Delft University of Technology
Master of Science Thesis in Embedded Systems

Design and Analysis of a framework for Dynamic Selection of TCP Congestion Control Algorithms

Soovam Biswal

TU Delft Delft University of Technology

Design and Analysis of a framework for Dynamic Selection of TCP Congestion Control Algorithms

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Soovam Biswal
S.Biswal-1@student.tudelft.nl
soovamb1997@gmail.com

14 October 2021

Author
Soovam Biswal (S.Biswal-1@student.tudelft.nl)
(soovamb1997@gmail.com)

Title
Design and Analysis of a framework for Dynamic Selection of TCP Congestion Control Algorithms

MSc Presentation Date
26 October 2021

Graduation Committee

dr. ir. Fernando Kuipers (chairman)	Delft University of Technology
dr. Jan S. Rellermeyer	Delft University of Technology
Belma Turkovic	TNO

Abstract

Introduction of new, more advanced services to the networking paradigm has led to an increased heterogeneity of media types and network traffic. Although several transport protocols have been developed over the years to cater to the Quality-of-Service requirements of these network services, the dynamic nature of the network condition is a variable that creates a hindrance in the mapping of an efficient transport protocol to a network service.

Dynamic Protocol Selection can serve as a potential solution for this by applying innovative techniques to adaptively select among pre-existing transport protocols during run-time, thus catering to these requirements dynamically. Although the concept has been proven to be beneficial, there are certain research gaps that are yet to be addressed. In this thesis, we attempt to take a step forward to address a few of these research gaps. As a result, we designed a standardized conceptual framework (DPS framework) for the Dynamic Selection of various TCP congestion control algorithms (as the pre-existing transport protocols). To enable this framework to function autonomously, we developed an online learning strategy implemented in the framework design. Also, to ensure efficient deployability of the framework in a real-network, we further proposed a fairness framework to manage multiple DPS framework-enabled application flows to co-exist sustainably.

Through our experiments, we demonstrated the trade-off between application flow performance and learning time for the proposed online learning strategy and the ways it can be tuned to benefit certain types of application flows. We further presented results that showcased around 15% improvement in the fairness performance between multiple application flows and stability in the average flow performance due to the implementation of the proposed fairness framework.

"It always seems impossible until it's done." – Nelson Mandela

Preface

This thesis represents my final work as an Embedded Systems Masters student at Delft University of Technology and comprises of the research that I carried out in the past 9 months in the Embedded and Networked Systems (ENS) research group. Although the journey was not easy but it was a fulfilling experience and I am honoured to have experienced this time, although remotely, with the members of the ENS group.

I would like to thank dr. ir. Fernando Kuipers for his extensive feedback and guidance during this thesis. It was only through your constant questionnaires during the literature survey that I was able to come up with a research question for the thesis.

To Belma Turkovic, thank you for always taking out time to assist me with setting up and troubleshooting the infrastructure for implementing the proposed work and discussing about the types of experiments to conduct.

I would also like to express my gratitude to dr. Jan S. Rellermeyer for being part of my thesis committee.

Last but not the least, I would like to thank my family and friends for their constant support and encouragement in successfully completing this thesis, especially during a pandemic.

Soovam Biswal

Delft, The Netherlands
14th October 2021

Contents

Chapter 1

Introduction

The past decade of the technology era has witnessed a dramatic growth in revolution of the Internet, especially in terms of its application coverage [53]. With greater and affordable bandwidth, the traditional data-handling methods have moved from local file backups and mirrors to locations accessible over a network. Further, the development of advanced networking concepts such as 5G [4] and the Internet-of-Things [39] have enabled numerous services each demanding superior network quality. For instance, services such as immersive virtual reality [54], interactive gaming [65], or telementoring [37] have tight latency and jitter tolerances to maintain an acceptable Quality of Experience (QoE) [42]. Moreover, sensitive services, e.g. cloud robotics [50], remote surgery [58], or cooperative driving [21] have additional reliability and availability constraints.

Introducing these new services has led to increased heterogeneity of media types and traffic, thus further augmenting network complexity. Thus, catering to such a wide spectrum of Quality of Service (QoS) requirements necessitates customization of network connections for handling different traffic patterns, types of data transmitted, and types of transmission media.

1.1 Motivation

Over the years, the transport layer protocols of the network stack have undergone numerous customizations, either with a goal to improve upon earlier versions or provide additional features for incrementing the number of use-cases supported. A classic example showcasing this variability in customization is the Transmission Control Protocol (TCP) [11] and its light-weight counterpart, the User Datagram Protocol (UDP) [48]. While TCP, with its complex system architecture ensures end-to-end reliable connectivity, it suffers from high transmission delays due to large packet overheads (20 bytes). UDP, instead, addresses this limitation by forming much lower packet overheads (8 bytes) trading off the complex reliability and connectivity features. Among other variations are Stream Control Transmission Protocol (SCTP) [20] that employs datagrams as packet-entities similar to UDP but closely aligns with many of TCP's features. Additionally, this protocol also supports multi-homing [52] and redundant paths enhancing reliability and resiliency.

Apart from the general trend of designing protocols based on certain qual-

ity of service requirements, another category of formulation is solely based on their application use-case. Licklider Transmission Protocol [8], developed for space communications focuses on domain-specific requirements such as energy-efficiency and prioritized reliability of data transmission. Sensor Transmission Control Protocol (STCP) [32] aims to create a generic, scalable and reliable transport protocol for wireless sensor networks (WSN). Similarly, transport protocols, for e.g. Interactive Real-time Protocols (IRTP) [47], Efficient Transport Protocol (ETP) [64] or Hybrid Multicast Transport Protocol (HMTP) [6] are designed for haptic data transmission (communication of the sense of physical touch).

However, TCP, being the best-known and extensively implemented transport protocol, majority of the transport protocol research is focused on improving upon TCP's impairments and ensuring fairness among its variants. This argument is evident with the large number of TCP versions (flavors) [46] that currently exist, each designed to handle a particular set of network conditions. This thesis, therefore, will present its further discussions taking the variabilities among these TCP flavors into consideration.

All TCP variants basically differ in the methods they adapt to deal with network congestion observed under specific traffic patterns or transmission media. For instance, TCP's Cubic [28] or BIC [67] variants perform best when operated over high bandwidth-delay product (BDP) links. However, their performance degrades with increasing link losses, commonly encountered in wireless connections. Due to its better handling of packet losses caused by transmission errors, TCP's Westwood variant [43] is generally a preferred choice for this environment type. Moreover, as most of these variants depend on packet loss as a metric for congestion indication, the packets inherently suffer from large round-trip times (RTTs). This issue is addressed with variants such as Vegas [7] or Lola [29], that depend solely on RTT as their congestion indication metric. However, from the current trends of Internet evolution, it can be inferred that it is indeed difficult to estimate the number of possible application environments. Therefore, adopting traditional methods to design protocols catering to a certain network condition set or application requirement is an inefficient technique.

Application of learning techniques for congestion detection and control [34] is a research area that serves a potential solution to this inefficiency. Remy [63] designs a congestion control (CC) algorithm generated offline by mapping (learning) congestion parameters to performance metrics via optimized state-space exploration. However, it suffers from low-fairness in heterogeneous networks and inherently leads to performance degradation on deviation from the actual input conditions. PCC Vivace [18] is another rate-control algorithm that leverages the concept of online (convex) optimization in machine learning to achieve superior performance to legacy TCP variants. Several algorithms [57, 31, 66] also leverage the concept of re-inforcement learning (RL) to optimize specific performance metrics, e.g. packet loss, re-transmissions, jitter, latency or throughput. However, these algorithms generally exhibit high computational complexity, memory, and convergence time. An inherent issue with these learning-based CC algorithms is incompatibility with current technology. For instance, most of the recent Linux versions provide a choice between about ten TCP variants. As most of these algorithms are tested in limited environments, further extensive research is required to resolve compatibility issues with the traditional CC algorithms.

Dynamic Protocol Selection [9] is yet another solution that has recently been a focus of research. It basically involves applying innovative techniques to adaptively select among pre-existing transport protocols (for instance, TCP variants offered by the Linux kernel) based on comparing these candidate protocols' performances. This solution inherently resolves the previously mentioned compatibility issue and can be easily extended to include any future protocol versions. Also, due to the offered flexibility in implementing protocols based on network and application requirements, compared to their static counterpart, a significant performance gain is achieved [9, 49, 51, 62]. However, as this direction of research is in its early stages, several issues still persist. Research works are mostly focused on few key areas:

- Demonstrating improved performance offered by dynamic protocol selection in different application domains such as wireless sensor networks [49], reliable multicast [51]

- Schemes aimed at efficiently selecting a suitable transport protocol from an available set of choices [22, 23, 35]

While the mentioned research goals are promising, they do not consider the practical issues involved in implementing this technique over existing networking solutions. To successfully identify and develop solutions for such problems, an end-to-end run-time analysis of the dynamic selection framework is essential.

1.2 Problem Definition

In this report, we aim to investigate ways to design and analyze a dynamic protocol selection framework to be implemented in the Linux kernel utilizing its available TCP variants as the default protocol set for the framework. This approach would help in better understanding of essential design parameters and identifying key design issues of a dynamic protocol selection framework.

Some problems need to be addressed before designing such a framework. To initiate protocol selection in a dynamic selection framework, some knowledge about the candidate protocols' performance needs to be available to make a decision based on comparison. To implement this, the present works [49, 51, 62, 35] assume the framework's protocol set to be (pre) partially-trained with their corresponding characteristic statistics before implementing to ensure lower selection time after deployment. However, this initial training step would hinder the framework's possibility to autonomously operate in changing environments. Addition of novel protocols to the framework would also suffer from a similar issue as they would need some amount of training before deployment. Hence, it is essential to consider designing a scheme implemented into the framework architecture that would bypass this additional training step and simultaneously learn about the candidate protocols while applying them to govern application flows.

Further, during the framework run-time, when a new transport protocol, different from the previously implemented one is selected, the framework needs to switch the currently applied protocol with the new one. A good framework should ensure negligible impact on the application flow's performance while

switching. As the framework's transmission and reception modules' logic directly translates to the switching methodology, designing them in the framework architecture should be given special attention.

Moreover, on deploying any application flow in a network, it should be ensured that the flow consumes network resources optimally and allows an equal share of the available resources to existing flows. Fairness is a key performance metric that quantifies this flow behaviour. As a typical dynamic protocol selection framework involves implementing multiple transport protocols during an application flow's lifetime, each of the candidate protocols might have different fairness behaviors that would affect the effective fairness behavior of the application flow. It is, therefore, important to also evaluate the framework's fairness performance.

These discussions can be summarized into the following research question:

> *How can we design an efficient framework for dynamic selection of transport protocols?*

This can be broken down into further sub-questions for a better idea about this thesis' research goals:

1. How can we design the framework to learn online with optimum efficiency and accuracy?

2. How can we design transmission and reception components for the framework for efficient protocol switching during run-time?

3. How can we ensure fairness among application flows running over the dynamic protocol selection framework?

1.3 Contributions

Our key contributions are as follows:

1. We proposed a conceptual Dynamic Protocol Selection (DPS) framework based on the design principles of a portfolio scheduler (Section 2.3) so as to standardize the framework.

2. We designed a strategy to facilitate online learning in the proposed DPS framework.

3. To maintain an acceptable fairness among multiple application flows governed by the proposed DPS framework, we further proposed a Fairness Framework for guiding the deployed flows.

4. We evaluated and analysed the performance of the proposed learning strategy and fairness framework.

1.4 Thesis Outline

The rest of the thesis is organized as follows: Chapter 2 provides brief descriptions of some concepts that are essential to understand the work performed in the thesis. Further, to highlight the research gaps and our motivation to develop the Dynamic Protocol Selection framework, Chapter 3 summarizes the existing literature on the concept. Finally, Chapter 4 provides a detailed description of our contributions in the thesis. The proposed work is evaluated and analysed in Chapter 5. To wrap up the report, a brief summary of our entire work in the thesis and some potential future improvements in the proposed frameworks are described in the final Chapter 6.

Chapter 2

Background

To gather a clear understanding of our work in this thesis, this chapter lays emphasis on describing the fundamental concepts of several elements that form the building blocks of the work.

2.1 Transmission Control Protocol (TCP)

The transport layer of the Internet Protocol Suite (IP-Suite) [13] or Open Systems Interconnection model (OSI) [15] network stacks provides host-to-host connectivity with its end-to-end message delivery services that are independent of the functionalities of the lower layers. With its location between the application layer at the top and network (IP) layer at the bottom, this layer encapsulates data received from the application layer into data-blocks called segments and forwards them to the network layer as IP-payloads. Moreover, to distinguish between several process flows of applications running in a machine, the layer introduces the concept of network ports, an operating-system allocated number for each communication channel required by an application.

Currently, several protocols [48, 11, 20, 27] exist that implement the basic functionalities of the transport layer along with some additional protocol-specific features. Among these, the Transmission Control Protocol (TCP) [11] is the most widely used. TCP is a connection-oriented protocol i.e., data transfer, using TCP, is only possible after establishing a dedicated connection to the receiver. This feature is just the first step among several others that TCP implements to ensure reliability in data transmission. Some of the key functionalities that TCP provides are as follows:

- Lost packet re-transmission
- Ordered data transfer
- Error Detection
- Flow Control
- Congestion Control

Due to the complex functionalities supported by the protocol, a typical TCP segment has a relatively large header size ranging from a minimum of 20 bytes

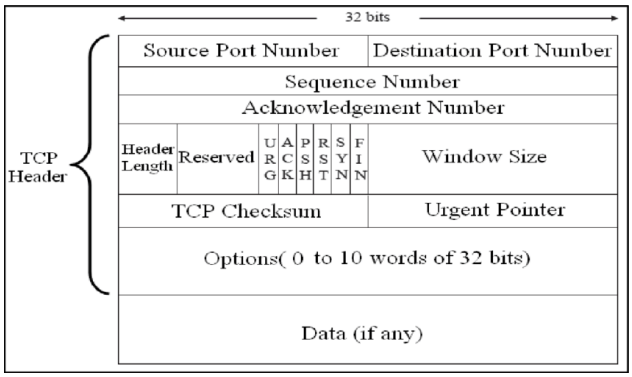


Figure 2.1: **TCP Header format. The options field has a variable length ranging from 0 to 40 bytes.**

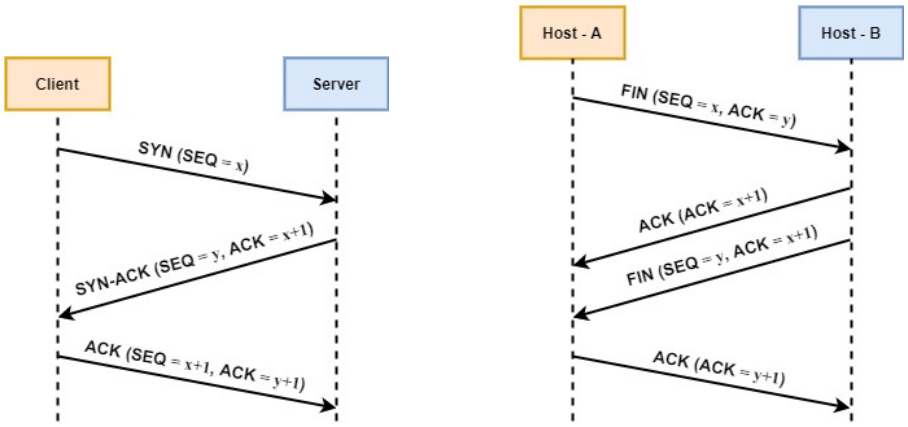
to a maximum of 60 bytes depending on the length of the options field as shown in Figure 2.1.

2.1.1 TCP Operation

The working of the protocol can be divided into three phases:

1. **Connection Establishment:** This is the first procedure that any TCP connection undergoes to establish a dedicated connection between a TCP client and server and is initiated by the client. However, for this procedure to work successfully, the connection should always be in a passive-open state i.e., the server should bind to and continuously listen at a specific network port that clients could connect to. In addition to the normal application data-carrying segment-types that TCP promises to transfer, the protocol also introduces segment-types specifically for establishing TCP connection: SYN, SYN-ACK and ACK (indicated by the specific TCP flags in the header shown in Figure 2.1). These segments do not carry any application data. Moreover, the sequential communication of these special segments is called three-way handshake that alters the passive-opened TCP connection state to active-open. As indicated in Figure 2.2a, the handshake begins with the client sending a SYN (synchronization) packet to the server. On SYN reception, the server identifies the new connection, allocates machine resources for the connection and acknowledges it with a SYN-ACK (synchronization acknowledgement) packet. Further, on SYN-ACK reception for the transmitted SYN, the client sends an ACK (acknowledgement) packet to the server for indicating the initiation of application data-transmission shortly.

2. **Data Transfer:** During this phase, the client begins sending application-data to the server over the established connection. The server, on receiving each TCP segment (data packet), acknowledges the client with the special ACK packet. However, TCP still needs to ensure guaranteed and ordered delivery of data. It implements these features with proper monitoring of the sequence (SEQ no.) and acknowledgement (ACK no.) numbers of the



(a) **TCP Connection Initialization 3-way handshake.**

(b) **TCP Connection termination 4-way handshake.**

Figure 2.2: **TCP Connection signalling during initialization and termination phases.**

transmitted packets as indicated in the TCP segment header in Figure 2.1. Normally, after the client sends a packet with a SEQ no., the server replies with an ACK packet having an ACK no. equal to the sum of the SEQ no. and the length of the received payload. Through this ACK no., the server indicates to the client for sending a next packet with the SEQ no. equal to the ACK no. Apart from this, more complex processes such as flow and congestion control also take place during this phase to ensure good data throughput.

3. **Connection Termination:** This phase typically involves a four-way handshake and can be initiated from either of the end-points. As illustrated in Figure 2.2b, when an end-point (A) wants to terminate the connection, it sends a special TCP segment-type packet: FIN (finish). On FIN reception, the other end-point (B) acknowledges it by sending an ACK packet in response. After sending this ACK, the end-point B tries to acknowledge all the unacknowledged received data packets from A, if there is any and finally sends a FIN packet. A, on receiving the FIN, acknowledges it with an ACK sent to B. This completely terminates the connection. However, the corresponding resources reserved for the connection is freed only after a few minutes, depending on the operating system.

2.2 TCP Congestion Control

Network congestion is a condition when the incoming data to a network cannot be processed at the normal rate due to overloaded network resources. Any further incoming data only adds to the congestion and increases the load of the already overloaded network leading to a worse situation. A simple solution

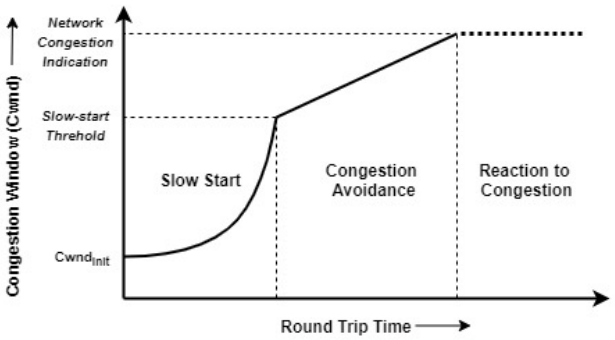


Figure 2.3: **Typical congestion window behaviour in TCP congestion control mechanism resulting in TCP congestion phases.**

for the network to recover from this congested state can be prohibiting any incoming flows into the network for a certain time duration or indicating the flows to back-off. Such practices of reacting to congestion or in some cases, taking preventive measures to avoid congestion is known as congestion control.

During the *data transfer* phase in TCP, it implements some of these congestion control solutions to tackle network congestion. Basically, the TCP sender keeps a check on the transmitting date rate by varying a metric known as Congestion Window. This window indicates the maximum number of data-bytes that can be transmitted by the sender without any interruptions, if it is lower than the advertised receiver window (maximum data bytes that can be processed by the TCP receiver). Depending on the network condition and TCP's algorithm, this congestion window can be varied in a number of ways throughout the data transfer phase.

Figure 2.3 illustrates a typical TCP congestion control scenario. A TCP connection always begins with a slow-start phase of the congestion control mechanism increasing the congestion window appropriately to find a balance between achieving high throughput and low probability of network congestion. The slow-start phase continues to operate until either a packet drop, receiver window threshold or a pre-calculated congestion window threshold (slow-start threshold) is encountered. On encountering the slow-start threshold, TCP's congestion mechanism changes phase to congestion avoidance that further decelerates the congestion window rise as a pre-cautionary measure to avoid building up large queues at network buffers. However, a loss encountered is assumed as network congestion by TCP and it varies the congestion window along with taking other appropriate measures according to the congestion control algorithm implemented by the specific TCP variant.

2.2.1 Categorizing TCP Congestion Control Algorithms

As mentioned in Chapter – 1, a large number of these TCP variants exist today, each catered to solving issues that previous versions suffered from or taking a completely new approach contrasting the current design practices. However, based on the methods employed to detect and react to network congestion, TCP congestion control algorithms can be broadly classified into four categories [60]:

- **Purely-loss based algorithms:** These algorithms depend only on packet loss to detect network congestion. As packet loss in a network mostly occurs due to excessive filling up of the network buffers, these algorithms aggressively transmit packets into the network if no packet loss is detected aiming to maximize the utilization of available resources. In-fact, these variants are the most aggressive among all categories. Well know TCP variants such as Cubic [28], Reno [1] or High-Speed (HS)-TCP [26] among many others fall into this category.

- **Delay-based algorithms:** These are the least aggressive congestion control algorithms among all categories as they only depend on RTT (Round-trip time) experienced by packets, a sensitive metric, for congestion detection. In contrast to pure-loss based algorithms, these algorithms significantly reduce the chances of filling up of network buffers. They achieve this by reducing the packet transmission rate into the network on detection of packets experiencing higher RTT than a specific threshold that mainly occurs due to higher waiting times at network buffers with long packet queues. Variants such as Vegas [7] and Lola [29] fall into this category.

- **Loss-delay based algorithms:** As the name suggests, these algorithms take a middle-ground between the two aforementioned categories. They still employ packet loss as a metric to detect network congestion. However, the calculations for varying the congestion window as a reactive measure to the detected congestion depends on the delay (RTT) experienced by the packets. Therefore, although these are less aggressive algorithms than pure-loss based ones, they still require filling up the network buffers to detect congestion. Variants such as TCP Illinois [40], Westwood [43] and TCP-Compound [56] adopt such mechanisms.

- **Model-based hybrid algorithms:** These algorithms take a fundamentally different approach for congestion control. Instead of relying on packet loss or RTT as metrics for reactive measure, the algorithm tries to estimate a model of the network that describes its present state and thus helps the TCP sender to proactively modify the congestion window (sending rate). TCP BBR [10] is a recent variant that tries to implement this mechanism by periodically monitoring the network to update its estimated model.

2.3 Portfolio Scheduling

The basic idea of Portfolio Scheduling was proposed by Huberman et. al. [30]. A portfolio scheduler dynamically selects among a set of policies based on user-defined rules and feedback mechanisms. Through this combination of policies, the scheduler becomes more flexible and reactive to varying application requirements compared to the constituent policies considered in isolation. Apart from its extensive application in finance [25] over the years, it has also been implemented for scheduling workloads of varying natures in data centers [17, 61, 41].

Deng et al. [17] describes four basic stages of a portfolio scheduler as illustrated in Figure 2.4:

- ***Creation:*** This stage involves creating or preparing the set of policies to be included in the portfolio scheduler. While including more policies en-

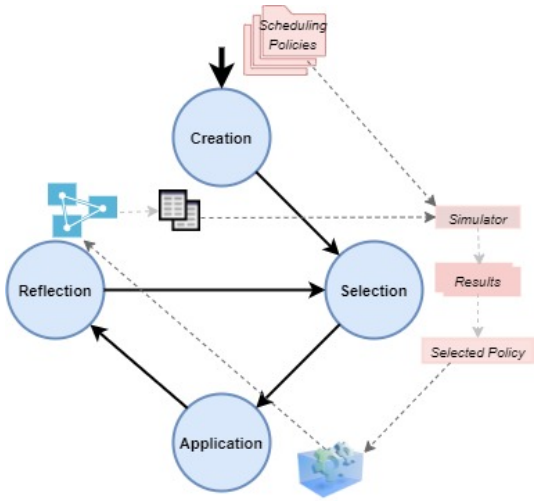


Figure 2.4: **Finite-state machine of the four stages of a portfolio scheduler used to schedule workloads in data centers.**

hances diversity, it necessitates more sophisticated algorithms to correctly select among the wide range of choices.

- ***Selection:*** This stage involves selecting a single policy from the policy set. The selection is generally based on the performance of corresponding policies measured in a simulated environment taken as input for a portfolio selection methodology (for instance, utility-based selection).
- ***Application:*** In this stage, the policy chosen in the selection stage is implemented in the real application environment to perform tasks.
- ***Reflection:*** The performance of the implemented policy is analyzed in this stage. The analysis report can serve as a basis for further improvement of the selection criteria or tune policy parameters.

Apart from the creation stage which is processed once in the life-cycle of the scheduler, all other stages are processed periodically.

Chapter 3

Related Work

As mentioned in Chapter 1, the research on the concept of Dynamic Protocol Selection is mainly focused on either showcasing the performance improvement achieved due to dynamic selection in several application areas or investigating methodologies for efficient selection of protocols catering to the service level requirements (SLRs) and network conditions.

In an effort to achieve better performance in MANETs, Rosenfeld et al. [49] propose a technique to intelligently select protocols in the transport and session layers based on the measured network conditions. The method utilizes the concept of Markov Random Fields (MRF) to dynamically assign labels to the network based on observation of certain network parameters. These labels keep track of the performance of the cross-layer protocol combination and the best performing combination is selected on encountering a similar network label. Although the work establishes a direct relation between performance gain and dynamic protocol selection, the analysis, however looks more into domain specific metrics such as average time for all nodes of the MANET to agree on a protocol (Average Delay to Agreement) or percentage of nodes realizing the classified network state (Belief Percentage). These metrics do not provide insights into the implementational or architectural aspect of the selection framework.

A similar issue persists in other works [51, 44]. In [51], Shibata et al. focus on improving multicast communication efficiency using the dynamic protocol selection concept. They propose a framework to dynamically select among multicast protocols by periodically monitoring the network based on the acquired receiver reports and derived scale of the multicast group based on the topology information. In addition to the demonstrated performance improvement, the authors briefly address the inherent problem with protocol switching in the proposed protocol selection framework and the overhead thus incurred. However, unlike our work, it fails to provide a thorough analysis into the switching overhead problem.

Enhancing flexibility and efficiency in Multi Agent Systems (MAS) [19] is yet another application area that benefits from protocol selection. This is showcased by Mehmood et al. [44] who propose a framework that enables an agent to directly upload a suitable transport protocol at run-time, if it is authorized by an adaptive behavior entity in the framework. The entity keeps track of the frequency of requests for several transport protocols from agents and only authorizes those protocol requests that are among the top three in terms of

request frequencies. This adaptability leads to a better resource usage in the agents as it avoids loading all transport protocols in their memory at compile time.

Another direction of research focuses on designing efficient architectures for dynamic transport protocol selection framework. These works generally aim at designing functional modules and suitable protocols for end-to-end framework deployment. Nakajima [45] proposes an architecture for protocol selection that is exclusively designed for Cobra [2] networks. Three design issues are considered in the work that forms the basis of the proposed architecture. The first issue concerns the ways in which transport protocols can be presented to the user. The next issue focuses on the permissible level of flexibility allowed for protocol selection. And the final issue concerns ways to handle exceptional situations such as connection setup failure.

Quality of Transport (QoT) [38] is another architecture proposed at the interface of transport and session layers of the OSI model [15]. It aims to autonomously select among multiple transports in heterogeneous wireless environments. QoT is comprised of sophisticated functional blocks for carrying out transport and service discovery, object exchange, transport switching and transport selection. Further, specific protocols are introduced for each functional block to realize their functionalities. For instance, a multi-transport discovery algorithm is implemented in the transport discovery block that enables devices (communicating end hosts) to discover common transport methods.

NEAT [36] is a user-space library that aims at reducing transport layer ossification [24] by decoupling the applications from the underlying transport protocols. Although its core design concept closely aligns the above-mentioned architectures, it differs in its implementation of the various functional blocks and transport selection strategy. NEAT's architecture includes a policy manager that ranks feasible transport protocols according to the destination information stored in its policy (PIB) and characteristic information base (CIB), that further undergo selection according to Happy Eyeballs algorithm [14].

Although the discussed dynamic selection frameworks aim to improve the framework functionalities to enhance their feasibility to be deployed in real-world networks, they can never be fully deployable without proper analysis of the framework performance at an implementation level, which is the key goal of our work. Moreover, the framework architectures have a general assumption that the available protocol set either has an initial knowledge about the protocols' characteristics [38, 45] or they undergo an initial training to learn about those characteristics [36]. However, to be truly self-sufficient, the frameworks should have a mechanism to autonomously learn about the protocol behaviors in any network environment they are deployed, a concept which is further discussed in our work.

The benefits obtained from dynamic protocol selection is largely dependent on the process of selecting appropriate protocols at appropriate circumstances during framework run-time, catering closely to the application requirements and network condition. Proper decision making for selecting the right protocol, therefore is an essential task in a dynamic selection framework and hence another direction of research. Chan et al. [12] propose a two-stage decision-making algorithm for selecting among transport segments (GPRS, UMTS, Satellite, etc.) based on their satisfaction of multiple objectives. The first stage fuzzifies the system and network level information (measurements) and uses Analytic

Hierarchy Process (AHP) [55] for weighing application (user) requirements. In the next stage, a decision making methodology introduced in [5] is performed over the outputs from the first stage to finally select a specific segment.

Focusing on the transport selection functional block of the QoT architecture [38], two decision-making approaches are introduced in [23] and [22] respectively. In [23], Duffin et al. propose Prioritized Soft Constrained Satisfaction (PSCS), an approach that involves selecting a suitable transport method based on user-defined preferences and priorities for transport characteristics. It achieves this by creating a binary tree with characteristic performance of transport methods as nodes and corresponding characteristic's user-defined priority as hierarchy. A decision is finally made after traversing the tree using depth-first-search [3] and reaching a single node (transport method).

In contrast to PSCS, a quantitative approach is proposed in [22]. In this approach, the measured value of a specific transport characteristic is assigned a utility based on the user-defined preferences of the corresponding characteristic. The net-utility per transport method is calculated as a weighted sum of the utilities of its individual characteristics. And a transport method with the highest calculated net utility is finally selected. As this approach has better alignment to realistic scenarios and has low complexity, a slightly modified version of this selection method is employed in the protocol selection stage of the framework implemented in our work.

Further, as our work focuses on looking at implementation of the conceptual design of a dynamic protocol selection framework, in contrast to the architectural design works of the framework discussed previously, and makes use of the available TCP congestion control algorithms as framework protocol set to analyze the performance of dynamic protocol selection framework, it can be considered as a closely related or extended version of the works in [9, 62, 35]. In [35], Johnson et al. propose a method to select among transport protocols based on the output from a Bayesian network, a directed acyclic graph comprising of nodes that represent protocol characteristics or variables that affect protocol selection. The conditional probabilities of the variables' states stored in these nodes are regularly updated with the most recent observations so as to enhance protocol selection accuracy. Selection is based on those transport protocols that report highest conditional probabilities for the preferred states in the leaf nodes of the network. However, as selection is based on a certain number of pre-defined states, it leads to a lower resolution for selection. Moreover, this method also assumes an initial training of the proposed framework to gather protocol statistics, which is an issue addressed in our work.

A similar issue occurs in ADYTIA [62], a dynamic selection framework that considers TCP congestion control algorithms as the framework's protocol set. It assumes a pre-defined mapping between network conditions (represented by certain network parameters), application type and preferred protocol to be present in its database before its deployment. At run-time, the framework measures the network conditions and selects appropriate protocols according to given application type. To summarize, the work mainly focuses on demonstrating the impact of dynamic selection on communication performance. Fortunately, the authors also take a step towards addressing the issue of unfairness that is inevitable with multiple flows of the framework, but fail to propose a solution for it. Our work, therefore, drives this discussion forward and approaches to solve it.

The works of Caini et al. in [9] is one of the first to analyze the performance

improvement achieved through dynamic protocol selection, specifically taking TCP congestion control algorithms as the protocol set similar to our work. It further proposes some brief ideas on feasibility and implementation of the selection framework in the existing network infrastructure. This literature, therefore can be considered a major motivation to our work in this thesis.

Table 3.1 presents a brief summary of the available research works in the domain of Dynamic Protocol Selection and thus highlights the research gaps that this thesis tries to address.

Category of Work	Literature	Research Areas Covered in the Literature				
		Application Area	***Selection Strategy Adopted***	***Online Learning Analysis***	***Switching Analysis***	***Fairness Analysis***
Application Domain Based DPSF[a]	*Rosenfeld [49]*	Ad-Hoc Networks	Network-label to Protocol Mapping	No	No	No
	Shibata [51]	Multicast Networks	Network-state to Protocol Mapping	No	Partial	No
	Mehmood [44]	Multi-Agent Systems Communication	Protocol request frequency	No	No	No
DPSF[a] Architecture Design	*Nakajima [45]*	COBRA Networks	Generalized[d]	No	No	No
	Knutson [38]	Heterogeneous Wireless Environments[b]	PSCS Algorithm	No	No	No
	Khademi [36]	Generalized[c]	Happy Eyeballs Algorithm	No	No	No
Decision Making Schemes	*Chan [12]*	Heterogeneous Mobile Environments[b]	Fuzzy Multiple Objective Decision-making Algorithm	Not Applicable	No	Not Applicable
	Duffin [23]	Heterogeneous Wireless Environments[b]	Depth-first Search Algorithm in a binary-tree	Not Applicable	No	Not Applicable
	Duan [22]	Heterogeneous Wireless Environments[b]	Overall parameters' utility	Not Applicable	No	Not Applicable
DPSF[a] Conceptual Design	*Caini [9]*	TCP based Networks	Not Applicable	No	No	No
	Johnson [35]	Generalized[c]	State's conditional probability in a Bayesian Network	No	No	No
	Vanzara [62]	TCP based Networks	Network-state to Protocol Mapping	No	No	No
	Our work	TCP based Networks	Overall parameters' statistics' utility	Yes	Yes	Yes

[a]Dynamic Protocol Selection Framework.
[b]The work can be extended to accommodate other environments.
[c]No application area is specified and hence, can be applied with any application use case.
[d]The architecture provides library to declare selection policies according to application use cases.

Table 3.1: **An overall summary of the existing literature describing the range of research work performed in the domain of Dynamic Protocol Selection.**

Chapter 4

Framework

To address the research gaps in the domain of Dynamic Protocol Selection (DPS) as can be inferred from Table-3.1, this chapter follows a systematic approach. In Section 4.1, it starts with defining a few high level requirements of a basic DPS framework and goes on to address each of those requirements in the following sections. Basically, it proposes a standardized model for the conceptual DPS framework design (Section 4.2) and provides a deeper understanding of its constituent components (Section 4.3) and operational flow (Section 4.4). Additionally, it also proposes a strategy for autonomous learning of the DPS framework (Section 4.5) and a framework for addressing the fairness issue (Section 4.6) pointed out in Chapter 3.

4.1 Framework Requirements

This section provides a list of requirements that should be met for designing an efficient framework for protocol selection.

- **RQ1:** The conceptual design of the dynamic protocol selection framework should have features that enable the framework to be standardized.
- **RQ2:** The framework should be truly autonomous, i.e., it should have the capability to operate without any external training in any network environment it is deployed.
- **RQ3:** The candidate protocols within the framework should be able to switch at framework run-time with minimum degradation of the application flow's performance.
- **RQ4:** Multiple flows deployed with the framework should demonstrate acceptable fairness.

4.2 Standardizing Dynamic Protocol Selection Framework Design

From Chapter 3, it can be inferred that previous works on DPS framework design proposals have either been inconsistent or vague with applying and presenting

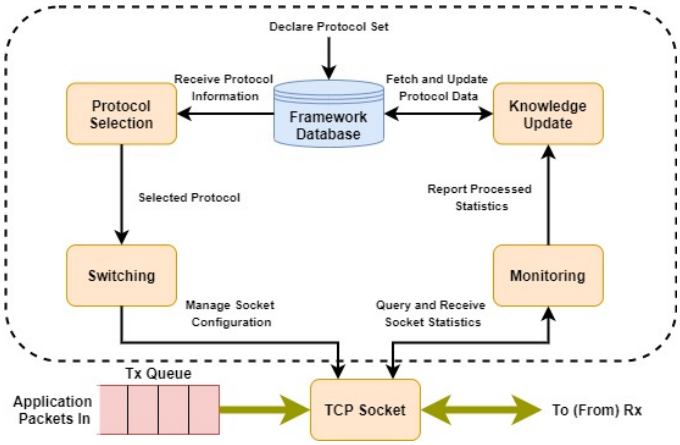
their functionalities and implementations respectively. Two essential and basic components: *a finite framework protocol set* and *a protocol selection stage* are the most common and hence, the only consistent components of those proposed DPS frameworks. Although attaining a performance gain is possible with these basic components in the DPS framework, in practice, these solutions are prone to some issues such as availability and fairness, thus making them difficult to implement.

In an attempt to remove these inconsistencies, we adopt the concept of *portfolio scheduling* into designing our DPS framework. The four key stages of a portfolio scheduler stated in Section 2.3 align closely with the basic components of a DPS framework (i.e., finite protocol set (creation stage) and protocol selection stage (selection stage)). Apart from this, it also comprises of additional stages (application and reflection stages) that could potentially cater to the framework requirements **RQ2** and **RQ3**. Moreover, this concept has demonstrated promising performance gains in the field of scheduling data center workloads. Thus, basing our DPS framework design on the design principles of a well-tested and implemented concept such that of a portfolio scheduler would ensure standardization of the DPS framework along with preservation of the fundamental functionalities of the DPS concepts **(RQ1)**.

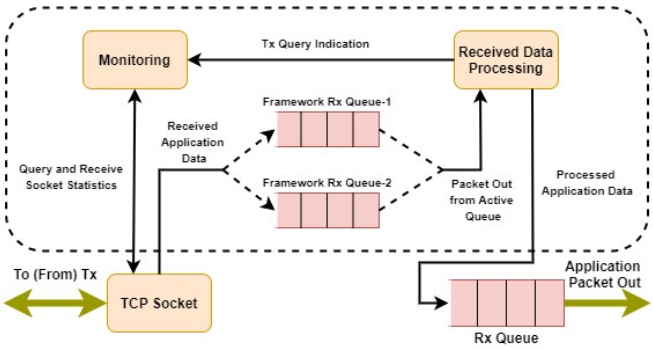
4.2.1 Modifying the selection stage design principles

The *selection* stage of a typical portfolio scheduler involves choosing a particular candidate policy based on its performance obtained from simulations. However, adding a simulation step comes at a price. Although simulating each policy provides an approximate policy performance information aiding better policy selection, the time complexity involved in simulating all the policies increases with the addition of candidate policies, leading to an overall high selection time. In [16], Deng et. al. propose a rank-based approach to simulating policies in a portfolio scheduler, where they continuously categorize policies based on their simulation performance during the scheduler run-time and select among the "best" category policies. Although this solution leads to an eventual simulation of all policies, the simulation step itself adds an extra time and computational overhead to the selection stage.

Apart from the general drawbacks of including a simulation step, designing one for our DPS framework would pose additional challenges. To roughly replicate the actual network the framework is deployed in, at the very least, a frequently updated overview of the network state and topology information should be available at the simulation step, along with other minute network statistics to further increase simulation performance accuracy. However, frequent update of these information would require additional (overhead) communication signals, thus adding to the network congestion. Moreover, this would also increase the design complexity of the framework. Therefore, to circumvent these issues, we neglect the simulation step in the selection stage of our adopted portfolio scheduler design and deploy policies purely based on their performance in the real network. To remove ambiguity, this modified version of the portfolio scheduler is addressed as "modified portfolio scheduler" in further discussions.



(a) **DPS Framework (main) at the sender based on the modified version of portfolio scheduler.**



(b) **DPS Framework (supporting) at the receiver implementing a receive buffer switching mechanism to ensure received data integrity.**

Figure 4.1: **A schematic diagram of the Dynamic Protocol Selection (DPS) framework implemented at the sender (Tx) and receiver (Rx) end-points of a connection.**

4.3 Framework Components

Based on the design principles of the modified portfolio scheduler, Figure 4.1 represents a schematic view of the designed DPS framework. Although parts of the framework are present at both the connection end-points, the sender side includes the main DPS framework (Figure 4.1a) that is based on the modified portfolio scheduler and the receiver side includes the supporting DPS framework (Figure 4.1b) that enables certain features at the receiver necessary for desired functionality of the main DPS framework at the sender. The sender, being the initiator of a typical TCP connection, has greater control and accessibility over the transmitting data than the receiver. Therefore it is considered a better location for the main DPS framework's placement that demands such functionalities.

The following discussions describe different components of the main DPS

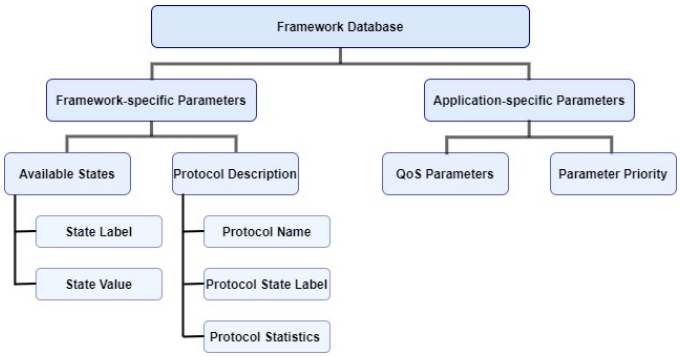


Figure 4.2: **An overview of the stored fields in the DPS Framework database.**

framework and their dependency on the supporting DPS framework.

4.3.1 Framework Database

The framework database is the storage space for framework and application specific information as illustrated in Figure 4.2. This is where the framework gathers knowledge about the protocols and network. The framework-specific information includes two types of mappings: network state mapping and protocol mapping. In the network state mapping, if the framework is equipped with a mechanism that allows it to obtain current network state, the database has provisions to record these states by identifying them as specific *State Labels* mapped to the data represented by them in the *State Value* field. In protocol mapping, the framework initially includes a list of protocols, identified either by their name or a unique identity number (*Protocol Name*), that comprise the candidate protocol set of the selection framework and hence, reflects the **"creation"** stage of the modified portfolio scheduler. The framework, upon operating, updates this protocol list with protocol statistics (throughput, delay, packet loss, etc) that can be stored in specific formats (parametric, raw, compressed, etc) in a protocol-protocol statistics mapped *Protocol Statistics* field. To represent the protocol characteristics under a certain network condition, the framework also keeps a record of the aforementioned network state label during which it collects these protocol statistics, in a similarly mapped *Protocol State Label* field.

On the other hand, the application-specific information part of the framework database provides an interface for the application flows to indicate their preferred quality of service (QoS) by providing a range (single-value) for various network parameters (latency, throughput, jitter, etc) to the framework. In our DPS framework, this application-provided QoS is represented as a set of four values ($[A, B, C, D]$), ranging from high (low) to low (high) utilities as illustrated in Figure 4.3. This method is chosen over single valued-QoS requirements as it better reflects the true application constraints. An additional priority information (0-1) per stated QoS parameter (that defaults to equal priority for all parameters) can also be indicated by the application flows for the framework to prioritize specific parameters over others during protocol selection.

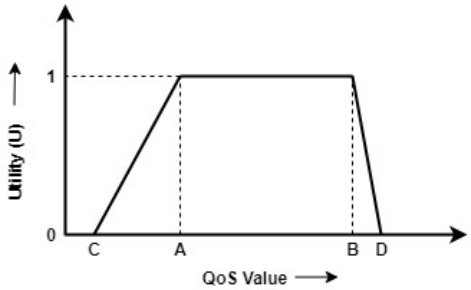


Figure 4.3: **Description of the syntax of QoS requirement to be provided to the framework by the application: [A, B, C, D]. The QoS value range (A - B) is the desired protocol's parameter's performance with the highest utility of 1. The utility decreases linearly until a low (C) or high (D) threshold value is reached, beyond which the performance levels are unacceptable to the application.**

4.3.2 Protocol Selection

This component of the DPS framework reflects the **"selection"** stage of the modified portfolio scheduler. An efficient protocol selection methodology aims for finding out the best performing protocol (high selection accuracy) within a short time duration (low processing time). Previously, this statement served as a convincing argument for eliminating the simulation step from the selection stage of the traditional portfolio scheduler. For the DPS framework, to reduce time and computational complexity in the selection component, a simple utility-based selection criteria is adopted for choosing among the candidate protocols.

However, for enhancing selection accuracy, it is equally important to decide on what protocol parameters to include in the utility calculations and the ways in which those parameters' statistics should be represented or used in the calculations so as to effectively reflect protocol behavior in the network. Firstly, it can be expected that as the number of protocol parameters increases, the selection accuracy would generally tend to improve because an increase in the parameter set provides a better reflection of the protocol behavior and highlights significant differences among protocols' characteristics. Therefore, without further investigation in this aspect, for simplicity, two protocol parameters: *Round-trip time (RTT)* and *Throughput* are considered in our DPS framework.

Next, regarding the representation of the protocols' parameters, a straightforward approach could be to create a distribution of the measured parameter values through a histogram. The resolution of this distribution would depend on the number of histogram intervals (bins)[1]. However, storing the entire histogram in the framework database would lead to a large memory consumption. A better representation would be fitting the distribution to one of the existing parametric distribution models and storing only the fixed set of parameters of the fitted model in the framework database. The specific protocol parameters' distribution can then be approximated from their corresponding fitted-parameters during utility calculations. In an effort to achieve this, we tried fitting and

[1]For achieving comparable resolution to a narrow distribution, a larger number of bins is required for a wide distribution.

Algorithm 1: Pseudo Code for Protocol Selection Algorithm.

Input : Protocol Set with Description, P_D
Output: Selected Protocol, P_{sel}

for *each candidate protocol* P *in* P_D **do**
  Calculate $U_{stat} \leftarrow \sum_{i=1}^{N} (u_{ij}.d_{ij})$;
  if $(U_{stat} > thres_{stat}) \; \forall$ *Parameters* **then**
    $Valid\_Set \leftarrow Valid\_Set \cup P$; ▷ app. constraint satisfied
  end
end
for P *in* $Valid\_Set$ **do**
  Calculate U_{net} ; ▷ calc. net utility
  $U_{temp} \leftarrow U_{temp} \cup U_{net}$
end
$U_{max} \leftarrow \text{Max}(U_{temp})$;
if U_{max} *occurs* > 1 *in* U_{temp} **then**
  for *all occurrences* **do**
    Calculate $\hat{U}_{net}$; ▷ calc. net mean utility
    $\hat{U}_{temp} \leftarrow \hat{U}_{temp} \cup \hat{U}_{net}$;
  end
  $\hat{U}_{max} \leftarrow \text{Max}(\hat{U}_{temp})$;
  if $\hat{U}_{max}$ *occurs* > 1 *in* $\hat{U}_{temp}$ **then**
    $P_{sel} \leftarrow$ corresponding Random($\hat{U}_{max}$ occurrences)
  else
    $P_{sel} \leftarrow$ corresponding $\hat{U}_{max}$;
  end
else
  $P_{sel} \leftarrow$ corresponding U_{max};
end

estimating these parameters for the measured RTT and throughput obtained with some TCP variants (Appendix A). However, the resulting fits were not consistent i.e., different parameters for different TCP variants, approximated to different distribution models. Also, the best fits did not always produce good accuracies due to multi-modal nature of some distributions. However, even after ignoring these inaccuracies, an extensive trial and error method to determine the best distribution model fit is necessary to capture a protocol's behavior. Similar to the simulation step in the traditional portfolio scheduler, introduction of this additional trial and error step would again lead to an increased computation and time complexity in the selection component.

Alternatively, instead of estimating a parameters' overall behaviour, its behaviour only within a certain utility carrying region i.e., the application-provided QoS (parameter) range could be recorded through a histogram with a constant interval (bin) size. As recording would be performed within a constant and relatively short parameter range, the memory usage would also be constant and low. For instance, if the parameter: RTT, represented as $[0, 300, 0, 400]ms$ is stored as a QoS requirement by an application, according to this approach, a histogram with a fixed bin size (say, 10) would be created within $300ms$ and

$400ms$ RTT range. All values lower than $300ms$ are equally important and above $400ms$ are equally useless and therefore, do not require further resolution using definitive bins. This can be considered a reasonable approach because the selection among transport protocols is based on those whose performance satisfies application constraints and therefore comparing their performance only within this satisfaction region (QoS range) would suffice to produce an efficient selection. If none of the transport protocols' performance satisfy the application constraints, a coarser comparison metric such as parameter's mean, variance, etc or a combination of these could be used for protocol selection.

In our DPS framework, the second approach is considered and applied according to Algorithm 1. For a set of M protocol parameters with corresponding priorities P_j, a distribution ratio per bin d_{ij}, calculated from a histogram of N bins with a corresponding bin utility u_{ij}, the net utility (U_{net}) per protocol can be obtained from Equation 4.1 (Alg. 1, line 8). However, the protocols considered for this calculation must be valid, i.e., they must satisfy application constraints (verified using a framework-specific tunable threshold, $thresh_{stat}$) (Alg. 1, lines 1-6). Finally, the protocol corresponding to the maximum U_{net} is selected as the best performing protocol (Alg. 1, line 24). However, if multiple protocols have the same maximum (Alg. 1, line 12), mean ($Mean_{tj}$) of the measured protocol parameter is considered a coarser comparison metric among those protocols. For a set of T transport protocols, the net utility per protocol ($\hat{U}_{net}$), under such circumstances can be obtained from Equation 4.2 (Alg. 1, line 14). Following similar steps as the previous comparison method, the protocol corresponding to the maximum $\hat{U}_{net}$ is selected as the best performing protocol (Alg. 1, line 21). Although with a very low probability of occurrence, if a utility tie between some protocols still occurs, a random selection among those protocols becomes a preferred choice (Alg. 1, line 19).

$$U_{net} = \sum_{j=1}^{M} P_j . \sum_{i=1}^{N} \left(u_{ij} . d_{ij}\right) \tag{4.1}$$

$$\hat{U}_{net} = \sum_{j=1}^{M} P_j . x_{tj}, \tag{4.2}$$

$$where,\ \ x_{tj} = \begin{cases} \frac{Mean_{tj}}{\sum_{t=1}^{T} Mean_{tj}} & if\ high\ \text{Mean}\ is\ better \\ 1 - \frac{Mean_{tj}}{\sum_{t=1}^{T} Mean_{tj}} & if\ low\ \text{Mean}\ is\ better \end{cases}$$

4.3.3 Switching

This component of the framework involves configuring the application flow to run using the selected protocol obtained from the protocol selection component. Therefore, it can be considered to reflect the **"application"** stage of the modified portfolio scheduler. To implement this switching functionality and start data transmission, we use the concept of network sockets. In software terms, network sockets are data structures that provide a means for the higher layers of the network stack to communicate with the transport and lower layers, i.e., they provide APIs to communicate with those layers. In logical terms, network sockets at both the end-hosts behave as end-points of a communication process and are identified by the [*transport protocol, sender IP address, receiver*

IP address, sender port, receiver port] unique combination. Moreover, as our DPS framework includes only TCP variants in its protocol set, we only deal with a specific socket type i.e., stream socket, which deals with reliable data transmission using TCP.

Before initiating an application flow, to define the type of TCP congestion control (TCP variant) for governing the flow, a simple *setsockopt* function from the *socket API*[2] library can be used that modifies the stream socket's configuration before its creation. However, the API does not provide any additional functionality for further modifications of a socket's configuration after its creation. This slightly complicates the switching component's design as its functionality involves modifying the TCP variant during framework run-time.

As our DPS framework aims to work with the existing network infrastructure, without any modifications to the socket API functionalities, we tackle the aforesaid issue with the only viable approach of creating a new stream socket configured with the appropriate protocol every time a protocol different from the existing implemented protocol is obtained from the protocol selection component. However, before creating the new socket, the existing (old) socket is instructed to close and all the application-flow's data is buffered to be channeled to the new socket. Meanwhile, the old socket does not have to necessarily stop transmitting application data the moment it is instructed to be closed. This is because, it still needs to transmit all the packets that were already buffered at its transmission buffer just before receiving the close command (FIN command for initiating TCP connection termination). Also, the socket being inherently reliable, closes only after confirming the delivery of all the transmitted packets (i.e., receiving ACKs for all transmitted packets). Even between this period of receiving the close command and actually closing, the socket still performs all operations according to the TCP congestion control algorithm it was configured with.

Looking from an application flow's perspective, the flow expects to benefit from all the advantages offered from a single continuous TCP connection, i.e., guaranteed and in-order data delivery, flow control, error control, etc., in addition to its indicated QoS requirements to the DPS framework. Therefore, the framework's multiple initiation and termination mechanisms of TCP connections within a single application flow to realize "protocol switching" might devoid the flow of some of these advantages. While these advantages remain valid within any configured TCP connection, the point of switching between these connections possess a significant risk of violating some TCP features. Taking a closer view at this switching point, as the old socket already guarantees the delivery of all unacknowledged packets transmitted before delivering the close command, it is essential to decide on ways to handle further application data transmission using the new socket.

As mentioned earlier, even after instructing the old socket to close and start transmitting data with the new socket, buffered data from the old socket still transmits. Therefore, at the receiver, data from the old socket and the new socket are both received in the corresponding sockets' receive buffer. As the received data are from different TCP connections, there is no provision to ensure in-ordered data delivery to the application layer. Therefore, the supporting DPS framework at the receiver (Figure 4.1b) comprises two separate framework

[2]Python Socket API: `https://docs.python.org/3/library/socket.html`

receive buffers. The incoming data from the old socket are queued in a first buffer, while the simultaneously incoming data from the new socket are queued in the other buffer. As soon as the old socket ends transmission, the first buffer inserts an *end-of-read* flag to indicate the receiver processing pipeline about the connection's end. The processing pipeline initially processes data from the first buffer and after encountering the *end-of-read* flag, switches to processing data from the second buffer and the cycle continues throughout the application flow. As data queued in the individual buffers are already in-order due to the corresponding TCP connection's inherent feature, the use of this two-buffer mechanism ensures in-order delivery to the application layer even during connection switching. Thus, the application flow preserves TCP's data integrity along with other features while running over the DPS framework, thus partially satisfying **RQ3**.

4.3.4 Monitoring and Knowledge Update

The last stage of the modified portfolio scheduler, **"reflection"** is reflected by these components of the DPS framework. The monitoring component's implementation in the framework solely depends on what protocol parameters are required for efficient protocol selection. As mentioned previously, to avoid design complexity by adding multiple parameters for monitoring, we simply consider two basic parameters: *Round-Trip time (RTT)* and *Throughput* experienced by the application flow utilizing a specific transport protocol.

To monitor these parameters, *ss* monitoring tool is used. *Socket statistics (ss)*[3] is a tool that allows an application to obtain various network statistics by querying the sockets. Although, technologies like *eBPF*[4], that allows better accessibility and coverage of parameters for monitoring in the network stack, could be adopted as the monitoring tool, *ss* is much simpler and light-weight that sufficiently exposes the required monitoring parameters and hence is a suitable design choice.

As RTT and throughput statistics of an application flow initiated by the sender are available at the sender and receiver end-hosts respectively, the monitoring component at the sender monitors RTT and throughput is measured at the monitoring component at the receiver. To obtain an overall parameter's statistics, the parameter values are accumulated (monitored) over a specific period of time called monitoring period and at the end of this period, the collected values are flushed out of the monitoring buffer and processed to obtain suitable statistics and the cycle continues again. These suitable statistics refer to the protocols' parameter representation discussed in Section 4.3.2. Based on the discussion, two types of statistics are calculated:

- a histogram of parameter values within application-specified QoS range (H).
- a mean of the parameter values (M).

Additionally, a total count of the parameter's observations (N) is also obtained that is necessary for the knowledge update component, resulting in total

[3]ss Tool's Manual: `https://man7.org/linux/man-pages/man8/ss.8.html`

[4]Monitoring with eBPF: `https://www.brendangregg.com/blog/2021-07-03/how-to-add-bpf-observability.html`

Algorithm 2: Pseudo Code for the Knowledge Update Component.

Input: histogram H, mean M, net-count N, protocol P

1 $P_{stats} \leftarrow$ Get_protocol_statistics(P);
2 **if** $P_{stats} \neq \phi$ **then**
3 $\quad N_{new} \leftarrow N_{old} + N$;
4 $\quad M_{new} \leftarrow \frac{M_{old}.N_{old}+M.N}{N_{new}}$;
5 $\quad D_{new} \leftarrow \frac{D_{old}.N_{old}+H}{N_{new}}$;
6 $\quad P_{stats} \leftarrow$ Update($P_{stats}, [D_{new}, M_{new}, N_{new}]$);
7 **else**
8 $\quad D \leftarrow \frac{H}{N}$;
9 $\quad P_{stats} \leftarrow P_{stats} \cup [D, M, N]$;
10 **end**
11 Store_protocol_statistics(P, P_{stats});

three post-processed statistical outputs.

While keeping track of these processes at the sender is easier as it is a part of the main DPS framework, synchronising the monitoring component of the supporting DPS framework with it requires some indication from the main DPS framework. Therefore, at the end of the monitoring period, the sender sends a DPS framework-specific keyword along with the application data called *magic-string*. The monitoring component at the receiver continually records the measured throughput until it receives this *magic-string*. On receiving this string, it flushes the recorded parameter values and processes them similar to the sender to generate the three post-processed statistical outputs and finally, sends them back to the sender.

With the obtained post-processed parameter statistics, the knowledge update component modifies the corresponding protocol's statistics in the framework database according to Algorithm 2. In case the framework has no knowledge about the protocol behaviour (Alg. 2, line 7), it simply stores the post-processed values in the framework database (Alg. 2, line 9). Otherwise, it updates the stored parameters ($N_{old}, M_{old}, D_{old}$ in Algorithm 2) according to lines $3-5$ of the algorithm. However, it should be noted that the obtained histogram is always converted into a distribution before storing because it can be directly used by the protocol selection component without further processing. Also, as the net count of observations increases, only one field i.e, N increases constantly in memory, while the other fields remain unaffected. This constant growth of the net-count field has a tendency of leading to a memory overload issue and is indeed one of the drawbacks of this parameter representation type. However, due to moderate observation sampling frequency, the rate of growth of this variable is very low. However, it is important to address this issue in future modifications of the framework.

4.4 Framework Operational Flow

After a detailed discussion on the various components of the DPS framework, the following pointers highlight the interactions among these components through a description of the operational flow of the framework (Figure 4.1).

1. For an application flow utilizing the DPS framework, when the flow initiates, it also creates an instance of the DPS framework. This is necessary for creating the appropriately configured stream socket for data transmission. However, it should be noted that at this point in time, only the main DPS framework (Figure 4.1a) is created. The supporting DPS framework instance (Figure 4.1b) would be created only after the receiver gets an initial connection request from the client for the application flow.

2. Assuming the main DPS framework has knowledge about the flow's QoS requirements and candidate TCP variants' statistics, by gathering this information from the database, the protocol selection component is triggered and decides on a selected TCP variant according to Algorithm 1.

3. On receiving this protocol information, the switching component creates a socket instance configured with the selected protocol and initiates the normal connection initiation process to the remote host (receiver). With the reception of the connection request, the receiver also creates a socket instance for the sender along with the supporting DPS framework instance. This is followed by normal application data transmission according to the two-buffer mechanism (Section 4.3.3).

4. During data transmission, the monitoring modules of both the main and supporting frameworks record the corresponding parameter values. However, as only the main framework is aware of the monitoring period, to indicate this to the supporting framework, at the end of this period, the main framework sends a *magic-string* to the receiver along with the application-data. At the receiver, before moving on to the main application queue, all received data pass through a post processing component that searches for the presence of framework-specific keywords, if any. When this component finds the *magic-string*, it forwards it to the monitoring module (Tx Query Indication), that stops monitoring and sends the monitored parameter statistics to the sender. The monitoring, further restarts after sending.

5. The main framework, upon receiving all the required parameter statistics, reports to the knowledge update component that updates the framework database according to Algorithm 2. With this step, the framework completes, what is termed as one operation cycle (framework cycle). The framework cycle repeats throughout the application flow following the same sequence of steps mentioned in pointers 2 - 5, except the additional process of creating new socket instances (if the selected protocol is the same as the previously implemented one) or new framework instances.

4.5 Online Learning Strategy

To understand this section, consider the DPS framework to be deployed for the first time, with a finite protocol set and no information about the candidate protocols' behavior. In other words, the framework database only includes the protocol identities (names) that declare the protocol set. Now an application flow wants to utilize this DPS framework and so, provides its QoS requirements to the framework. However, at this point in time, the protocol selection

Algorithm 3: Pseudo Code for Online Learning Strategy.

```
Input: Protocol List P,
       Protocol List satisfying application constraints P_satisfy,
       Protocol List with no parameter statistics info. P_empty
if random_trigger == ON then
    if P_empty ≠ ∅ then
        random_choice(P_empty);
        set_monitor_period(t);
    else
        random_choice(P);
        set_monitor_period(t);
    end
end
if P_satisfy == ∅ and P_empty ≠ ∅ then
    random_choice(P_empty);
    set_monitor_period(t);
end
if P_satisfy == ∅ and P_empty == ∅ then
    protocol_selection(P);
    set_monitor_period(T);
end
if P_satisfy ≠ ∅ and P_empty ≠ ∅ then
    protocol_selection(P_satisfy);
    set_monitor_period(T);
end
```

component of the framework cannot make a valid selection as it does not have any information about the parameters' statistics of any protocol, although the application-specific parameter is present in the framework database. In such a situation, without further appropriate steps the framework is certain to fall into a deadlock.

To deal with such situations, previous works generally prefer an initial parameters' statistics to be present for all candidate protocols in the framework database. While this is a viable solution, it devoids the framework of its ease of deployability feature. This is because, whenever such a framework is planned to be deployed, an additional training step for gathering parameters' statistics of all candidate protocols needs to be performed before actually using it for the real application. Also with the simulation step ruled out due to previously discussed arguments, this necessitates the development of a strategy through which the framework can autonomously learn about its candidate protocols' behavior while dealing with application flow transmission or in other words, learn online.

The DPS framework, therefore applies the online learning strategy according to Algorithm 3, aiming to satisfy requirement **RQ2**. This learning strategy is driven by its objective to maintain a balance between learning time and overall application performance. Learning time refers to the total time taken by the strategy to gather required parameters' statistics for all the candidate protocols. And overall application performance refers to the net satisfaction or utility level

obtained from the concerned application-QoS metrics' performance.

According to the learning strategy, there are two phases of the DPS framework: *learning phase*, during which the framework learns about its candidate protocols' behavior and, *learnt phase*, when the framework has at-least some knowledge about all its candidate protocols. With the initial state of the framework having no protocol behavior information (i.e., $P_{empty} == P$), the framework obviously moves to a learning phase and the only option available is to make a random choice among the candidate protocols in P_{empty} (Alg. 3, lines 10-11). Fortunately, if the statistics obtained from the implementation of the selected protocol satisfy the application constraints, it becomes a valid protocol and is moved to $P_{satisfy}$ from P_{empty} and the strategy can again select the same protocol for the next framework cycle (Alg. 3, lines 18-19). However, if the statistics do not satisfy the constraints, the strategy continues the same process with the remaining protocols in P_{empty} (Alg. 3, lines 10-11) hoping to find a protocol whose performance does satisfy the constraints. In the worst case, after learning about all the candidate protocols' behavior ($P_{empty} == \phi$), if none of the protocols' performance satisfy the application constraints, the strategy simply selects among the candidate protocols based on the protocol selection method described in Section 4.3.2 (Alg. 3, lines 14-15). This would atleast ensure that the best performing protocol is implemented, despite of their constraint violation.

On the other hand, it is logically correct if the strategy finds a valid protocol and continues selecting it for further framework cycles because it would anyway adequately satisfy the application requirements. However, with this approach, there is a finite chance for the framework to miss out on some better performing protocols that might be present within the framework protocol set that could enhance overall application performance. Also, the main purpose of the strategy to simultaneously learn about the protocols while transmitting application data fails with this approach as learning stops on finding the valid protocol. Therefore, the concept of *random_trigger* event is introduced into the online learning strategy. This *random_trigger* event basically refers to one of the framework cycles when the protocol selection component makes a random selection among candidate protocols irrespective of the performance of those protocols, but prioritizing protocols in P_{empty} (Alg. 3, lines 1-6). As protocols are randomly selected with this trigger, it ensures that the framework does not implement a single protocol throughout the application flow and hence, keeps gathering and updating knowledge about other candidate protocols even during its *learnt phase*. Moreover, the priority to P_{empty} protocol set guarantees the learning of all candidate protocols' behavior by the framework.

However, learning time depends on the probability of occurrence of this *random_trigger* event. Out of N total framework cycles (i.e., implicitly N protocol selection triggers), if one of the selection triggers is governed by the *random_trigger* event, its probability of occurrence is defined as $\frac{1}{N}$. If this probability of occurrence increases, frequency of *random_trigger* events increases and thus, protocols in P_{empty} set are selected more frequently, resulting in lower learning time. However, a considerable risk of selecting an inadequately performing protocol also rises, resulting in an overall application performance degradation. Therefore, to handle this trade-off, the probability of occurrence is a tool that can be used to maintain a proper balance between learning time and overall application performance, which is the main objective of the online learn-

ing strategy. In addition to this tool, to further reduce application performance degradation, the monitoring period can also be tuned appropriately. Therefore, using the learning strategy, while applying protocols selected according to a random selection step, a lower monitoring period is allotted (t) to that framework cycle than when the selection is performed through a protocol selection step (T). As random selection always poses a risk of selecting a poor performing protocol, a lower monitoring period leading to a lower framework cycle time, would effectively reduce the impact of the performance degradation caused by that protocol over the framework cycle time.

4.6 Fairness Framework

Now that a standardized Dynamic Protocol Selection (DPS) conceptual framework is designed, to make it truly usable, it is essential to enable its deployment in a real-network (which is generally heterogeneous in nature). A heterogeneous network comprises of traffics of different intensities and types, governed by different protocols and transmitting between different host operating systems. *Fairness* of a flow is a metric that measures the overall impact the flow creates on other co-existing flows in the network. It measures this impact in terms of its resource sharing capability, for e.g., if a flow is introduced into a network link with a few existing flows, a fair flow would tend to share an equal amount of link bandwidth with the other flows. As the flow is governed by a protocol, the flow's fairness can also be defined as the protocol's fairness. Therefore, fairness between flows can be used as a metric to ensure deployability of the DPS-governed application flows.

Focusing on the fairness only among TCP variants, Belma et. al. [59] performed an extensive survey on the fairness attained with the co-existence of flows governed by different TCP variant combinations. The survey demonstrated the different levels of fairness attained for different TCP variant combinations along with additional effects due to combination of flows running over network paths with unequal RTTs. Therefore, taking into account these variabilities in fairness obtained due to different TCP variant combinations, the DPS framework can potentially leverage its dynamicity in protocol selection functionality to apply only those protocols that showcase an acceptable fairness level at run-time, resulting into an overall fair application flow.

However, the DPS framework designs considered in the previous works or this work are not yet capable to achieve this functionality. Therefore, as a starting point for further research into this direction and an attempt to satisfy requirement **RQ4**, we propose a fairness framework to deploy multiple fair DPS governed-application flows (referred as DPS flows in further discussion) from the same sender host machine. To understand this framework, we consider a sender trying to deploy two DPS flows to either same or different receiver(s). For simplicity, we also consider each DPS framework to include two candidate protocols (P1 and P2) in their protocol sets, without any initial protocol behavior knowledge. The sequence diagram in Figure 4.4 represents this scenario and further illustrates the operational flow of the proposed fairness framework.

The proposed fairness framework (FF) can be considered as a higher-level framework that manages several DPS frameworks. So, an application trying to utilize the DPS framework, first needs to access the FF, that would then

admit the flow according to a certain schedule. The proposed FF, schedules (manages) multiple DPS flows in a round-robin fashion. Therefore, even when multiple flows arrive simultaneously, the FF admits them one-at-a-time, as is illustrated by DPS Flows - 1 and 2 in Figure 4.4. The FF manages each flow through a *lock* and *unlock* methodology. As it manages one DPS flow at a time, it unlocks only that flow while the remaining flows are in the *locked* state. The flows in the *locked* state are not permitted to change protocols while flows in the *unlocked* state are permitted.

In Figure 4.4: step - 1, the FF initially unlocks the DPS Flow-1 along with an additional *State* information. Although, the notion of a *State* in the DPS framework database (Section 4.3.1) refers to the measured network state, the FF uses it to provide each DPS flow with its own state information, i.e., a list of corresponding protocol names (identities) that each existing flow managed by the FF is locked with. With this information, the DPS Flow-1 updates its framework database for the corresponding *State* after undergoing a learning phase according to the online learning strategy (Section 4.5). After moving to a *learnt phase*, the flow indicates it to the FF, which further indicates the DPS Flow-1 to lock itself with a specific protocol (Fig. 4.4, steps 2-3). The DPS Flow-1, on receiving the lock command, selects the best performing protocol according to the selection strategy (Section 4.3.2), locks itself and indicates it to the FF along with the protocol's identity (P1) with which it is locked (Fig. 4.4, step 4). After a lock confirmation from DPS Flow-1, the FF searches for the presence of other DPS flows. If none are present, it would again go back to unlocking DPS Flow-1. However, in our considered scenario, the FF goes on to unlock DPS Flow-2 using similar steps as with Flow-1 (Fig. 4.4, step 5). However, due to presence of multiple flows, the FF simultaneously also initiates a violation detection mechanism to detect unacceptable fairness situations, during the learning phase of DPS Flow-2. The FF implements this violation detection mechanism according to Algorithm 4, that calculates and observes an estimate of the fairness of the deployed DPS flows from the flow sending rates monitored using the *ss* tool[5]. The fairness is calculated using Jain's fairness index [33]. In case a violation is detected (Alg. 4, line 9), the FF indicates it to the unlocked flow (DPS Flow-2 in this scenario) (Fig. 4.4, step 6). On receiving this violation signal, the flow assumes that this violation was caused due to the application of the currently implemented protocol and therefore, promptly modifies it and marks its as a "violating" protocol in the database for the current *State*. When DPS Flow-2 moves to the learnt phase, it carries out the same sequence of steps as Flow-1 (Fig. 4.4, steps 7-9) to move to the locked state. The FF again unlocks DPS Flow-1, but now with a different *State* (P2), which is the protocol with which other flows (DPS Flow-2) managed by the FF is locked (Fig. 4.4, step 10). However, unlike the previously unlocked DPS Flow-1 scenario, the violation detection mechanism initiated previously still keeps running due to the existence of multiple flows in the FF (Fig. 4.4, step 11). However, during further FF operation, if the FF unlocks a flow with an already learnt state for the provided *State* information, the flow simply makes a selection according to Algorithm 3, runs for the appropriate monitoring period, updates the database

[5]The FF, being located at the sender side, sending rates can be monitored seamlessly without any communication overhead compared to throughput rates and the trends in throughput and sending rates, being similar, an approximate calculation of fairness can therefore also be obtained from sending rates of the concerned flows.

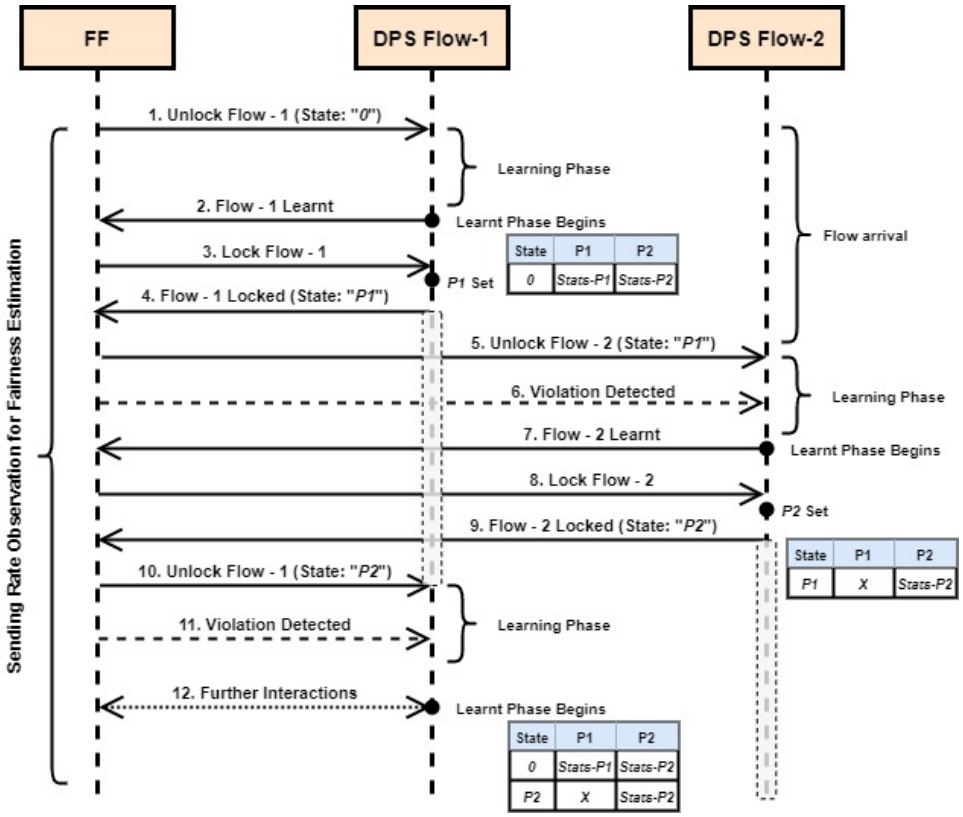


Figure 4.4: **Sequence Diagram for the fairness framework (FF) mechanism during its initial stage of operation. The *locked* states of the DPS application flows (DPS Flow-N) are represented by semi-opaque vertical blocks over the flows' lifelines. The tables represent flow-corresponding DPS framework database's states at specific instances and the opaque dots over flows' lifelines mark important events in the respective DPS frameworks. The dashed arrows represent uncertain interactions.**

for the applied protocol and indicates the FF with a learnt signal for locking itself as in Figure 4.4, steps 2-4.

Moreover, there can be situations where a protocol is wrongly marked as "violating" in the framework database due to reasons such as irregular network fluctuations during protocol implementation or inaccuracies within the violation detection strategy. In such situations, due to the existence of the *random_trigger* event in the online learning strategy, the "violating"-marked protocol still gets a fair chance to rectify its behavior information recorded in the database as it has an equal probability to be selected as any other candidate protocol during the *random_trigger* event.

Algorithm 4: Pseudo Code for Fairness Violation Detection.

Input: Threshold Fairness F_{thresh},
Estimated Fairness F,
Previous Estimated Fairness F_{prev}

$violation_{count}$, V_{thresh}: No. of consecutive fairness threshold violations, corresponding count threshold;
$decrease_{count}$, D_{thresh}: No. of consecutive fairness drops, corresponding count threshold;
if $F < F_{thresh}$ **then**
    $violation_{count} \leftarrow violation_{count} + 1$;
    if $F < F_{prev}$ **then**
        $decrease_{count} \leftarrow decrease_{count} + 1$;
    end
    if $decrease_{count} > D_{thresh}$ *or* $violation_{count} > V_{thresh}$ **then**
        $violating \leftarrow$ True ; ▷ violation detected
        reset_to_zero($violation_{count}$, $decrease_{count}$);
    end
else
    decrement_by_1_until_zeroed($violation_{count}$, $decrease_{count}$);
end

Chapter 5

Evaluation

Through a series of experiments and further analysis on the observations, this chapter validates our proposed frameworks discussed in Chapter 4. Through a systematic approach, it initially describes the environment used to conduct the experiments in Section 5.1. It further describes the different types of experiments performed to analyze and validate the functionalities of different components our proposed frameworks: Protocol Switching Component (Section 5.2), Online Learning Strategy (Section 5.3) and Fairness Framework (Section 5.4).

5.1 Experiment Setup

All experiments were conducted using the *Mininet* network emulator[1] in a Linux (version 4.10.0-27 generic) system configured with Ubuntu 16.04.6 LTS operating system. Figure 5.1 illustrates the network topology used in the Mininet environment for performing the experiments. As most of the complexities of our proposed work lies at the end-hosts, considering a simple topology as in Figure 5.1 suffices for our purpose. The implementations of our proposed DPS and Fairness Frameworks at the end-hosts were done using Python3.7. Additionally, to perform measurements for the experiments, we relied on netem[2] and tcpdump[3].

Further, to verify and analyze the proposed solutions correspondingly addressing the previously stated DPS framework requirements (Section 4.1), three different types of experiments were performed:

1. Protocol Switching Performance (verifying **RQ3**)

2. Online Learning Analysis (verifying **RQ2**)

3. Fairness Framework Performance (verifying **RQ4**)

[1]Mininet Github Repository: `https://github.com/mininet/mininet`
[2]TC-NETEM Manual: `https://man7.org/linux/man-pages/man8/tc-netem.8.html`
[3]TCPDUMP Manual: `https://www.tcpdump.org/manpages/tcpdump.1.html`

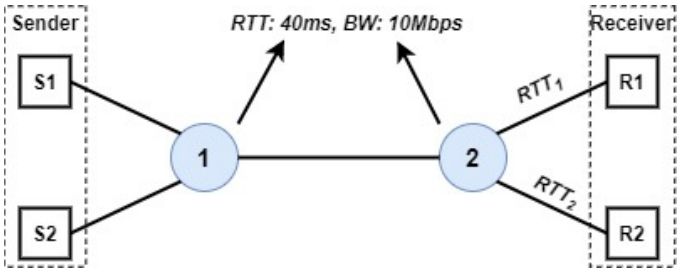


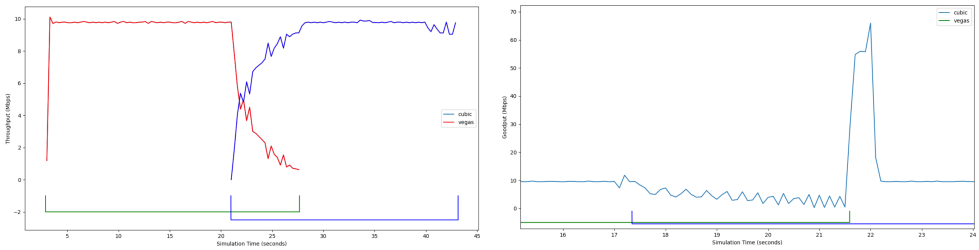
Figure 5.1: **Mininet topology used for the experiments. The link between the routers "1" and "2" is configured to have a bandwidth of 10Mbps and a round-trip delay (RTT) of 40ms for all experiments. To tune the RTTs of the network paths towards receivers "R1" and "R2", the corresponding variables, "RTT_1" and "RTT_2" are appropriately set in some experiments (default: 0ms).**

5.2 Protocol Switching Performance

As described in Section 4.3.3, despite of the creation of a new connection during protocol switching, the characteristics of a typical single TCP connection are preserved due to: (i) inherent nature of the stream socket (guaranteed packet delivery) and, (ii) two-buffer mechanism in the supporting DPS framework (in-order packet delivery). However, to minimize the effects of protocol switching and mimic a single connection behavior, it is equally important to ensure an unaffected delivery rate during the switching period and the upcoming switching instance. We define switching period as the total time taken by the old TCP connection to complete data transmission after receiving a termination command. The switching instance is the point in time when the application just begins receiving data from the new TCP connection.

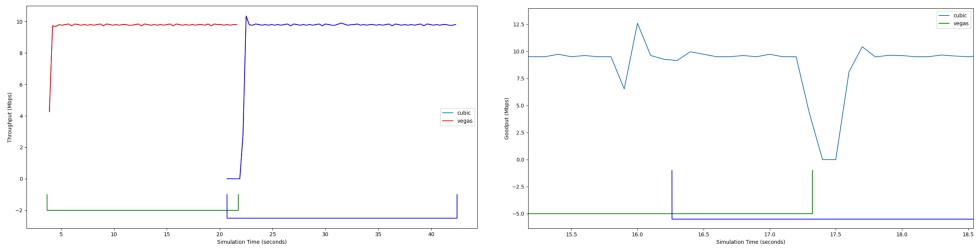
5.2.1 Switching Experiment

To examine this behavior during the switching period we considered two extreme scenarios of switching methods in our experiment. In the first scenario (Scenario-1), during switching, we configured the DPS framework to start data transmission from the new connection immediately after indicating the old connection to terminate (simultaneous data transmissions from the corresponding connections during the switching period). In the second scenario (Scenario-2), we configured the framework to start data transmission from the new connection only after the old connection ends, i.e., after receiving a FIN packet from the receiver in the old connection (single data transmission from the old connection during the switching period). For both the scenarios, we used the topology in Figure 5.1 with a single application flow from S1 to R1. To observe the switching behavior under extreme circumstances, the application sending rate was configured to be much higher than the topology's link bandwidth (10Mbps). Also, to track minor details in the application's delivery rate, a high application receive rate was configured. In the experiment, we basically configured the framework to run a first protocol for a pre-defined duration and then perform protocol switching with a second protocol. The configuration details are listed



(a) **Throughput for Scenario-1.**

(b) **Application goodput (magnified) for Scenario-1.**



(c) **Throughput for Scenario-2.**

(d) **Application goodput (magnified) for Scenario-2.**

Figure 5.2: **Plots demonstrating the impacts of the switching methodologies in the two considered scenarios on throughput and corresponding application goodput during the switching period at the receiver. In the throughput plots, red indicates TCP Cubic and blue indicates TCP Vegas. The horizontal square brackets indicate the duration of a single TCP connection. The switching period is indicated by the start of the blue and end of the red horizontal square brackets.**

in Table 5.1.

Parameters	Values
Application Sending Rate	1 Gbps
Application Receive Rate	80 Mbps
Run-time per Protocol	15 secs

Table 5.1: **Configuration for the switching experiment.**

5.2.2 Switching Observation

With the aforementioned switching experiment setup, for a protocol switch from TCP's Cubic to Vegas variant, Figure 5.2 demonstrates the impact on application delivery performance for the two scenarios at the receiver.

In Scenario-1, as both TCP connections are involved in transmission during the switching period and share the same link (Link 1-2 in Figure 5.1), they compete for the link's available bandwidth throughout this period, resulting in a drop in the respective connection's throughputs (Figure 5.2a). However, from the application flow's perspective, data received from the Vegas connection is

not useful until there is an incoming data flow from the old (Cubic) connection. Therefore, the reduced throughput of the Cubic connection leads to a reduction in the application flow goodput during the switching period (Figure 5.2b). However, at the switching instance, as all incoming data from the Cubic connection are already processed, incoming data from the new (Vegas) connection becomes useful. Hence, there is a sharp momentary rise in application goodput due to the already buffered data from the Vegas connection during the switching period (Figure 5.2b).

In contrast to Scenario-1, Scenario-2 has an opposite trend of the switching behavior. As Cubic connection is the only one to transmit data during the switching period, there is no such competition for the link's bandwidth like in Scenario-1. Therefore, the connection's throughput remains unchanged (Figure 5.2c). And this directly translates to an unchanged (higher than in Scenario-1) application flow goodput during the switching period (Figure 5.2d). However, the switching instance experiences a momentary dip in the application goodput (Figure 5.2d). This is caused due to the absence of any data to process from the new (Vegas) connection just after finishing processing data from the Cubic connection. In Scenario-2, as data transmission in the new (Vegas) connection begins only after receiving a FIN for the old (Cubic) connection, it takes 1 RTT ($\frac{1}{2}$ RTT for FIN transmission from receiver to sender + $\frac{1}{2}$ RTT for transmission of first packet from sender to receiver) after the end of the Cubic connection to start processing data from the Vegas connection.

Moreover, comparing the lengths of the switching periods of both scenarios (Figures 5.2a and 5.2c), it is much shorter in Scenario-2 as the total data buffered in the old (Cubic) connection's transmit buffer is transferred much faster (high throughput) than in Scenario-1.

5.2.3 Switching Analysis

Similar switching experiments were performed for different combinations of TCP variants. As there are a large number of TCP variants today, to reduce evaluation complexity and make a generalized analysis, we considered including one variant from each congestion control category (Section 2.2.1) resulting in a total of 4 variants (i.e., Cubic (*Loss-based*), Vegas (*Delay-based*), Illinois (*Loss-Delay-based*), BBR (*Model-based*)) and 12 switching combinations. Also, rather than only considering extreme scenarios of switching methodologies, we added three other scenarios taking a middle ground between the aforementioned scenarios. The 5 considered switching methods are as follows:

- **Full-Rate:** Start sending data at application rate from the new connection immediately after indicating to close the old connection (previously, Scenario-1).

- **250pps-Rate:** Start sending data at a rate slightly lower than the application rate (250pps (packets-per-second)) from the new connection immediately after indicating to close the old connection.

- **Variable-Rate:** Start sending data at a non-linearly increasing rate, r ($r_n = ((\frac{5}{4})^n r_o$, where $n \epsilon \mathbb{N}$) limited by 5pps (r_o) and 250pps from the new connection immediately after indicating to close the old connection.

- **5pps-Rate:** Start sending data at a rate much lower than the application rate (5pps (packets-per-second)) from the new connection immediately after indicating to close the old connection.

- **Full-pause:** Start sending data at application rate from the new connection only after termination of the old connection (receiving a FIN indication from the old connection) (previously, Scenario-2).

For all these combinations and scenarios, we analysed two performance metrics during the switching period: average application goodput and average duration of the switching period[4]. Figure 5.3 presents our findings.

Among the considered scenarios, 5pps-Rate and Full-pause switching methods are the most consistent in presenting desirable values of both switching performance metrics for all the protocol switch combinations. Therefore, these can be considered the best switching methods among others. Moreover, the application goodput's momentary dip at the switching instance, discussed previously, is an issue for the full-pause switching method that 5pps-Rate method does not experience due to the presence of data transmission (although at a very low rate) from the new connection during the switching period. However, the overall average performance is relatively high for full-pause. Therefore, this is a trade-off that needs be addressed while further selecting among the two methods[5].

Comparing the impact of switching methods per protocol switch combination, switches from Vegas have the worst switching performances because, Vegas being a delay-based variant, is the least aggressive among all and so, the new TCP connection governed by another variant takes over most of the bandwidth share during the switching period. On the other hand, Cubic, being a loss-based variant, is the most aggressive and, therefore, switches from Cubic have an overall consistent and good switching performance. The duration of the switching period is a direct result of the combined effects of the size of the old connection's untransmitted data at the beginning of the switching period and the simultaneous data transmission of the new connection. Therefore, although switches from Illinois mostly have good application goodputs being a loss-delay-based variant, its switching period's duration is exceptionally high. This is because Illinois, in an attempt to utilize more bandwidth before encountering congestion, increments its congestion window curve in a concave fashion rather than the regular convex trend leading to a much higher data injection into the connection's transmit buffer compared to other variants, ultimately resulting into long untransmitted data queues during switching period initiation. Looking at BBR, it has an inherent mechanism to periodically estimate the network path's true RTT by momentarily making a steep increase and then decrease in its congestion window to fill up and empty the network buffers respectively. As a typical BBR flow always initiates with this estimation mechanism, if a new connection is governed by BBR, the connection would always take over the available bandwidth momentarily from the old connection. Therefore, during a switch to

[4]Although switching period's duration does not affect the application performance directly, it can be an important parameter for the efficient operation of the DPS framework.

[5]Although the 5pps-Rate method might seem to have a significantly lower switching performance than Full-pause, according to the trend, the performance can improve for further lower-rate methods making the trade-off more significant. The basic takeaway, here, is to demonstrate the competitive nature of both full-pause and lower-rate switching methods.

		Protocol Switches											
		V-C	V-I	V-B	C-V	C-I	C-B	I-V	I-C	I-B	B-V	B-C	B-I
Scenarios	Full-Rate	0.37	0.34	1.28	9.26	8.09	7.42	8.41	8.05	3.21	7.27	6.25	3.70
	250pps-Rate	0.81	0.64	1.57	9.09	7.27	9.51	8.44	8.04	4.15	7.87	7.35	2.47
	Var.-Rate	0.99	0.84	2.84	9.15	8.97	8.97	8.79	8.11	4.22	8.65	6.79	6.19
	5pps-Rate	7.55	7.42	7.54	8.92	7.31	7.89	8.19	7.82	5.91	8.89	8.76	8.12
	Full-pause	9.78	9.79	9.78	9.78	9.81	9.77	9.78	9.79	9.79	8.84	8.74	8.61

***C = Cubic, V = Vegas, I = Illinois, B = BBR**

(a) **Average application goodput achieved during protocol switching (in Mbps).**

		Protocol Switches											
		V-C	V-I	V-B	C-V	C-I	C-B	I-V	I-C	I-B	B-V	B-C	B-I
Scenarios	Full-Rate	41.95	115.20	16.10	2.48	3.05	3.08	9.11	9.50	18.21	1.68	1.95	4.46
	250pps-Rate	33.51	70.00	15.82	2.95	3.18	3.02	8.61	9.33	15.51	1.57	1.71	4.54
	Var.-Rate	21.24	38.77	6.26	2.34	2.83	2.46	8.80	9.34	15.60	1.45	1.60	2.89
	5pps-Rate	2.87	2.24	2.33	2.55	2.86	2.77	8.74	9.24	12.22	1.39	1.48	1.56
	Full-pause	1.78	1.70	1.72	2.36	2.28	2.30	7.46	7.41	7.53	1.38	1.32	1.37

***C = Cubic, V = Vegas, I = Illinois, B = BBR**

(b) **Average duration of switching period (in seconds).**

Figure 5.3: **Effects on the considered switching performance metrics for different scenarios during protocol switching. The darker the shade of the cell, the more preferred is the corresponding performance metric value within the cell. Each scenario (row) was run 10 times for each protocol switch combination (column).**

BBR, Illinois, being a loss-delay-based variant, reacts to this momentary congestion with a slow rising congestion window similar to a delay-based variant and thus, leads to a worse performance compared to its other protocol switches. On the other hand, switches from BBR have a fairly good performance because of BBR's periodic estimation mechanism.

However, it should be noted that the switching performances presented in Figure 5.3 are obtained only from the first protocol switch in the proposed DPS framework. ***The subsequent switches in the upcoming framework cycles will always result in a better or similar switching performance to the first***. To justify this statement, consider a switching scenario: (P1-P2, P2-P1, P1-P2), where protocols P1 and P2 are consecutively switched for three framework cycles. Considering this scenario taking place in an isolated network, protocol P1 would initiate by increasing its congestion window to a higher range in the first framework cycle than in the third. This is because during the third cycle, when P1 initiates, it would already be sharing part of the link bandwidth with the previous P2 connection that limits its congestion window rise compared to the first framework cycle, when P1 utilizes the entire bandwidth to increase its congestion window. And as higher congestion window results into higher data injection into a connection's transmit buffer, it ultimately results into forming longer data queues during switch from P1 in the first cycle than in the third. As a result, P1-P2 switch, during the first framework cycle has a longer switching period's duration (resulting in a worse switching performance) than any subsequent P1-P2 switches.

5.3 Online Learning Analysis

As discussed in Section 4.5, while learning online through the proposed learning strategy, the DPS framework experiences a trade-off between the overall application flow performance that it governs and learning time. However, the strategy also introduces the concept of *random_trigger* event whose probability of occurrence (p) can be used as a tool to regulate this trade-off, either statically (p remains the same throughout the application flow) or dynamically (p can be modified during the application flow).

5.3.1 Learning Experiment

In this experiment, we analyzed the impact of this probability of occurrence of *random_trigger* event (p) on overall application performance and learning time by the DPS framework. For obvious reasons, all experiments were performed during the learning phase of the DPS framework. To realize this experiment, a single application flow governed with our DPS framework was deployed in the Mininet network of Figure 5.1 from S1 to R1 and run until the framework completed learning. Similar to the protocol switching experiment, the application sending rate was configured to a high value due to the same reason. As per the discussions in Section 4.5, within the learning strategy, the monitoring period for framework cycles with a random selection step is configured with a lower value compared to framework cycles with a protocol selection step[6]. Moreover, drawing conclusions from the switching experiments (Section 5.2.3), we configured the DPS framework with the Full-pause switching methodology due to its superior switching performance. However, as a protocol would have a relatively higher probability to misalign from its actual behavior during the switching period, it is best to avoid recording its behavior during this period to further avoid selection inaccuracies. Therefore, in addition to the monitoring period, we introduce a switch period that begins just after initiating the new connection during protocol switching. During this period, no monitoring takes place. Therefore, the switch period is configured to a value slightly higher than the maximum switching period from Figure 5.3b for Full-pause switching method. These configured values are listed in Table 5.2.

Parameters	**Values**
Application Sending Rate	1 Gbps
Monitoring Period$_{protocol-selection}$	60 secs
Monitoring Period$_{random-selection}$	30 secs
Switching Method	Full-pause
Switch Period	10 secs
Application Priority$_{RTT}$	0.4
Application Priority$_{Throughput}$	0.6
Application Constraints$_{RTT}$	$[0, 300, 0, 350]\, ms$
Application Constraints$_{Throughput}$	$[9.5, 50, 8, 0]\, Mbps$

Table 5.2: **Configuration for the online learning experiment.**

[6]Although monitoring period can be tuned appropriately to regulate the degree of application performance degradation, we do not make further analysis on this as direct inferences can already be drawn without any specific experiments

5.3.2 Learning Observation and Analysis

Figure 5.4 presents our findings. The learning cycles represent the number of framework cycles it takes for the DPS framework to complete learning. As the duration of a framework cycle is governed by the monitoring and switch period, the learning time is directly proportional to the learning cycles[7]. The Net Utility field represents the overall application performance during learning. For a single experiment, it is calculated as the sum of the utilities of the selected protocols for all learning cycles during the learning time. However, to reflect the impact of *random_trigger* event occurrence probability (p) with a relatively small number of experiment iterations, the method to calculate utility values from the observations are as follows:

- If a selected protocol satisfies application constraints, the utility is 1.
- If a selected protocol does not satisfy application constraints, the utility is 0.

For a DPS framework with 2 candidate protocols, it can be observed from Figure 5.4a that with a decrease in the occurrence probability (p) of *random_trigger* event, the framework learning time (learning cycles) increases. This observation correctly aligns with one of the goals of p as a tuning tool[8]. From observing the trend in the net utility, one might infer that the overall application performance improves with a drop in p. However, the presented utility value is the overall utility over the learning time. So, calculating net utility over a higher learning time would anyways result in a corresponding higher net utility. Hence, the observed trend does not correctly represent the second goal of p as a tuning toll i.e., improving overall application performance with reducing p. To verify the second goal, a similar trend should be obtained with a constant time interval. Figure 5.5a demonstrates this trend by evaluating net utilities over constant times. For a simple yet effective analysis of comparing among utilities obtained for different p values, these constant times are set to learning times corresponding to one of the p values (t_1 and $t_{0.5}$ respectively for each observation table, where t_p = average learning time of DPS framework configured with occurrence probability p). Therefore, from Figures 5.4a and 5.5a, the role of p as a tuning tool can be justified.

Moreover, to further demonstrate the validity of our findings to different framework configurations, we also verified the existence of a similar nature of the impact of different p values on the overall application performance and learning time(cycles) for a DPS framework with 4 protocols as illustrated in Figures 5.4b and 5.5b.

[7]Learning cycles and learning time are presented side-by-side to give an idea that learning time does not always hold such a high value and thus, can be tuned appropriately based on the monitoring and switch periods.

[8]It should, however be noted that this impact of p can only be observed if at least one of the candidate protocols satisfies the application constraints. Because, if none of the protocols satisfy the constraints, the learning strategy would consecutively select all the candidate protocols to find the one that does satisfy the constraints and as a result would always lead to a learning time(cycles) corresponding to $p = 1$ despite of any configured p values.

Protocol Set: (Cubic, Vegas)			
Occurrence Prob. (p)	**Net Utility**	**Learning Time (t_p)**	**Learning Cycles**
1	1.40 ± 0.19	$101.69 \pm 0.99s$	2 ± 0
0.5	1.70 ± 0.33	$131.43 \pm 19.31s$	2.45 ± 0.29
0.25	2.50 ± 0.78	$177.97 \pm 40.01s$	3.15 ± 0.62

(a) **Effect with 2 protocols in the framework protocol set.**

Protocol Set: (Cubic, Vegas, Illinois, BBR)			
Occurrence Prob. (p)	**Net Utility**	**Learning Time (t_p)**	**Learning Cycles**
1	2.75 ± 0.28	$220.69 \pm 5.96s$	4 ± 0
0.5	4.25 ± 0.57	$345.83 \pm 27.25s$	5.8 ± 0.41
0.25	8.95 ± 1.41	$636.92 \pm 84.54s$	10 ± 1.23

(b) **Effect with 4 protocols in the framework protocol set.**

Figure 5.4: **Impact of the *random_trigger* event occurrence probability (p) on overall application performance (represented in terms of utility) and learning time. For each value of p, the experiment was repeated 20 times. Results with 90% confidence are presented.**

Occurrence Prob. (p)	**Net Utility**
1	1.40 ± 0.19
0.5	1.48 ± 0.16
0.25	1.70 ± 0.12

Occurrence Prob. (p)	**Net Utility**
0.5	1.70 ± 0.33
0.25	1.97 ± 0.19

(a) **Effect with 2 framework protocol-set configuration (Vegas, Cubic). The constant time interval is set to the learning time corresponding to $p = 1$ (left table) and $p = 0.5$ (right table).**

Occurrence Prob. (p)	**Net Utility**
1	2.75 ± 0.28
0.5	2.82 ± 0.17
0.25	3.35 ± 0.10

Occurrence Prob. (p)	**Net Utility**
0.5	4.25 ± 0.57
0.25	4.44 ± 0.18

(b) **Effect with 4 framework protocol-set configuration (Vegas, Cubic, Illinois, BBR). The constant time interval is set to the learning time corresponding to $p = 1$ (left table) and $p = 0.5$ (right table).**

Figure 5.5: **Impact of the *random_trigger* event occurrence probability p on overall application performance (represented in terms of utility) over a constant time interval. For each table, this constant time interval is taken to be equal to the average learning time corresponding to the highlighted p value (row). For each value of p (non-highlighted rows), the net utility calculation within the constant interval was done for all 20 iterations of the experiment. Results with 90% confidence are presented.**

5.3.3 Application of p as a tuning tool

Now that the goals of p as a tuning tool is verified, it is equally essential to identify potential application areas where this tuning can prove to be beneficial.

Based on the duration of existence of a typical TCP connection (application flow), flows can be broadly categorized into two types: (i) Short flow and, (ii) Long flow. Let us define short flows as those flows that complete within the DPS framework learning time ($t_{run} \leq t_p$). And long flows as those flows that

run for a much longer duration than the framework learning time ($t_{run} >> t_p$). Let us consider the framework starts with the learning phase (i.e., it has no initial knowledge about the candidate protocols' behavior) and the application flow (short/long) starts with the initiation of the DPS framework. After the framework completes learning, in case of the long flow, as it would still have a significant duration of run-time to complete, it can easily benefit from a learnt DPS framework during its remaining run-time. However, in case of the short flow, the flow would have already completed its run-time before the framework completes learning and therefore can only benefit while the framework learns. Therefore, to enable both types of flows to benefit from the DPS framework, the following strategy can be used:

- For long flows, a faster learning method can be adopted. The faster the framework completes learning, the higher the proportion of the application flow run-time governed by the learnt phase of the DPS framework. Therefore, the DPS framework can be configured with a higher value of p while governing such types of flows.
- For short flows, application performance during the learning phase of the framework can be prioritized over learning time. Reducing the learning time does not make much sense to the application flow here as the flow would anyway complete within the learning time. Therefore, the DPS framework can be configured with a smaller value of p while governing such types of flows.

5.4 Fairness Framework Performance

To enable the co-existence of multiple DPS framework governed-application flows (DPS flows) originating from a single sender, Section 4.6 proposed a framework, referred to as Fairness Framework (FF), that ensures an acceptable fairness between these flows. Rather than directly influencing what protocols are to be implemented in the DPS flows, as the FF only aids the corresponding DPS frameworks during their learning phase, the instantaneous fairness between the application flows are not always guaranteed to be satisfactory.

5.4.1 Fairness Experiment

In this experiment, we observed and analyzed the overall fairness and flow performance achieved with the application of the Fairness Framework (FF) while deploying multiple DPS flows. For simplicity, we deal with 2 application flows deployed in the Mininet topology in Figure 5.1, originating from S1 to receivers R1 and R2 respectively and run for a pre-defined duration. Although the FF is designed to benefit multiple DPS flows originating from the same sender, to observe the nature of fairness in case of flows originating from multiple senders, a separate scenario with two DPS flows each originating from S1 and S2 to R1 and R2 respectively is also considered. Inferring from Section 5.3.2, to speed up the process of observing the overall fairness both during the learning and learnt phase of the DPS flows, we tuned the *random_trigger* event occurrence probability (p) to a higher value during the learning phase. However, we dynamically reduced p during the learnt phase of the flows to minimize the risk of

application performance degradation. The remaining configurations were kept similar to the previous experiment set up. Table 5.3 lists these configurations.

Parameters	Values
Application Sending Rate	1 Gbps
Application Receive Rate	10 Mbps
Monitoring Period$_{protocol-selection}$	60 secs
Monitoring Period$_{random-selection}$	30 secs
Switching Method	Full-pause
Switch Period	10 secs
Application Priority$_{RTT}$	0.4
Application Priority$_{Throughput}$	0.6
Application Constraints$_{RTT}$	$[0, 350, 0, 450]\, ms$
Application Constraints$_{Throughput}$	$[9.5, 50, 8, 0]\, Mbps$
Occurrence Probability ($p_{learning}$)	1
Occurrence Probability (p_{learnt})	0.25
Threshold Fairness	0.80
Experiment run-time	1200 secs

Table 5.3: **Configuration for the fairness framework performance experiment.**

5.4.2 Fairness Observation

With the set configuration, Figure 5.6 illustrates the variation in the fairness obtained throughout the operation of the FF. For simplicity in understanding the reasoning behind these fairness fluctuations, this observation is made for DPS flows with only 2 protocols (Vegas and Cubic) in their frameworks' protocol set. Moreover, as the FF manages each flow in a round-robin fashion, the observations are recorded only after the moment when the FF begins managing the second flow (after locking the first flow) i.e., when 2 flows start to co-exist in the network.

The blue-shaded region indicates the consecutive learning phases of the DPS flows respectively. However, due to the availability of only 2 candidate protocols in the frameworks, it is relatively simple and fast to find the protocol combination that provides the best fairness in the existing network and therefore the learning phase completes within a short duration. As a result of the decisions made by the DPS flows during their learning phases, a good overall fairness is thus achieved during their learnt phases (regions except blue-shaded). Moreover, due to the occurrence of *random_trigger* events during learnt phase, there might be situations where a poorly performing protocol (in terms of fairness) is implemented, leading to a drop in the fairness (indicated in the grey-shaded regions). Although the violation detection algorithm manages to detect the fairness drop and prevent further performance degradation, the duration of this drop (degree of performance degradation) depends on the algorithm's detection speed and accuracy.

Therefore, the overall fairness is directly influenced by the fairness performance during the learning phase and the learnt phase. While a relatively good fairness can be expected in the learnt phase, the fairness obtained in the learning phase highly depends on the nature of the candidate protocols in the network

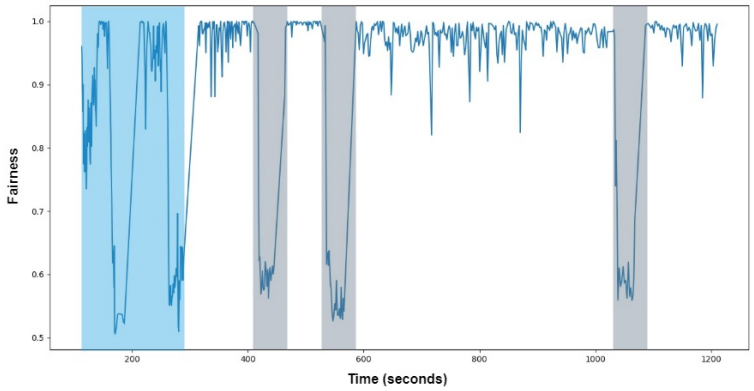


Figure 5.6: **Variation of instantaneous fairness between the two DPS flows deployed in the Mininet topology. Each DPS flow includes 2 protocols (Cubic and Vegas) in their framework protocol set and these flows are managed by the proposed Fairness Framework (FF). The blue-shaded region corresponds to the fairness obtained during learning phases of the DPS flows. The grey-shaded region corresponds to the fairness obtained due to implementation of "violating" protocols during *random_trigger* events in the flows' DPS frameworks.**

and their fairness properties while co-existing with flows governed by different(same) protocol.

5.4.3 Fairness Analysis

The nature of the observed fairness variations are expected to vary depending on the number of candidate protocols considered for the DPS flows and the behavior of these protocols in different network conditions. Therefore, to further validate our inference from Section 5.4.2, we extended our experiment scenario to also observe fairness between DPS flows with a framework protocol set consisting of 4 candidate protocols (Vegas, Cubic, Illinois, BBR) in addition to the previously discussed 2-candidate protocol scenario. For emulating variation in the network condition, we again assume 2 scenarios: (i) Application flows with similar network path delay ($RTT_1 = RTT_2 = 0ms$ in Figure 5.1) and , (ii) Application flows with different network path delays ($RTT_1 = 0ms, RTT_2 = 100ms$ in Figure 5.1).

Further, to validate the importance of the existence of our proposed Fairness Framework (FF), for each of the above mentioned scenarios, we carried out similar experiments using DPS flows without the FF, i.e., we simply deployed DPS flows in the network without any entity managing the flows. As a comparison baseline, we also evaluated the fairness obtained with flows deployed without any DPS framework i.e., the flows' governing protocols remain constant throughout their run-time (static flows). As stated previously, we also carried out experiments with flows originating from multiple senders. Similar to the protocol switching experiments, each TCP variant considered for this experiment represents one of the congestion control categories (Section 2.2.1)

so as to enhance the generality of our experiment.

Figure 5.7 presents the average fairness obtained between static flows for different protocol combination scenarios. The low fairness obtained for some combinations highlights the limitation of the static nature (no provision to change protocols) of the flows. Further, on comparing Figures 5.7a and 5.7b it can be inferred that the fairness does not remain constant with different network conditions (in this case, degrades with an increase in RTT difference). However, the existence of some protocol combinations with acceptable (high) fairness values in both the network conditions highlights the potential improvements in fairness that can be brought about by dynamically configuring the flows to one of these protocol combinations.

Our proposed DPS framework along with the support from FF, therefore leverages the existence of these protocol combinations with acceptable fairness to produce acceptable overall fairness, even with different network conditions as represented by the second scenario in Figure 5.8. As discussed previously and presented in Figure 5.8, for the scenario with the support of FF (same sender), the average fairness obtained during the learnt phase (Learnt Fairness) is relatively higher than during the learning phase (Learning Fairness). And as the average fairness obtained in these phases directly affects the average fairness obtained over the total emulation time (Net Fairness), it is relatively lower.

However, despite of using the DPS framework, the average fairness between DPS flows is much lower when deployed without the support of our proposed FF (no FF scenario in Figure 5.8). This low fairness further highlights the importance of the FF and the necessity of another entity to manage the DPS flows. When multiple DPS flows are deployed without proper management, each DPS framework independently (simultaneously) tries to implement the most efficient protocol according to its recorded statistics. A protocol switch in a single DPS flow could potentially change the network state and behavior of the protocols governing neighboring flows, which might further lead to a protocol switch in the neighboring DPS flows as well. This would eventually lead to a domino effect, resulting in frequent protocol switch, unstable and hence, unreliable overall application performance and fairness.

Moreover, as previous works only deal with the DPS framework (i.e., no additional fairness framework), the fairness obtained for the "no FF" scenario can be considered to represent the fairness performance of the existing literature. Therefore, our work can be considered to improve the fairness performance of the deployed DPS flows by $\approx 15\%$.

On the other hand, when multiple DPS flows originating from different senders (with FF (different senders) scenario in Figure 5.8) are considered, even with the FF implemented, their fairness performance is similar to the no FF scenario. This is because, as each flow is guided by a different FF (located at the corresponding sender), both flows are unaware of each others existence and as a result the fairness drops.

Further, taking a closer look at the individual application flow performance (utility)[9] for the aforementioned scenarios, Figure 5.9 provides some inference. As the scenario with FF (FF_1) is systematic and presents a better fairness performance, it directly translates to a stable and an approximately equal applica-

[9]The calculated utility per experiment iteration is the average utility based on equation 4.1 with the configured parameter constraints and priorities (Table 5.3) for a 100 seconds sliding window over the experiment run-time.

	Vegas	Cubic	Illinois	BBR
Vegas	0.96	0.60	0.64	0.72
Cubic	0.60	0.99	0.65	0.89
Illinois	0.64	0.65	0.95	0.95
BBR	0.72	0.89	0.95	0.98

(a) **Inter- and Intra-fairness with equal RTT** ($\Delta RTT = 0ms$).

	Vegas	Cubic	Illinois	BBR
Vegas	0.89	0.56	0.56	0.71
Cubic	0.56	0.99	0.64	0.81
Illinois	0.56	0.64	0.86	0.88
BBR	0.71	0.81	0.88	0.74

(b) **Inter- and Intra-fairness with unequal RTT** ($\Delta RTT = 100ms$).

Figure 5.7: **Fairness between two application flows governed by different(same) TCP variants that remain static (i.e., no dynamic changes of protocols) throughout the respective flows' duration. The darker the shade of the cell, the more preferred is the corresponding fairness value within the cell. The experiment for each combination of variants was repeated 10 times.**

Scenario	Protocol Set	Net Fairness	Learnt Fairness	Learning Fairness
no FF	C,V	0.77	-	-
	C,V,I,B	0.74	-	-
with FF	C,V	0.89	0.93	0.82
(same sender)	C,V,I,B	0.85	0.91	0.81
with FF	C,V	0.72	-	-
(different senders)	C,V,I,B	0.73	-	-

***C = Cubic, V = Vegas, I = Illinois, B = BBR, FF = Fairness Framework**

(a) **Fairness with equal RTT** ($\Delta RTT = 0ms$).

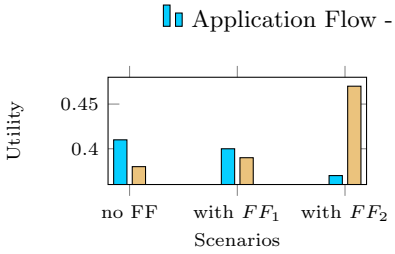
Scenario	Protocol Set	Net Fairness	Learnt Fairness	Learning Fairness
no FF	C,V	0.75	-	-
	C,V,I,B	0.72	-	-
with FF	C,V	0.85	0.93	0.76
(same sender)	C,V,I,B	0.84	0.89	0.81
with FF	C,V	0.75	-	-
(different senders)	C,V,I,B	0.71	-	-

***C = Cubic, V = Vegas, I = Illinois, B = BBR, FF = Fairness Framework**

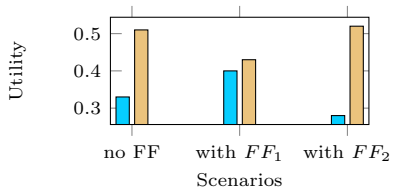
(b) **Fairness with unequal RTT** ($\Delta RTT = 100ms$).

Figure 5.8: **Fairness between two application flows governed by our proposed DPS framework and fairness framework. The darker the shade of the cell, the more preferred is the corresponding fairness value within the cell. The experiment for each combination of variants was repeated 10 times.**

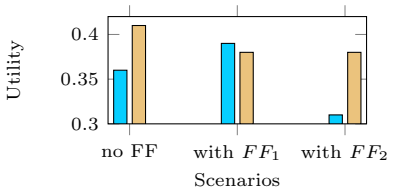
tion performance for both flows every time. This is evident from comparing the average and worst-case (maximum difference between utilities of both flows) utilities for the scenarios. In other scenarios (no FF and FF_2), the utilities between the flows frequently fluctuate due to relatively low fairness, resulting in much higher or inversely lower performance in either of the flows.



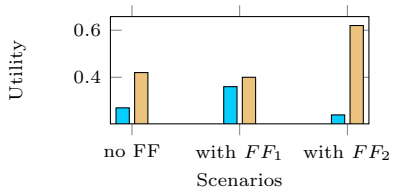
(a) **Equal RTT ($\Delta RTT = 0ms$). 2 protocol set configuration. Average performance.**



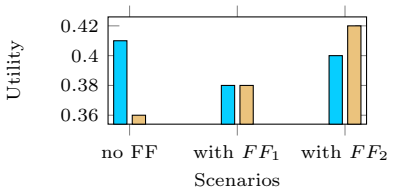
(b) **Equal RTT ($\Delta RTT = 0ms$). 2 protocol set configuration. Worst-case performance.**



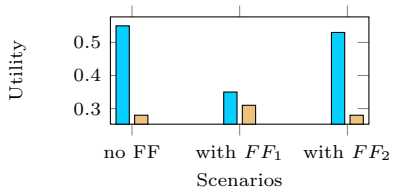
(c) **Equal RTT ($\Delta RTT = 0ms$). 4 protocol set configuration. Average performance.**



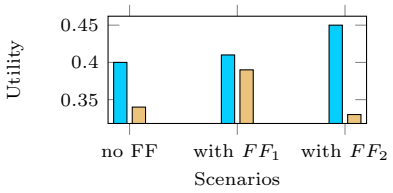
(d) **Equal RTT ($\Delta RTT = 0ms$). 4 protocol set configuration. Worst-case performance.**



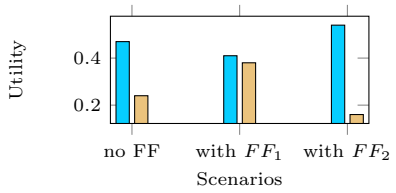
(e) **Unequal RTT ($\Delta RTT = 100ms$). 2 protocol set configuration. Average performance.**



(f) **Unequal RTT ($\Delta RTT = 100ms$). 2 protocol set configuration. Worst-case performance.**



(g) **Unequal RTT ($\Delta RTT = 100ms$). 4 protocol set configuration. Average performance.**



(h) **Unequal RTT ($\Delta RTT = 100ms$). 4 protocol set configuration. Worst-case performance.**

***FF_1 = with FF (same sender), FF_2 = with FF (different senders)**

Figure 5.9: **Plots representing application flow performance (in terms of utility) for both the flows governed by our proposed DPS framework and fairness framework in different configurations. The plots to the left represent the average performance and those to the right represent worst-case performance. The experiment for each configuration was repeated 10 times.**

Chapter 6

Conclusion

This chapter summarizes our work in this thesis by briefly describing the solution to the research question stated in Chapter 1. In Section 6.2, it further identifies potential improvements that are currently limiting factors of our work.

6.1 Conclusion

This thesis involved the development of the conceptual building blocks of a Dynamic Protocol Selection framework. Despite of the rich literature highlighting the performance improvements incurred using the Dynamic Protocol Selection concept, through our literature survey, we further identified research gaps in defining a proper conceptual framework for the same and looking into its run-time behavior. Therefore, in this thesis, we proposed a standardized Dynamic Protocol Selection (DPS) framework based on the design principles of a portfolio scheduler. Considering only TCP variants as the candidate protocols of the DPS framework, we, further, made design decisions while defining the constituent components of the DPS framework and in doing so, addressed the main research question: *How can we design an efficient framework for dynamic selection of transport protocols?*

The following pointers briefly describe the solutions to further sub-questions stated in Chapter 1:

1. **How can we design the framework to learn online with optimum efficiency and accuracy?**
   We proposed an online learning strategy in Section 4.5 to address this question. By performing random and efficient selection of protocols at appropriate circumstances, the strategy enables the DPS framework to autonomously learn about the candidate protocols' behavior in any network condition while transferring application data, thus making the framework plug-and-play enabled. We further introduced the concept of *random_trigger* event within this strategy whose probability of occurrence acts as a tool to tune the degree of trade-off between learning time and overall application performance.

2. **How can we design transmission and reception components for the framework for efficient protocol switching during run-time?**

To enable switching among protocols at the transmitter (sender), every time a different TCP variant needs to be implemented, we proposed to terminate the old TCP connection configured with the previous variant and create a new TCP connection with the currently selected variant (Section 4.3.3). We further proposed various switching methods and analyzed their impacts on the application goodput during switching and the switching duration (Section 5.2.3). At the receiver, to preserve the functionalities of a single TCP connection even while switching between two TCP connections (old to new), we proposed a two-buffer mechanism concept in the DPS framework at the receiver that ensures in-order data arrival (Section 4.3.3).

3. **How can we ensure fairness among application flows running over the dynamic protocol selection framework?**
   In Section 4.6, we proposed a Fairness Framework to ensure an acceptable fairness between DPS framework governed application flows (DPS flows). This framework behaves as an entity above the DPS frameworks which observes the behavior of all the DPS flows and indicates the flows in case of any fairness violations. The superior fairness performance ($\approx 15\%$) obtained with the support of this Fairness Framework (Section 5.4), further showcased the importance and validity of the framework.

6.2 Future Work

There are plenty of opportunities for potential improvements to our proposed work.

- **Switching Methodology:** In our work, we analyzed various protocol switching methods based on the ways in which we vary packet transmission rates in the new TCP connection during the switching period. However, in all of these methods, we initiated data transmission in the new connection immediately after its creation due to which data from both the old as well as new TCP connection co-existed during the entire switching period. However, if we somehow manage to track the length of the packet buffer of the old connection in real-time, based on the additional knowledge about the remaining data to be transmitted in the old connection, the new connection can decide on when to initiate its own transmission so as to reduce the time duration of packets from both connections to co-exist. This would prove to be a highly efficient switching method.

- **Monitoring Component:** Currently, the monitoring component in our proposed DPS framework is not able to identify network states i.e., there is no mechanism for the framework to detect a change in the network condition. The only way we were able to use the concept of network state was in the functioning of the fairness framework, where the framework feeds a DPS flow with the network state that is basically the set of protocols the neighboring flows are locked-in with. To identify a network condition (state), it is important to decide on ways to distinguish between different network conditions. This would further lead to another question on what network parameters to consider to distinguish.

- **Fairness Framework:** Our proposed Fairness Framework currently operates specifically when multiple DPS flows originate from the same client. However, this severely limits its application in a realistic network where multiple DPS flows can originate from multiple clients. One way to approach this issue is to locate this Fairness Framework at the bottleneck routers (switches) in the network which would manage the encountered DPS flows by interacting with the corresponding senders and further indicate the corresponding flows on detecting fairness violations similar to its current operating principle. Although this method would introduce some communication overheads while interacting with the flows' senders, but the potential benefits would easily outweigh this minor drawback.

Bibliography

[1] Transmission Control Protocol. RFC 793, September 1981.

[2] Common object request broker architecture. `https://www.omg.org/spec/CORBA/2.1/About-CORBA/`, Sept. 1997. Last accessed: Aug. 21, 2021.

[3] Admin AfterAcademy. Graph traversal: Depth first search. `https://afteracademy.com/blog/graph-traversal-depth-first-search`, Feb. 2020. Last accessed: Aug. 21, 2021.

[4] Arun Agarwal, Gourav Misra, and Kabita Agarwal. The 5th generation mobile wireless networks- key concepts, network architecture and challenges. *American Journal of Electrical and Electronic Engineering*, 3:22–28, 03 2015.

[5] R. R. Ager. Multiple objective decision-making using fuzzy sets. *Int. J. Man Mach. Stud.*, 9:375–382, 1977.

[6] Azzedine Boukerche and Haifa Maamar. An efficient hybrid multicast transport protocol for collaborative virtual environment with networked haptic. In *2006 IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE 2006)*, pages 78–83, 2006.

[7] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, page 24–35, New York, NY, USA, 1994. Association for Computing Machinery.

[8] S. Burleigh, M. Ramadas, and S. Farrell. Licklider transmission protocol - motivation. *RFC*, 5325:1–23, 2008.

[9] Carlo Caini, Rosario Firrincieli, and Daniele Lacamera. The tcp "adaptive-selection" concept. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 5026–5030, 2007.

[10] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016.

[11] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

[12] P.M.L. Chan, Y.F. Hu, and R.E. Sheriff. Implementation of fuzzy multiple objective decision making algorithm in a heterogeneous mobile environment. In *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609)*, volume 1, pages 332–336 vol.1, 2002.

[13] Craig Hunt. Tcp/ip network administration, 3rd edition. `https://www.oreilly.com/library/view/tcpip-network-administration/0596002971/ch01.html`. Last accessed: Aug. 21, 2021.

[14] D. Wing and A. Yourtchenko. Happy eyeballs: Success with dual-stack hosts. `http://www.ietf.org/rfc/rfc6555.txt`, Apr. 2012. Last accessed: Aug. 21, 2021.

[15] John Day and Hubert Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71:1334 – 1340, 01 1984.

[16] Kefeng Deng, Junqiang Song, Kaijun Ren, and Alexandru Iosup. Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds. page 55, 11 2013.

[17] Kefeng Deng, Ruben Verboon, Kaijun Ren, and Alexandru Iosup. A periodic portfolio scheduler for scientific computing in the data center. In Narayan Desai and Walfredo Cirne, editors, *Job Scheduling Strategies for Parallel Processing*, pages 156–176, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[18] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association.

[19] Ali Dorri, Salil Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:1–1, 04 2018.

[20] Thomas Dreibholz, Erwin P. Rathgeb, Irene Rüngeler, Robin Seggelmann, Michael Tüxen, and Randall R. Stewart. Stream control transmission protocol: Past, current, and future standardization activities. *IEEE Communications Magazine*, 49(4):82–88, 2011.

[21] Falko Dressler, Florian Klingler, Michele Segata, and Renato Lo Cigno. Cooperative driving and the tactile internet. *Proceedings of the IEEE*, 107(2):436–446, 2019.

[22] Qiuyi Duan, Lei Wang, C.D. Knutson, and M.A. Goodrich. Axiomatic multi-transport bargaining: a quantitative method for dynamic transport selection in heterogeneous multi-transportwireless environments. In *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, volume 1, pages 98–105, 2006.

[23] H.R. Duffin, C.D. Knutson, and M.A. Goodrich. Prioritized soft constraint satisfaction: a qualitative method for dynamic transport selection in heterogeneous wireless environments. In *2004 IEEE Wireless Communications*

and Networking Conference (IEEE Cat. No.04TH8733), volume 4, pages 2527–2532 Vol.4, 2004.

[24] Korian Edeline and B. Donnet. A bottom-up investigation of the transport-layer ossification. *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 169–176, 2019.

[25] Alejandra Flechas, Leonardo Gomes, and Paulo Nascimento. The evolution of project portfolio selection methods: from incremental to radical innovation. *Revista de Gestão*, 26, 06 2019.

[26] Sally Floyd. Highspeed tcp for large congestion windows, 2003.

[27] Sally Floyd, Mark J. Handley, and Eddie Kohler. Datagram Congestion Control Protocol (DCCP). RFC 4340, March 2006.

[28] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[29] Mario Hock, Felix Neumeister, Martina Zitterbart, and Roland Bless. Tcp lola: Congestion control for low latencies and high throughput. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 215–218, 2017.

[30] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[31] Kaoshing Hwang, Mingchang Hsiao, Chengshong Wu, and Shunwen Tan. Multi-agent congestion control for high-speed networks using reinforcement co-learning. In Jun Wang, Xiao-Feng Liao, and Zhang Yi, editors, *Advances in Neural Networks – ISNN 2005*, pages 379–384, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[32] Y.G. Iyer, S. Gandham, and S. Venkatesan. Stcp: a generic transport layer protocol for wireless sensor networks. In *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, pages 449–454, 2005.

[33] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[34] Huiling Jiang, Qing Li, Yong Jiang, Gengbiao Shen, Richard O. Sinnott, Chen Tian, and Mingwei Xu. When machine learning meets congestion control: A survey and comparison. *CoRR*, abs/2010.11397, 2020.

[35] Darrin P Johnson, Cesar A.C. Marcondes, and Anders D Persson. Method and system for using bayesian network inference for selection of transport protocol algorithm, U.S. Patent 7 672 240, Jun. 2008.

[36] Naeem Khademi, David Ros, Michael Welzl, Zdravko Bozakov, Anna Brunstrom, Gorry Fairhurst, Karl-Johan Grinnemo, David Hayes, Per Hurtig, Tom Jones, Simone Mangiante, Michael Tuxen, and Felix Weinrank. Neat: A platform- and protocol-independent internet transport api. *IEEE Communications Magazine*, 55(6):46–54, 2017.

[37] Yoon Sang Kim. Surgical telementoring initiation of a regional telemedicine network: Projection of surgical expertise in the wwami region. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, volume 1, pages 974–979, 2008.

[38] C.D. Knutson, H.R. Duffin, J.M. Brown, S.B. Barnes, and R.W. Woodings. Dynamic autonomous transport selection in heterogeneous wireless environments. In *2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No.04TH8733)*, volume 2, pages 689–694 Vol.2, 2004.

[39] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6, 12 2019.

[40] Shao Liu, Tamer Başar, and R. Srikant. Tcp-illinois: A loss- and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation*, 65(6):417–440, 2008. Innovative Performance Evaluation Methodologies and Tools: Selected Papers from ValueTools 2006.

[41] Shenjun Ma, Alexey Ilyushkin, Alexander Stegehuis, and Alexandru Iosup. Ananke: A q-learning-based portfolio scheduler for complex industrial workflows. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 227–232, 2017.

[42] Enrico Masala and Antonio Servetti. Performance vs quality of experience in a remote control application based on real-time 3d video feedback. In *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 28–29, 2013.

[43] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 287–297, New York, NY, USA, 2001. Association for Computing Machinery.

[44] Amjad Mehmood, Abdul Ghafoor, H. Farooq Ahmed, and Zeeshan Iqbal. Adaptive transport protocols in multi agent system. In *Fifth International Conference on Information Technology: New Generations (itng 2008)*, pages 720–725, 2008.

[45] T. Nakajima. Dynamic transport protocol selection in a corba system. In *Proceedings Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000) (Cat. No. PR00607)*, pages 42–51, 2000.

[46] Amol P. Pande and S. R. Devane. Study and analysis of different tcp variants. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–8, 2018.

[47] Li Ping, Lu Wenjuan, and Sun Zengqi. Transport layer protocol reconfiguration for network-based robot control system. In *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.*, pages 1049–1053, 2005.

[48] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[49] Aaron Rosenfeld, Robert Lass, William Regli, and Joseph Macker. Dynamic selection of persistence and transport layer protocols in challenged networks. In *MILCOM 2013 - 2013 IEEE Military Communications Conference*, pages 1470–1475, 11 2013.

[50] Olimpiya Saha and Raj Dasgupta. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, 7, 08 2018.

[51] K. Shibata, K. Okamura, and K. Araki. Design and evaluation of dynamic protocol selection architecture for reliable multicast. In *Proceedings 2002 Symposium on Applications and the Internet (SAINT 2002)*, pages 262–269, 2002.

[52] Hyo-Jeong Shin, Dan Pei, M. Lad, Yanghee Choi, and Lixia Zhang. The impact of multi-homing on network reliability and stability: a case study. In *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, pages 543 – 548, 11 2005.

[53] Simon Kemp. Digital trends 2020: Every single stat you need to know about the internet. `https://thenextweb.com/news/digital-trends-2020-every-single-stat-you-need-to-know-about-the-internet`, 2020. Last accessed: Aug. 21, 2021.

[54] B. Katalin Szabó. Interaction in an immersive virtual reality application. In *2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, pages 35–40, 2019.

[55] Hamed Taherdoost. Decision making using the analytic hierarchy process (ahp); a step by step approach. *International Journal of Economics and Management Systems*, 01 2017.

[56] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, 2006.

[57] A.A. Tarraf, I.W. Habib, and T.N. Saadawi. Reinforcement learning-based neural network congestion controller for atm networks. In *Proceedings of MILCOM '95*, volume 2, pages 668–672 vol.2, 1995.

[58] W. Tian, Mingxing Fan, C. Zeng, Yajun Liu, D. He, and Q. Zhang. Telerobotic spinal surgery based on 5g network: The first 12 cases. *Neurospine*, 17:114 – 120, 2020.

[59] Belma Turkovic, F. Kuipers, and S. Uhlig. Fifty shades of congestion control: A performance and interactions evaluation. *ArXiv*, abs/1903.03852, 2019.

[60] Belma Turkovic and Fernando Kuipers. P4air: Increasing fairness among competing congestion control algorithms. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–12, 2020.

[61] Vincent van Beek, Giorgos Oikonomou, and Alexandru Iosup. Portfolio scheduling for managing operational and disaster-recovery risks in virtualized datacenters hosting business-critical workloads. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 94–102, 2019.

[62] Rakesh Vanzara, Priyanka Sharma, Haresh Bhatt, Sudeep Tanwar, Sudhanshu Tyagi, Neeraj Kumar, and Mohammad Obaidat. Adytia: Adaptive and dynamic tcp interface architecture for heterogeneous networks. *International Journal of Communication Systems*, 32, 11 2018.

[63] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 123–134, New York, NY, USA, 2013. Association for Computing Machinery.

[64] Raul Wirz, Manuel Ferre, Raul Marín Prades, Jorge Barrio, José Claver, and Javier Ortego. Efficient transport protocol for networked haptics applications. In *EuroHaptics*, 06 2008.

[65] Seok Ho Won, Markus Mueck, Valerio Frascolla, Junhyeong Kim, Giuseppe Destino, Aarno Pärssinen, Matti Latva-aho, Aki Korvala, Antonio Clemente, Taeyeon Kim, Il-Gyu Kim, Hyun Kyu Chung, and Emilio Calvanese Strinati. Development of 5g champion testbeds for 5g services at the 2018 winter olympic games. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5, 2017.

[66] Kefan Xiao, Shiwen Mao, and Jitendra K. Tugnait. Tcp-drinc: Smart congestion control based on deep reinforcement learning. *IEEE Access*, 7:11892–11904, 2019.

[67] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524 vol.4, 2004.

Appendix A

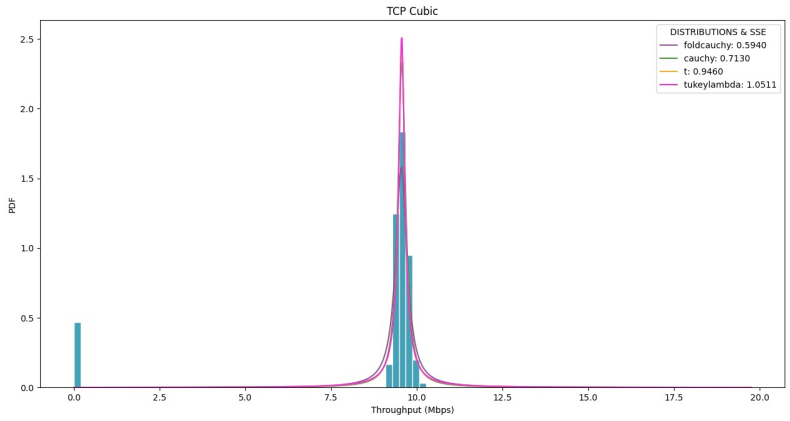
Fitting Parametric Distribution Models

In this chapter, to gain a deeper understanding of the variation in the distribution of RTTs and throughputs of flows deployed with different TCP variants, we present the results from our experiment conducted with one TCP variant from each congestion control categories (Section 2.2.1): Cubic, Vegas, Illinois and BBR respectively. For this experiment, we used a dumbbell network topology with one sender and one receiver emulated in a Mininet[1] environment. The RTT and bottleneck bandwidth between the end-hosts were configured to be 40ms and 10Mbps respectively. Also, the experiments for each TCP variant were run for 250ms.

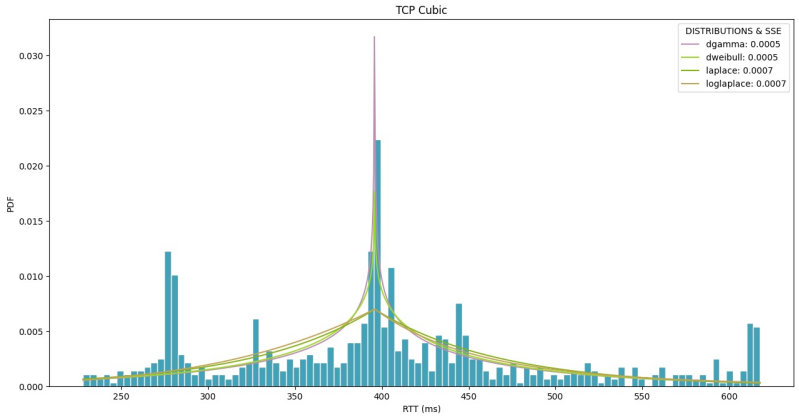
For parametric distribution model fitting, we used *scipy's Statistical functions* module[2], and evaluated fits for 86 different distribution models from the module's library over the probability density functions (PDF) of the RTT and throughput parameters of the considered TCP variants. The following figures sum up our observations and showcases the top 4 model fits for each TCP variant according to a sum-squared error (SSE) performance metric.

[1]Mininet Github Repository: `https://github.com/mininet/mininet`

[2]Scipy's Statistical Functions module: `https://docs.scipy.org/doc/scipy/reference/stats.html`

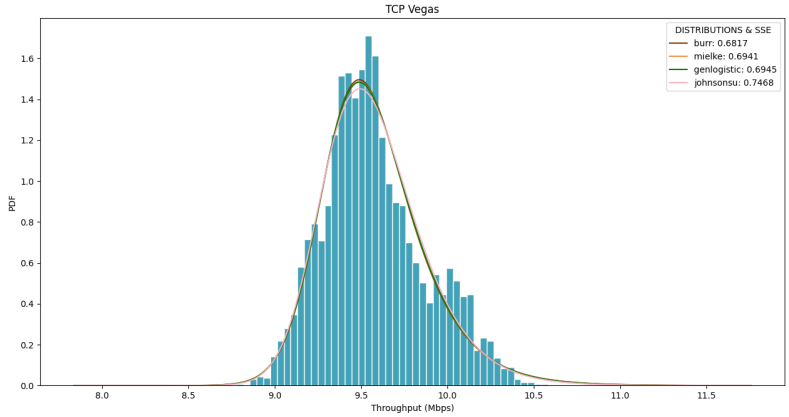


(a) **Parameter under observation: Throughput (Mbps)**

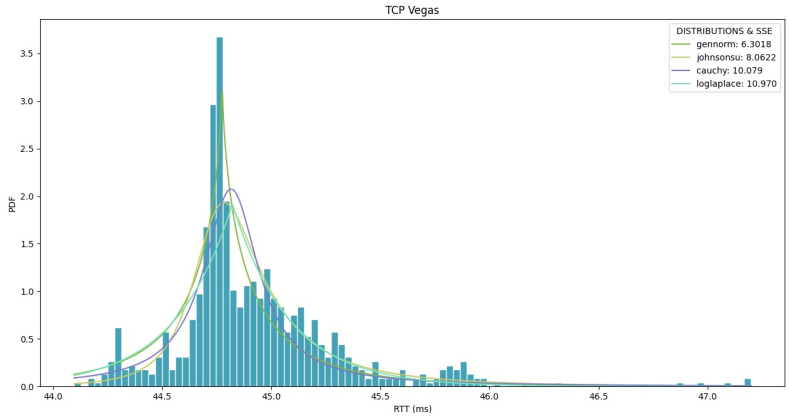


(b) **Parameter under observation: RTT (ms)**

Figure A.1: **The probability density distributions of the *Throughput* and *RTT* obtained on implementing TCP Cubic as the governing transport protocol. The top 4 best parametric distribution model fits are plotted over the original distribution plots.**

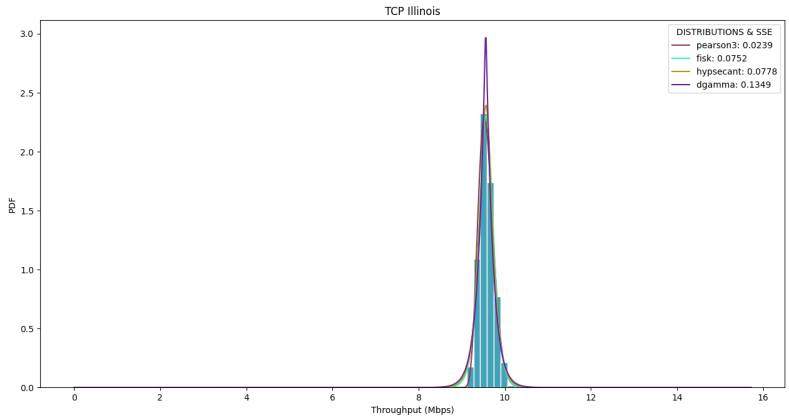


(a) **Parameter under observation: Throughput (Mbps)**



(b) **Parameter under observation: RTT (ms)**

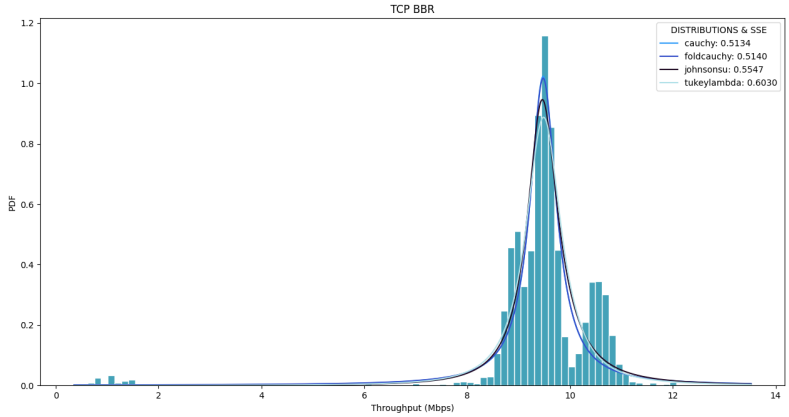
Figure A.2: **The probability density distributions of the *Throughput* and *RTT* obtained on implementing TCP Vegas as the governing transport protocol. The top 4 best parametric distribution model fits are plotted over the original distribution plots.**



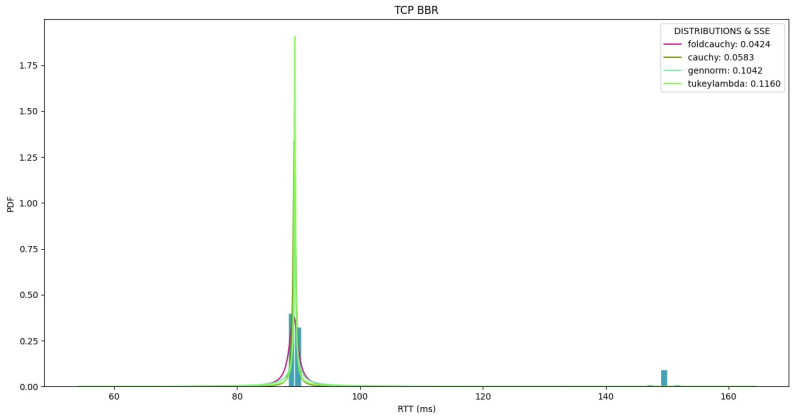
(a) **Parameter under observation: Throughput (Mbps)**

(b) **Parameter under observation: RTT (ms)**

Figure A.3: **The probability density distributions of the *Throughput* and *RTT* obtained on implementing TCP Illinois as the governing transport protocol. The top 4 best parametric distribution model fits are plotted over the original distribution plots.**



(a) **Parameter under observation: Throughput (Mbps)**



(b) **Parameter under observation: RTT (ms)**

Figure A.4: **The probability density distributions of the *Throughput* and *RTT* obtained on implementing TCP BBR as the governing transport protocol. The top 4 best parametric distribution model fits are plotted over the original distribution plots.**