. Network of the Future (NOF)*, vol. 50, no. 5, pp. 1–5, 2015. 30

[62] R. Jmal and L. C. Fourati, "An openflow architecture for managing content-centric-network (ofam-ccn) based on popularity caching strategy," *Comput. Stand. Inter*, vol. 51, pp. 22–29, 2017.

[63] X. Z. B. Z. J. Cao, D. Pei and Y. Zhao, "Fetching popular data from the nearest replica in ndn," *Proc. 25th Int. Conf. Computer Communication and Networks (ICCCN)*, vol. 51, pp. 1–9, 2016. 30

[64] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018. 31, 32, 42

[65] T. Jiang, Q. Zhao, D. Grace, A. G. Burr, and T. Clarke, "Single-state q-learning for self-organised radio resource management in dual-hop 5g high capacity density networks," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 12, pp. 1628–1640, 2016. 31, 32

[66] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," *AAAI/IAAI*, vol. 2002, pp. 326–331, 2002. 32

[67] V. Lehman, A. Gawande, B. Zhang, L. Zhang, R. Aldecoa, D. Krioukov, and L. Wang, "An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS).* IEEE, 2016, pp. 1–10. 34

[68] D. Posch, B. Rainer, and H. Hellwagner, "Saf: Stochastic adaptive forwarding in named data networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1089–1102, 2016. 34

[69] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "Inform: a dynamic interest forwarding mechanism for information centric networking," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, 2013, pp. 9–14. 34, 36

[70] S. M. Iqbal, M. M. Hoque *et al.*, "A source-driven reinforcement learning-based data reply strategy to reduce communication overhead in named data networks (ndn)," *Cluster Computing*, pp. 1–27, 2021. 34, 58, 60

[71] J. V. Torres, I. D. Alvarenga, A. d. C. P. Pedroza, and O. C. M. Duarte, "Proposing, specifying, and validating a controller-based routing protocol for a clean-slate named-data networking," in *2016 7th International Conference on the Network of the Future (NOF)*. IEEE, 2016, pp. 1–5. 34, 35, 36

[72] J. Son, D. Kim, H. S. Kang, and C. S. Hong, "Forwarding strategy on sdn-based content centric network for efficient content delivery," in *2016 international conference on information networking (ICOIN)*. IEEE, 2016, pp. 220–225. 36

[73] T. Guesmi, A. Kalghoum, B. M. Alshammari, H. Alsaif, and A. Alzamil, "Leveraging software-defined networking approach for future information-centric networking enhancement," *Symmetry*, vol. 13, no. 3, p. 441, 2021.

[74] A. Kalghoum, S. M. Gammar, and L. A. Saidane, "Towards a novel cache replacement strategy for named data networking based on software defined networking," *Computers & Electrical Engineering*, vol. 66, pp. 98–113, 2018.

[75] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang, "Towards software defined icn based edge-cloud services," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*. IEEE, 2013, pp. 227–235.

[76] G. Wang, T. E. Ng, and A. Shaikh, "Programming your network at run-time for big data applications," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 103–108. 34

[77] E. Aubry, T. Silverston, and I. Chrismen, "Implementation and evaluation of a controller-based forwarding scheme for ndn," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2017, pp. 144–151. 35

[78] O. Ascigil, S. Rene, I. Psaras, and G. Pavlou, "On-demand routing for scalable name-based forwarding," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, 2018, pp. 67–76. 35

[79] J. V. Torres, I. D. Alvarenga, R. Boutaba, and O. C. M. B. Duarte, "Evaluating cros-ndn: a comparative performance analysis of a controller-based routing scheme for named-data networking," *Journal of Internet Services and Applications*, vol. 10, no. 1, pp. 1–24, 2019. 35

[80] J.-H. Cha, Y.-H. Han, and S.-G. Min, "Named data networking over a software-defined network using fixed-size content names," *IEICE Transactions on Communications*, vol. 99, no. 7, pp. 1455–1463, 2016. 35, 36

[81] A. Mahmood, C. Casetti, C. F. Chiasserini, P. Giaccone, and J. Härri, "Efficient caching through stateful sdn in named data networking," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 1, p. e3271, 2018. 36

[82] M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, and A. Iera, "Understanding name-based forwarding rules in software-defined named data networking," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6. 36

[83] A. L. R. Madureira, F. R. C. Araújo, G. B. Araújo, and L. N. Sampaio, "Ndn fabric: where the software-defined networking meets the content-centric model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 374–387, 2020. 36

[84] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, "Information centric networking over sdn and openflow: Architectural aspects and experiments on the ofelia testbed," *Computer Networks*, vol. 57, no. 16, pp. 3207–3221, 2013. 36

[85] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008. 36, 40

[86] F. Ansari, R. A. Rehman, and B.-S. Kim, "Csf: Controller based selective forwarding in software defined named data based manets," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6. 36

[87] S. H. Ahmed, S. H. Bouk, D. Kim, D. B. Rawat, and H. Song, "Named data networking for software defined vehicular networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 60–66, 2017. 36

[88] M. Alowish, Y. Shiraishi, Y. Takano, M. Mohri, and M. Morii, "A novel software-defined networking controlled vehicular named-data networking for trustworthy emergency data dissemination and content retrieval assisted by evolved interest packet," *International Journal of Distributed Sensor Networks*, vol. 16, no. 3, p. 1550147720909280, 2020.

[89] I. Wahid, S. Tanvir, A. Hameed, and M. Ahmad, "Software-defined networks and named data networks in vehicular ad hoc network routing: Comparative study and future directions," *Security and Communication Networks*, vol. 2022, 2022. 36

[90] A. Tariq, R. A. Rehman, and B.-S. Kim, "Epf—an efficient forwarding mechanism in sdn controller enabled named data iots," *Applied Sciences*, vol. 10, no. 21, p. 7675, 2020. 36

[91] R. K. Dudeja, A. Singh, R. S. Bali, and G. S. Aujla, "An optimal content indexing approach for named data networking in software-defined iot system," *IET Smart Cities*, vol. 4, no. 1, pp. 36–46, 2022. 36

[92] J. Lv, X. Tan, Y. Jin, and J. Zhu, "Drl-based forwarding strategy in named data networking," in *2018 37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 6493–6498. 36

[93] Y. Zhang, B. Bai, K. Xu, and K. Lei, "Ifs-rl: An intelligent forwarding strategy based on reinforcement learning in named-data networking," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 54–59.

[94] B. Fu, L. Qian, Y. Zhu, and L. Wang, "Reinforcement learning-based algorithm for efficient and adaptive forwarding in named data networking," in *2017 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2017, pp. 1–6.

[95] O. Akinwande, "Interest forwarding in named data networking using reinforcement learning," *Sensors*, vol. 18, no. 10, p. 3354, 2018. 58

[96] M. Zhang, X. Wang, T. Liu, J. Zhu, and Q. Wu, "Afsndn: A novel adaptive forwarding strategy in named data networking based on q-learning," *Peer-to-Peer Networking and Applications*, vol. 13, no. 4, pp. 1176–1184, 2020. 58

[97] N. Dutta, S. Tanwar, S. K. Patel, and G. Ghinea, "Svm-based analysis for predicting success rate of interest packets in information centric networks," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2020488, 2022.

[98] F. Abdi, M. Ahmadi, and M. Ghanem, "La-mdpf: A forwarding strategy based on learning automata and markov decision process in named data networking," *Future Generation Computer Systems*, vol. 134, pp. 22–39, 2022.

[99] N. Dutta, "A bargain game theory assisted interest packet forwarding strategy for information centric network," *Journal of Network and Computer Applications*, vol. 209, p. 103546, 2023. 36

[100] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: an open-source simulator for NDN experimentation," *ACM Computer Communication Review*, Jul. 2017. 58

[101] S. M. A. Iqbal *et al.*, "Adaptive forwarding strategies to reduce redundant interests and data in named data networks," *Journal of Network and Computer Applications*, vol. 106, pp. 33–47, 2018. 58

[102] L. Huo, "Packet-level-based traffic aggregation to optimize ndn content delivery," *International Journal of Communication Systems*, vol. 33, no. 12, p. e4473, 2020. 58

[103] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2847–2886, 2016. 58

[104] A. T. Prodhan, R. Das, H. Kabir, and G. C. Shoja, "Ttl based routing in opportunistic networks," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1660–1670, 2011. 58

[105] A. Medina, I. Matta, and J. Byers, "Brite: a flexible generator of internet topologies," 2000. 60

[106] G. Rossini and D. Rossi, "Coupling caching and forwarding: Benefits, analysis, and implementation," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 2014, pp. 127–136. 61

[107] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content centric networking (extended version)," *Research report*, 2011. 61