Building Intelligent Network Control Plane

by

Venkatraman Balasubramanian Nolastname

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved August 2022 by the
Graduate Supervisory Committee:

Martin Reisslein, Chair
Antonia Papandreou Suppappola
Akhilesh Thyagaturu
Yanchao Zhang

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

Software Defined Networking has been the primary component for Quality of Service provisioning in the last decade. The key idea in such networks is producing independence between the control and the data-plane. The control plane essentially provides decision making logic to the data-plane, which in-turn is only responsible for moving the packets from source to destination based on the flow-table entries and actions. In this thesis an in-depth design and analysis of Software Defined Networking control plane architecture for Next Generation Networks is provided. Typically, Next Generation Networks are those that need to satisfy Quality of Service restrictions (like time bounds, priority, hops, to name a few) before the packets are in transit. For instance, applications that are dependent on prediction popularly known as ML/AI applications have heavy resource requirements and require completion of tasks within the time bounds otherwise the scheduling is rendered useless. The bottleneck could be essentially on any layer of the network stack, however in this thesis the focus is on layer-2 and layer-3 scheduling. To that end, the design of an intelligent control plane is proposed by paying attention to the scheduling, routing and admission strategies which are necessary to facilitate the aforementioned applications requirement. Simulation evaluation and comparisons with state of the art approaches is provided with reasons corroborating the design choices. Finally, quantitative metrics are defined and measured to justify the benefits of the designs.

"All that I am, and all that I ever hope to be, I owe it to my mother."

To my Amma and Appa

ACKNOWLEDGEMENTS

I would like to thank the thesis committee. My advisor Dr. Martin Reisslein, Dr. Antonia Papandreou-Suppappola, Dr. Yanchao Zhang and Dr. Akhilesh Thyagaturu.

Additionally, I would like to thank my lab-mates and friends here Parth, Dheeraj, Lee, Prateek, Choi, especially Ahmed Nasrallah and Mu Wang who have made me better at this craft of development and research. Through this PhD, I was able to work and contribute to various industry groups like Samsung Semiconductors Inc. and Nokia Bell Labs, a huge shout out to the teams there. I am grateful to Nithya Ramakrishnan and Dr. Zhengyu Yang from Samsung Semiconductors Inc (currently ByteDance) who put me on the industrial R&D map by supporting me in my prototyping and patenting activities. Cheers to the folks in TU Delft, Netherlands, for all the learning experience. A part of this research has come off of criticisms and rigorous reviews by Dr. Moayad Aloqaily, who has been exploring innovative ideas with me ever since my MSc at uOttawa. None of this would have been possible without the fundamentals I imbibed from Dr. Karmouch, Dr. Heli, Wiji and Faisal at uOttawa. I will always love you guys.

I am grateful to have a bunch of supportive people around me. One of those namely Alisha Kulkarni would be signing up to be my wife. I am thankful to Arizona State University to have brought the two of us together. I would be remiss if I don't mention Lynn Pratte, who has been more than just an academic advisor to me. Having lost my father Dr. V. Balasubramanian before I began this PhD journey, there was little motivation or drive left in me to pursue my interests. However, these ladies, Dr. N. Seethalexmy, my mother and Pooja Balasubramanian, my sister have had my back all my life, without their support this PhD thesis wouldn't have come to fruition.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Software defined Networks or SDNs have been regarded as the key component for QoS provisioning. In this paradigm, the decoupling of the control plane from the data-plane allows independent decision making at the controller. Various requirements such as packet delay, time-slot assignments, bandwidth re-allocation, network reconfiguration is supported by the SDN controller application built (as an orchestration or management layer) for providing a software service. In this thesis, we dive deeper into these requirements and provide details about how the provisioning is carried out from abstract concepts to reality for next generation networks. Typically, next generation networks comprises of applications such as ML/AI models that perform bandwidth intensive tasks within a network or within one of the components in a network (namely a server). These applications could be prediction of caching location (such as FedCo), prediction of vehicle locations (such VeNet) or any real time streaming application (such as EdgeBoost). Therefore, the focus of this thesis would be all of these three applications. We carry out extensive mathematical analysis followed by a software simulation design to show the feasibility of making a product in the future. In order to make this thesis accessible to a beginner and an expert in Network Engineering each chapter is organized as follows:

1. Each chapter starts with an introduction where the broader problem and motivation is explained.

2. Each chapter has an extensive literature survey that encompasses the current state of the art models that would help the reader understand how each of these

models work. Some models require re-iteration, ostensibly these frameworks would be mentioned more than once in different chapters.

3. Each chapter then dives into a system model that we design and provide mathematical analysis and subsequent proofs of the same model.

4. Finally, each chapter provides a simulation experiment to coherently present how theory and practical applications come together. Eventually, a few concluding remarks and future work is provided.

Broadly, for QoS provisioning in next generation networks there are key points which we must consider. To that end, this thesis begins with Chapter 1 where-in a hot take on Time sensitive networks (TSN) and SDN control plane design for such networks is provided. Continuing in the same vein, in Chapter 2, we consider fault tolerant scenarios for TSN and explore in depth how SDN control plane can make independent decisions to get a network back online in times of emergencies and fault.

In Chapter 3, we discuss FedCo a learning model for content caching which takes inspiration from our previous work EdgeBoost Balasubramanian *et al.* (2019). A key feature in NGN is the need for seamless handover at the edge. We show the control plane of this architecture seamlessly controls mobility handling with fundamental SDN principles. In Chapter 4, we extend the mobility characteristics to Vehicular Networks and perform deep neural networked learning tasks for vehicular networks. Aptly called as VeNet, this model proposes a new paradigm of Internet of Vehicles architecture for producing a seamless automotive system. We use raspberry pi vehicles to perform some key experiments to show its performance benefits.

In Chapter 5, we provide concluding remarks and a few future directions to take this research forward.

Chapter 2

CONTROL PLANE FOR INDUSTRIAL IOT : A TIME SENSITIVE NETWORK PERSPECTIVE

2.1 Introduction

Emerging technologies, such us industrial Internet of Things (IoT), autonomous vehicular networks, and smart healthcare systems, are critical time-sensitive applications Al Ridhawi *et al.* (2018); Gavriluţ and Pop (2020); Otoum *et al.* (2018); Ridhawi *et al.* (2020). Machines in industrial automation have started to implement time-sensitive control loops that require a strict delay bound. With the advent of Industry 4.0 and the Industrial IoT the new challenge of managing the time-sensitive traffic has arisen Baccarelli *et al.* (2017); Raptis *et al.* (2019). In order to address the real-time communication needs of these applications, as documented in detail in Belliardi, R., et al. (2018), the IEEE Time Sensitive Network (TSN) Task Group has developed the IEEE 802.1Qbv standard Bello and Steiner (2019). The IEEE 802.1Qbv standard includes the Time Aware Shaper (TAS) which relies on synchronized time cycles and features gate time periods for high-priority scheduled traffic Nasrallah *et al.* (2019). TAS strives to provide on-time packet services. To cope with dynamic network scenarios, various reconfiguration mechanisms need to be considered that optimally allocate network resources. In particular, there is a need to maximize the traffic flow admissibility in the network for a given bounded network capacity. Network Utility Maximization Ferragut and Paganini (2011); Karakoc *et al.* (2020); Wang *et al.* (2020) is a common strategy that has been used over decades for optimal network resource allocation so as to maximize a utility, whereby the network

Quality of Service (QoS) requirements are quantified using a concave function.

While establishing QoS metrics in the network, Software Defined Network (SDN) control has played a key role in recent years. Numerous studies, such as Barakabitze *et al.* (2020); Castillo *et al.* (2020); Gerhard *et al.* (2019); Kellerer *et al.* (2019); Leng *et al.* (2019); Pinheiro *et al.* (2017); Yang and Yeung (2020), have shown varying benefits of remote monitoring of network events. Primarily, the orthogonality of the control-plane and the data-plane enables a programmatic independence that allows efficient resource management. Generally, latency issues can be very well managed via the SDN control strategies Wang *et al.* (2018).

In this paper, we design a reconfiguration framework based on the IEEE 802.1Qbv standard in collaboration with the IEEE 802.1Qcc standard. We use an SDN controller to implement a control mechanism that incorporates the 802.1 Qcc variables (including the flow instantiation parameters and the 802.1Qbv gate control parameters). The SDN controller cohesively communicates with the data-plane TSN switches, which are time synchronized, e.g., via the IEEE 1588 Precision Time Protocol. We reduce the network reconfiguration problem to a network utility maximization problem as the rate stability constraints simultaneously affect the network reconfiguration and utility. We show via gating constraints that the achieved bound ensures a rate-stable physical network. Further, we utilize a virtual queue framework that is software controlled to maximize the network utility in the virtual queues, Finally, we prove that this virtual queue framework provides stability of the physical network. The main contributions of this article are:

i We design the novel TSN_u policy that uses virtual queues and we formulate the utility maximization problem by partitioning the objective function. In doing so, we divide the network functions into three phases, namely admission control, routing, and scheduling.

ii We jointly optimize the admission control, routing, and scheduling in a TSN with gate constraints.

iii We theoretically analyze the proposed TSN_u policy to prove the network stability while maximizing the network utility. We also simulate the proposed TSN_u policy, whereby we use a generic POX controller to build our custom software-defined TSN_u controller. We compare the proposed TSN_u with two state-of-the-art approaches, namely Non-TSN_u Nasrallah *et al.* (2019), virtual queue baseline Neely (2010), and TSSDN* Nayak *et al.* (2017). The presented evaluation results illustrate how TSN_u can facilitate the Industry 4.0 and industrial IoT paradigms.

The remainder of this paper is structured as follows. The related work is reviewed in Section 5.2. Section 3.3 introduces the design and formulation of our TSN_u framework. Sections 3.4 through 2.3.5 present the theoretical analysis, while Section 3.6 presents the simulation evaluation and comparison with state-of-the-art approaches. Finally, Section 3.7, provides concluding remarks and outlines future research directions in the real-time communication domain, such as Industry 4.0.

2.2 Related Work

This section reviews the TSN studies that are closely related to our study. The main distinctions of our study from the existing related studies are summarized in Table 2.1. Farzaneh et al. Farzaneh *et al.* (2016); Farzaneh and Knoll (2017) have proposed a prototypical experimental set-up that is derived from their previous work on logic based modelling. Both studies Farzaneh *et al.* (2016); Farzaneh and Knoll (2017) model a simulation set-up for automating TSN schedules. Both studies ignore the key gate control constraints, i.e., these existing studies do not jointly consider

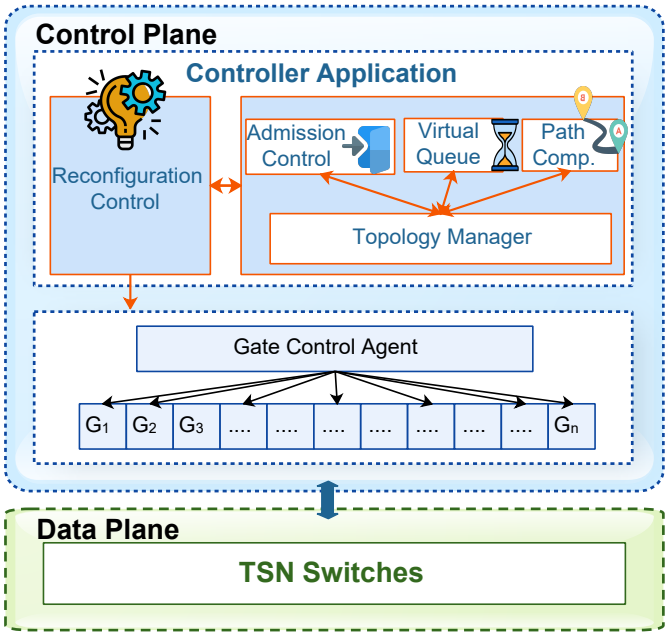


Figure 2.1: Architecture Design of Proposed Sdn-based Tsn_{u} Framework: In The Control-plane, the Reconfiguration Control Interacts with the Topology Manager as Well as Admission Control, Virtual Queue Data Base, And Path Computation to Control the Gate Control Agent (Engine). The Gate Control Engine Creates the Gate Control Entries That in Turn Control The Tsn Switches in the Data-plane

admission control, routing, and scheduling to determine the gate control function. In contrast, we jointly consider admission control, routing, and scheduling for the gate control function which is a key constraint in the TSN utility maximization. Further, we make use of the SDN controller to distribute the time-triggered schedule among the TSN network switches. Primarily, the benefit of using SDN controller in this scenario is for the ease of programmability and dynamic reconfiguration.

Stanton et al. Stanton (2018) have shown the relative benefits of exploiting precisely calculated time synchronization methods. The study Stanton (2018) elaborates the usage of the IEEE 802.1AS standards that define the deterministic sub-

microsecond timing with the Precision Time Protocol (PTP). The authors show how different clock algorithm techniques are ideal to manage the gate control mechanisms of a TSN switch. Stanton et al. Stanton (2018) mainly describe routines to show how shared time can be used in Industry 4.0 scenarios.

Zhao et al. Zhao *et al.* (2018) have proposed a design that inter-twines the priority based scheduling and time triggered credit based shaper mechanism to widen the solution space for gate control list preparation. The worst case latency for individual critical flows or time triggered flows is computed. The study Zhao *et al.* (2018) provides an optimized solution for time triggered networks, however, the complexity of operation and the number of iterations required to produce the gate control list (GCL) is very high. Additionally, Zhao *et al.* (2018) does not simultaneous optimize admission, routing, and scheduling as the main focus was on the schedule preparation in form of the gate control entries. Similarly, Zhang et al. Zhang *et al.* (2019) used network calculus based analysis of TSN for industry 4.0. The analysis in Zhang *et al.* (2019) mainly targets the applications specific to Industry 4.0 which is also one of the integral applications that we consider in this study.

Boehm et al. Boehm *et al.* (2019) have designed a unified controller for time-sensitive networks and SDN. Boehm et al. Boehm *et al.* (2019) built on Nayak *et al.* (2017) and showed that the key difference between the two studies is the combined controller architecture that provides real-time communication support. However, both of these studies did not provide any time synchronization of the SDN switches, which translates as not performing any management of the Gate Control (GC) parameters. Falk et al. Falk *et al.* (2019) have proposed a network simulation framework called $NeSTiNg$ in OMNeT++ that analyzes the behavior of TSN. A similar TSN simulation study has been presented in Nasrallah *et al.* (2019). Falk et al. Falk *et al.* (2019) analyze the different available scheduling mechanisms that produce reliable

Table 2.1: Summary of Comparison with Related Work.

Related Work	Main limitations and differentiation from our study
Farzaneh et al. Farzaneh *et al.* (2016); Farzaneh and Knoll (2017)	Ignores the key gate control constraints
Stanton et al. Stanton (2018)	Mainly prescribes routines to show how shared time can be used in Industry 4.0 scenarios, does not provide any on-line solution for reconfiguration
Zhao et al. Zhao *et al.* (2018)	Complexity of operation and the number of iterations required to produce the GCL are very high
Boehm et al. Boehm *et al.* (2019)	Does not provide any time synchronization to the SDN switches, which translates to not performing any management of the Gate Control (GC) parameters
Said et al. Said *et al.* (2019)	Ignores optimizing the three important aspects of admission control, routing, and scheduling
Nayak et al. Nayak *et al.* (2017)	Exploits the global view of the controller for routing and scheduling, with a pre-assumption of admission

transmission selection, e.g., the credit based shaper and priority queueing. However, both Falk *et al.* (2019); Nasrallah *et al.* (2019) do not consider the existence of a programmable control plane. Also, optimization of routing and scheduling is not considered in the simulation models in Falk *et al.* (2019); Nasrallah *et al.* (2019). In contrast, in this study we develop an optimization framework for a cost effective TSN implementation. Silva et al. Silva *et al.* (2019) have briefly examined the adequacy of integrating both SDN and TSN for Industry 4.0 settings. Silva et al. Silva *et al.* (2019) have outlined the benefits of real time Ethernet protocols in combination with the network programmability of SDN.

Said et al. Said *et al.* (2019) implemented the IEEE 802.1Qcc model using the SDN approach. Said et al. showed how SDN principles can accelerate the time to integrate new flows or configure a TSN network. Said et al. Said *et al.* (2019) consider mostly a model for static scenarios and do not examine in detail how the model can be extended to incorporate dynamic scenarios with real-time computation and decisions. The model ignores optimizing the three important aspects of admission control, routing, and scheduling. On the other hand, our TSN_{u} framework caters to these key constraints and prioritizes the three aspects of admission control, routing, and scheduling for dynamic real-time communication scenarios.

Nayak et al. Nayak *et al.* (2017) have proposed a Time Sensitive Software Defined Network (TSSDN) architecture that is the closest related study to our work. Nayak et al. Nayak *et al.* (2017) exploit the global view of the controller for scheduling and routing with a pre-assumption of admission. Our evaluations indicate that it is necessary to have a joint policy for admission control, routing, and scheduling in order to gain performance benefits. We compare our TSN_{u} approach with the TSSDN approach to quantify the significant benefits produced by our TSN_{u} approach.

We briefly note for completeness that a specific time-triggered traffic management

based on conflict graph techniques has been studied in Atallah *et al.* (2019); Falk *et al.* (2020). Multicast traffic in TSN networks has been studied in Schweissguth *et al.* (2020); Yu and Gu (2020), while our focus is on unicast traffic. Also, joint routing and scheduling in deterministic Internet Protocol (IP) networks with network layer techniques has been studied in Krolikowski *et al.* (2020); in contrast, our focus is on the TSN link layer.

2.3 TSN_u System Design and Formulation

2.3.1 Design

Figure 2.1 shows our proposed control plane design. The control plane is mainly responsible for configuring the forwarding plane (data-plane). The key calculations of the paths taken by the packets and the schedules needed by the switches are prepared in the control-plane. These control actions are conducted remotely i.e., a logically centralized SDN controller with a global view executes the controller application. The OpenFlow protocol McKeown *et al.* (2008) acts as the south-bound interface for communicating with the switches in the data-plane. Our design is mainly divided into six components:

Reconfiguration Control

The reconfiguration control module takes input from the topology manager and the gate control engine. Primarily, the topology manager gives the total number of hops which need to be traversed for a particular reconfiguration to take place; while the gate control engine provides the time-slot within which the reconfiguration should take place. The module executes the online algorithm necessary for dynamic scenarios where each path that has failed or is congested needs a re-computation or a fault-

tolerance based suggestion. The path computation element feeds this information directly to the reconfiguration control module.

Topology Manager

The topology manager mainly handles the switches and the end-hosts. The topology manager relies on the resource list in the reconfiguration control module and enforces the seven hop recommendation of the IEEE 802.1D standards. A typical scenario to consider is as follows: In a common industrial environment, sensors that periodically or sporadically send ambient measurements to a local gateway require certain Quality of Service (QoS) guarantees. In such a volatile and dynamic environment, new machinery that requires prioritized execution (e.g., emergency cooling procedures or maintenance tasks for network traffic tests) may be brought onto the factory floor. To deal with such scenarios, the TSN shapers via Gate Control Lists (GCLs) in coordination with the Network Management Entities (NMEs), e.g., Centralized Network Configuration (CNC), have to adapt to changing environmental conditions by judiciously applying reconfigurations such that stream deadlines, QoS, and total stream utilization times (reported by a stream registration procedure) are satisfied. To support the TSN shapers, the IEEE 802.1AS precision time protocol (PTP) provides a synchronized clock on the end-hosts, while the source nodes can use Intel DPDK modules Redzovic *et al.* (2020); Xiang *et al.* (2019); Xiang *et al.* (2022) and hardware accelerations Niemiec *et al.* (2019); Shantharama *et al.* (2020); Thyagaturu *et al.* (2022b) for low-delay processing of the source applications.

Admission Control

The admission control element is called upon a new stream registration. The admission control supports the topology manager and maintains the information and

variables necessary to determine whether a flow should be accepted or rejected. More specifically, the admission control maintains the gating function $\mathcal{F}_e(t)$ which limits the number of packets that are serviced on a link e in a time-slot t.

Mainly, dynamic scenarios pose difficulties and hence require close attention. When a new flow needs to be scheduled, the source sends a flow request that includes the destination information and the time-stamp indicating when it is transmitting. The admission control module executes the online algorithm that checks whether the flow can be accepted. If accepted, a suitable schedule is prepared with the time-slot and routing information. The controller updates the flow table in the switches for routing the packets and hands over control to the path computation element.

Throughout, the admission decision is based on the currently available time-slots and the available lowest-cost path. Thus, when a given flow requests admission, we first check the time-slot availability on the available lowest-cost path. If all time-slots are filled, then we check the next available lowest-cost path. We continue this process until we have checked all the time-slots on all available paths.

Virtual Queue Database

This module contains the necessary transmission schedule parameters. The virtual queue database is a repository of all the resources available in the form of queues. It maintains the mapping policy of the physical queues which are occupied by the packets that the controller admitted in the time-slots t_1 to t_2, i.e., $\mathcal{Q}(t_1, t_2)$, to the packets admitted to the virtual queue $\hat{\mathcal{Q}}(t_1, t_2)$. The arrival function and its imposed constraints affect the stability of the queues which, which are analyzed in detail in Section 2.3.3.

The virtual queue database module also plays the important role of deleting flows from the schedule once flows expire. The resources are freed for new flows. The

updates are then made via the OpenFlow interface. The prepared global transmission schedule is important because once the choice of the schedule is made, then the time-slot is dedicated to that particular flow and the entire path is reserved, as elaborated in Section 3.4.

Path Computation Element and Gate Control Engine

These two modules are employed together due to the role they play in the architecture. These modules play a vital role in static and dynamic scenarios. Every time a stream is admitted, the path computation element finds equal-cost paths when a path failure or congestion occurs. Further, the routing proceeds differently for both scenarios. In static scenarios, the routes are prepared offline and flow tables can easily be updated with new routes. However, for dynamic scenarios, the routing proceeds in multiple stages. Once the control is handed over from the admission control module, the source is informed about the schedule and the transmission begins. The update is carried via the OpenFlow interface. The necessary time-slots are picked by the Gate Control Agent among the Gate Control Entries defined as $G_1 \dots G_n$. Once the entire path is reserved for a particular flow, no other time triggered scheduled traffic flow will interfere with it. Thus, all flows are isolated from one another. Once a flow expires, a removal request is made and the path computation element relays this information to the virtual queue module to delete the flow from the schedule.

Although any routing algorithm could be employed for the path computation, we employ routing based on maximal edge-disjoint spanning trees. This means that different outgoing edges can distribute packets in the same time-slot. We follow a similar approach as Edmonds's algorithm employing breadth first search (BFS) for selecting the edges. BFS always selects a path with the minimum number of edges. Therefore, the worst case computational complexity of the routing is $O(J^2)$, i.e.,

quadratic in the number J of vertices (TSN switches) in the graph.

The admission control requires a linear/binary search over all the flows (essentially an iteration over the mid-points of the time-slots and a matching with the expected flows that need to use a given time-slot) that is the number of all flows being registered, so that the binary search effort is less than the routing complexity for practical scenarios. Additionally, flow updates, e.g., an already ongoing flow requests a higher bite rate, are similar in that every update requires checking flow time-slots and assigning updated flows to time-slots, which takes linear time. Therefore, the routing complexity dominates the overall framework complexity.

2.3.2 Optimization Problem Formulation

In this section we formulate the TSN reconfiguration problem as a network utility maximization problem. We describe our goal of designing a stable control policy for the virtual queue framework. We relate how this framework can be mapped to a physical network and show rate stability in both physical and virtual queue scenarios. The key factors which we consider in our design are the joint optimization of admission control, routing, and scheduling. In pursuit of our routing policy, we consider a weighted graph, where each edge is weighted by the queue-length; in particular, we follow a weighted shortest path from source to destination. For our admission control, we jointly consider the available routes and the links in the topology, whereby packets are admitted via a minimum-cost function. Finally, the link scheduling policy follows a simple drift minimization strategy per slot.

Table 2.2: Summary of Key Notations.

Notations	Meaning
J	Set of network nodes (TSN switches)
e	Network edge, i.e., direct link between two TSN switches
E	Set of edges in network
$\mathcal{R}$	Set of routes in the network graph $G = (J, E)$
	Timing Structure
t	Time-slot corresp. to gate control entry duration, with a link either ON or OFF during a time-slot.
T	Time horizon available at the controller, which is divided into base periods (TSN cycle times), which are further divided into time-slots
c	Traffic class, where $c = 1$ means ST, $c = 2$ means BE
γ	Instantiation of traffic class $\gamma = c$ for a given flow
$\tau(t)$	Link state process, $\tau(t) \in \{0, 1\}$, i.e., given link is ON or OFF in time-slot t,
$\hat{\mathcal{V}}_e(t)$	Virtual queue length (in number of packets) in link e in time-slot t
$\mathcal{V}_e(t)$	Physical queue length, i.e., number of packets waiting to traverse edge e
$\mathcal{F}(t)$	Gating function (packets/time-slot) for a given flow in time-slot t.
$C_e(t_1, t_2)$	Service allocation (Gbit/s) to link e between time-slots t_1, t_2
$\hat{\mathcal{Q}}_e(t)$	Total number of packets the controller allows in the link e in time-slot t
$\nabla(T)$	Rate in packets/time-slot number of packets received by a given destination up to end time T
$\tau(t)p_e a_e$	Service rate for virtual queue of link e in time-slot t
$\mathcal{Z}$	Utility function for TSN network

Preliminaries

We first note that our problem constraints form a closed and bounded set and that our objective function is strictly concave. In particular, the rate stability constraints form a closed set as the aggregate rate on a link is bounded by the finite link capacity and every queue length is bounded (packets arriving to a queue filled to capacity are dropped). Also, the objective function is strictly concave because the second derivative of the objective function is strictly negative; intuitively, the rate of effect on the objective function of a link capacity change from, e.g., 1 kbps to 100 kbps is much higher than the increase from 1 Mbps to 1.1 Mbps. Our maximization problem thus satisfies the requirements for obtaining a unique maximum.

Consider a network graph $G(J, E)$, with J as the set of nodes and E as the links, see Table 5.2 for a summary of the main notations. The time scale is slotted according to the TAS specifications into periodic base periods (TSN cycle times) which are further partitioned into time-slots. A link is either ON or OFF with binary states 1, 0, respectively, in a given time-slot. A time-slot can only transmit if the link is ON. Further, external packets from the Scheduled Traffic (ST) and Best Effort (BE) Traffic are admitted via the IEEE 802.11Qcc controller $\mathcal{Q}$. Consider the traffic class to be represented as c, whereby c is either ST or BE. We define a monotonically increasing utility function $\mathcal{Z}$. Consider the set of all routes that a given packet from a given traffic class γ as an instantiation of the traffic class c takes in the graph G represented as $\mathcal{R}$. For our network, we consider an admission controller $\mathcal{Q}$ that has unlimited number of packets from traffic class γ. This controller determines packets to be allowed into the network from each traffic class in the time-slot it has been provided to transmit on a per-flow basis. Intuitively, at most $\mathcal{Q}(t)$ number of packets are allowed into the network in each time-slot t.

Utility Maximization Problem

Let there be Π admissible policies. The utility maximization problem finds the policy $\pi \in \Pi$ that maximizes the total of the utilities of all the traffic classes while maintaining the rates of transmission in the network to be stable. Let the rate of transmission be $\nabla(T)$, for a given traffic class γ in a policy π transmitting for a time period T. Further, let the queue length at link e be denoted by $\mathcal{V}_e(T)$. Thus, the utility maximization problem can be given as

$$\max_{\pi} \mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(r_\gamma)\right] \tag{2.1}$$

with the two constraints

$$\lim_{T\to\infty} \nabla(T)/T = r_\gamma \tag{2.2}$$

$$\lim_{T\to\infty} \sum_{e} \mathcal{V}_e(T)/T = 0 \quad \forall e \in E. \tag{2.3}$$

Note that Equation (2.2) can be interpreted as how fast the transmission needs to be conducted and Equation (2.3) can be interpreted as the rate at the which the queue is being serviced. Correspondingly, the lower the congestion, the lower the delay.

2.3.3 TSN Virtual Queue Framework

We design a TSN virtual queue system that follows an optimal stability policy and then map the virtual queue system to the physical network system. Suppose the controller admits $\mathcal{Q}(t)$ packets from one of the classes c, that activates the link τ at one time-slot t. The service rate for the virtual queue, on the link demarcated by a random stationary ergodic process $a_e \in \{0, 1\}$ that is given as $\hat{\mathcal{V}}_e$ is $\tau(t)p_e a_e$ packets per time-slot, where in a time-slot t, on a link e, we can transmit p_e packets when the link is ON. We assume that all the packets enter the virtual queues immediately

on the selected path. Hence, the growth of the virtual queue can be given as

$$\hat{\mathcal{V}}_e(t+1) = \hat{\mathcal{V}}_e(t) + \hat{\mathcal{Q}}_e(t) - \tau(t) p_e a_e, \tag{2.4}$$

whereby $\hat{\mathcal{Q}}_e(t)$ represents total number of packets the controller has allowed in the queue in the time-slot t, which in turn is related to the route $\mathcal{R}$. For designing a stability control policy, we utilize the drift-plus-penalty function Neely (2010). According to the drift-plus-penalty function, we can form a Lyapunov function for virtual queues by directly mapping the physical queue length Neely (2010):

$$L(\hat{\mathcal{V}}(t)) = \sum_e \hat{\mathcal{V}}_e^2(t) \tag{2.5}$$

with the conditional drift

$$\Delta(\hat{\mathcal{V}}(t), a(t)) = \mathbb{E}\left[L(\hat{\mathcal{V}}(t+1)) - L(\hat{\mathcal{V}}(t)) \mid \hat{\mathcal{V}}(t), a(t) \right]. \tag{2.6}$$

Under the policy conditions,

$$\Delta(\hat{\mathcal{V}}(t), a(t)) \leq \mathbb{E}\left[\hat{\mathcal{Q}}_e^2(t) + p_e^2 + 2\hat{\mathcal{V}}_e(t)(\hat{\mathcal{Q}}_e(t) - \tau(t) p_e a_e(t)) \mid \hat{\mathcal{V}}(t), a(t)) \right]. \tag{2.7}$$

The terms $\hat{\mathcal{Q}}_e^2$ and p_e^2 are finite constants, upper bounded by $\mathcal{Q}_{\max}^2 + p_{\max}^2$; we denote $\mathcal{H}$ for this upper bound. Thus, the right-hand side of (2.7) becomes

$$= \mathcal{H} + 2\mathbb{E}\left[\sum_e \hat{\mathcal{V}}_e(t)(\hat{\mathcal{Q}}_e(t)) \mid \hat{\mathcal{V}}(t), a(t) \right] - 2\mathbb{E}\left[\sum_e \hat{\mathcal{V}}_e(t)\tau(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t) \right]. \tag{2.8}$$

Interchanging summation gives then for right-hand side of (2.7):

$$= \mathcal{H} + 2\mathbb{E}\left[\sum_\gamma \hat{\mathcal{Q}}_\gamma(t) \sum_e \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c(t)\}} \mid \hat{\mathcal{V}}(t), a(t) \right] - 2\mathbb{E}\left[\sum_e \hat{\mathcal{V}}_e(t)\tau(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t) \right],$$

whereby $1_{\{\cdot\}}$ denotes the characteristic indicator function which is one if the argument is true and zero otherwise. As, $\hat{\mathcal{Q}}_e(t)$ depends on the routes selected for the traffic classes γ, we re-write the controller term as,

$$\hat{\mathcal{Q}}_e(t) = \sum_{\gamma} \hat{\mathcal{Q}}_\gamma(t) 1_{\{e \in \Psi_c(t)\}}. \tag{2.9}$$

Thus, neglecting the constant $\mathcal{H}$, the objective is to minimize the function

$$2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma(t) \sum_{e} \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c(t)\}} \mid \hat{\mathcal{V}}(t), a(t)\right] - 2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t)\tau(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t)\right] - 2J \sum_{\gamma} \mathcal{Z}_\gamma(\hat{Q}_\gamma(t)). \tag{2.10}$$

The above equation is obtained following the drift-plus penalty framework that considers admission control, routing, and scheduling. On minimizing this equation we obtain our TSN_u policy that jointly considers both admission control and routing jointly while enforcing rate stability.

2.3.4 Partitioning the Objective Function

The first term of Eqn. (2.10) represents the routing policy that considers the traffic flow originating from a source node s to a sink node t. Consider a graph $\hat{G}$ of virtual queues, each link e with a virtual queue length $\hat{\mathcal{V}}_e(t)$. Here, all traffic classes $\gamma \in c$ are assigned to the shortest route $\tau(t) \in R$ that gives the paths in the graph $\hat{G}$.

The second term of Eqn. (2.10) represents the link scheduling policy. When this term is minimized, we get the link scheduling policy that activates all links that are in the ON state. The first two terms of Eqn. (2.10) when considered jointly provide the admission control policy:

$$2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma(t) \sum_{e} \hat{\mathcal{V}}_e(t)(e \in c^\gamma(t)) \mid \hat{\mathcal{V}}(t), a(t)\right] - 2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t)\tau(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t)\right]. \tag{2.11}$$

This leads to the first theorem, which claims that under the TSN_u policy, the TSN virtual queues are rate-stable and the expected utility is close to optimal.

Theorem 1: *Let $\mathcal{Z}^*$ be the optimal utility attainable for some policy $\pi \in \Pi$, then the TSN$_u$ policy achieves a utility of at least $\mathcal{Z}^* - \mathcal{O}\left(J^{-1}\right)$ for a positive constant J.*

Proof: From Neely (2010), clearly a random stationary policy called RND exists for a small $\epsilon > 0$ that takes random actions of (network admissibility and scheduling) based on the current network link status $a(t)$ and gives results that are close to optimal given as

$$\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma^{RND}(t))\right] \geq \mathcal{Z}^* - \epsilon \tag{2.12}$$

$$\mathbb{E}\left[\hat{\mathcal{Q}}_e^{RND}(t) - \tau(t) p_e^{RND}(t) a_e(t)\right] \leq \epsilon, \ \forall e \in E. \tag{2.13}$$

These expectations are taken over the network process $a(t)$ and the randomized actions in RND. Now, applying the same approach in our case gives for the conditional drift from Eqn. (2.6):

$$\Delta(\hat{\mathcal{V}}(t), a(t)) - J \sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t)) \leq$$

$$\mathcal{H} + 2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma^{\text{TSN}_\text{u}}(t) \sum_{e} \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c^{\text{TSN}_\text{u}}(t)\}} \mid \hat{\mathcal{V}}(t), a(t)\right]$$

$$-2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t) \tau^{\text{TSN}_\text{u}}(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t)\right] \tag{2.14}$$

and subsequently,

$$-J \sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma^{\text{TSN}_\text{u}}(t)) \ \leq \ \mathcal{H} + 2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma^{RND}(t) \sum_{e} \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c^{\text{RND}}(t)\}} \mid \hat{\mathcal{V}}(t), a(t)\right] \tag{2.15}$$

$$-2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t) \tau^{RND}(t) p_e a_e(t) \mid \hat{\mathcal{V}}(t), a(t)\right] - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma^{RND}(t)) \mid \hat{\mathcal{V}}(t), a(t)\right]. \tag{2.16}$$

The second inequality follows from the TSN_u construction in Section 2.3.3, which is required mainly to minimize the upper bound on the conditional drift plus penalty. The policy π^{RND} is independent of the virtual queue length $\hat{\mathcal{V}}(t)$. The conditioning on the queue length can therefore be ignored; thus, for the remaining term:

$$\Delta(\hat{\mathcal{V}}(t), a(t)) - J \sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t)) \leq \mathcal{H} + 2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma^{RND}(t) \sum_{e} \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c^{\text{RND}}(t)\}} \mid a(t)\right] - 2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t) \tau^{RND}(t) p_e a_e(t) \mid a(t)\right] - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma((Q_\gamma^{RND}(t)) \mid a(t)\right]. \quad (2.17)$$

Taking expectation of the inequality over the network process $a(t)$,

$$\Delta(\hat{\mathcal{V}}(t)) - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t)) \mid \hat{\mathcal{V}}(t)\right] \leq \mathcal{H} + 2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_\gamma^{RND}(t) \sum_{e} \hat{\mathcal{V}}_e(t) 1_{\{e \in \Psi_c^{\text{RND}}(t)\}}\right] - 2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_e(t) \tau^{RND}(t) p_e a_e(t)\right] - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma^{RND}(t))\right]. \quad (2.18)$$

Rearranging the right-hand side,

$$\Delta(\hat{\mathcal{V}}(t)) - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t) | \hat{\mathcal{V}}(t)\right] \leq \mathcal{H} + 2(\sum_{e} \hat{\mathcal{V}}_e(t) \mathbb{E}\left[\hat{\mathcal{Q}}_e^{RND}(t) - \tau^{RND}(t) p_e a_e(t)\right] - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma^{RND}(t))\right]. \quad (2.19)$$

Based on the properties of π^{RND},

$$\Delta(\hat{\mathcal{V}}(t)) - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t)) \mid \hat{\mathcal{V}}(t)\right] \leq \mathcal{H} + 2\epsilon \sum_{e} \hat{\mathcal{V}}_e(t) - J\mathcal{Z}^* + J\epsilon. \quad (2.20)$$

For ϵ tending to zero,

$$\Delta(\hat{\mathcal{V}}(t)) - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t)) \mid \hat{\mathcal{V}}(t)\right] \leq \mathcal{H} - J\mathcal{Z}^*. \quad (2.21)$$

Taking expectation over $\mathcal{V}(t)$,

$$\mathbb{E}\left[L(\hat{\mathcal{V}}(t+1))\right] - \mathbb{E}\left[L(\hat{\mathcal{V}}(t))\right] - J\mathbb{E}\left[\sum_{\gamma} \mathcal{Z}_\gamma(Q_\gamma(t))\right] \leq \mathcal{H} - J\mathcal{Z}^*. \quad (2.22)$$

For a performance guarantee over a time horizon T, we divide both sides by T,

$$\frac{1}{T}\mathbb{E}\left[L(\hat{\mathcal{V}}(t))\right] \leq \mathcal{H} + \frac{1}{T}\mathbb{E}\left[L(\hat{\mathcal{V}}(0))\right] + J\left(\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}(\mathcal{Q}_{\gamma}(t))\right] - \mathcal{Z}^{*}\right) \quad (2.23)$$

By the non-negativity of the Lyapunov function,

$$\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}\left(\frac{1}{T}\sum_{t=1}^{T}\mathcal{Q}_{\gamma}(t)\right)\right] \geq \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}(\mathcal{Q}_{\gamma}(t))\right]. \quad (2.24)$$

Thus,

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}(\mathcal{Q}_{\gamma}(t))\right] \geq \mathcal{Z}^{*} - \frac{\mathcal{H}}{J}. \quad (2.25)$$

Given the optimal achievable utility $\mathcal{Z}^{*}$, and applying Jensen's inequality to $\mathcal{Z}_{\gamma}$,

$$\lim_{T\to\infty}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}(\mathcal{Q}_{\gamma}(t))\right] \leq \lim_{T\to\infty}\mathbb{E}\left[\sum_{\gamma}\mathcal{Z}_{\gamma}\left(\frac{1}{T}\sum_{t=1}^{T}(\mathcal{Q}_{\gamma}(t)\right)\right] \leq \mathcal{Z}^{*}. \quad (2.26)$$

Hence, the optimality of the utility is proven.

Applying the limits to Inequality (2.21),

$$\lim_{T\to\infty}\sup\frac{1}{T}\mathbb{E}\left[L(\hat{\mathcal{V}}(T))\right] \leq \mathcal{H}. \quad (2.27)$$

The drift-plus penalty framework $L(\cdot)$ implies that $L(\hat{\mathcal{V}}(t)) = \sum_{e}\hat{\mathcal{V}}_{e}^{2}(t)$. Thus,

$$\lim_{T\to\infty}\frac{1}{T}\mathbb{E}\left[L(\hat{\mathcal{V}}^{2}(T)\right] \leq \mathcal{H} \text{ which gives } \mathbb{E}\left[L(\hat{\mathcal{V}}^{2}(T))\right] \leq \mathcal{H}. \quad (2.28)$$

From Jensen's inequality, further,

$$\mathbb{E}\left[\hat{\mathcal{V}}(T)\right] \leq \sqrt{B}. \quad (2.29)$$

The upper bound of each virtual queue is given by the maximum amount $Q_{\max}$ of packets that the controller allows. Suppose for a positive β, we set $\beta Q_{\max}$. Thus, for a time horizon T,

$$\hat{\mathcal{V}}(T) \leq \beta Q_{\max} T. \quad (2.30)$$

As Equation (2.30) provides a uniform bound, the bounded convergence theorem applies. Hence,

$$\mathbb{E}\left[\lim_{T\to\infty}\frac{\hat{\mathcal{V}}(T)}{T}\right] = \lim_{T\to\infty}\frac{\mathbb{E}\left[\hat{\mathcal{V}}(T)\right]}{T} = 0. \quad (2.31)$$

2.3.5 *Stability of Physical Time Sensitive Networks based on Gate Control List Constraints*

Intuitively, the number of packets admitted in the virtual network is the same as in the physical network. Further, the rules followed for the scheduling and routing are the same in both networks. The key difference, however, in the physical network is the stability. Unlike the virtual queues, the physical queues are subjected to the constraints of the TSN network. Mainly, the Gate Control List determines the packets that will traverse a TSN switch. The determined routes allow the total number of physical packets for a time interval between $(t_1, t_2]$) to be $\mathcal{Q}(t_1, t_2)$. This corresponds to the virtual queue size $\hat{\mathcal{Q}}(t_1, t_2)$. Now, if a service $\hat{C}(t_1, t_2)$ is granted to the virtual queue, then we apply the Skorokhod mapping policy Ramanan *et al.* (2008):

$$\hat{\mathcal{V}}(t) = \sup_{0 \leq t_1 \leq t} (\hat{\mathcal{Q}}(t_1, t) - \hat{C}(t_1, t)). \tag{2.32}$$

As the arrivals are equivalent,

$$\mathcal{Q}(t_1, t_2) = \hat{\mathcal{Q}}(t_1, t_2) \text{ and } C(t_1, t_2) = \hat{C}(t_1, t_2). \tag{2.33}$$

For a gating function $\mathcal{F}(t)$, the arrival function can be given as,

$$\mathcal{V}_e(t) \leq C_e(t) + \mathcal{F}(t). \tag{2.34}$$

The packet scheduling rules in TSN networks Nasrallah *et al.* (2019); Nasrallah *et al.* (2019) resolve contention at the time of resource allocation. These are also the rules that act as the major constraints in the physical network queues. Similar to a first-in-first-out (FIFO) schedule in traditional networks, the gating mechanisms governs the packet scheduling in the TSN network.

Definition 1: Given a time-based signal (timer), the gating function $\mathcal{F}(t)$ enables or disables a queue from transmission. The current window slot is calculated based

on the global time. In accordance to the calculated slot we check if the current time belongs to the traffic classes (ST or BE).

Definition 2: In the two traffic classes, we produce distance based approximations for the two classes to create a fine-grained policy for the TSN network. To this end, we prioritize packets based on decreasing order of their distance from the source. Based on this definition, if a packet traverses more hops than another packet, then the scheduling policy prioritizes the packet with fewer hops.

Combining the above two definitions, gives the following Lemma:

Lemma 1: *A non-negative function called the gating function $\mathcal{F}(t)$ bounds the total number of packets serviced, which exists for every chosen TSN path Ψ. The definition of $\mathcal{F}(t)$ is given by $\sup_{0\leq t'\leq t}\mathcal{F}(t')$, whereby, $\mathcal{F}(t)$ is monotonically increasing in t and acts as a bound.*

Proof: As $\mathcal{F}(t)$ is non-decreasing, we can define another function $\mathcal{G}(t) = \sup_{0\leq t'\leq t}\mathcal{F}(t')$. Thus, the claim that $\mathcal{F}(t)$ is bounded follows.

Theorem 2: *The physical queues are stable when the TSN$_u$ policy is applied. The rate-stability is given as:*

$$\lim_{T\to\infty}\sum_{e}\frac{\mathcal{V}_e(t)}{T}. \tag{2.35}$$

Proof: Suppose $P_e(0)$ is the total number of packets taking link e in the time-slot $t = 0$. Suppose that the destination of these $P_e(0)$ packets is h hops away, we consider the number $P_h(t)$ of packets that are destined h hops away in time-slot t. If there are multiple such packets sent or if there are duplicates created along the way, we have to sum as

$$P_h(t) = \sum_{\Psi} P_{e,\Psi}(t), \tag{2.36}$$

whereby $P_{e,\Psi}(t)$ represents the total number of packets traversing link e via the routing branch Ψ. We now use induction to prove the bounds.

If $h = 0$, for a specific link e at time $t = 0$, then $\mathcal{V}_e(0)$. The total number of packets serviced by the network is given as $C_e(t_0, t) + \mathcal{F}(t)$. As the gates follow a non-decreasing service, these are the packets which would need some link capacity on link e in the future. Based on the distance approximation, we prioritize $h = 0$ packets. The link e initially had some packets to transmit, which appear as the serviced packets $C_e(t_0, t)$. Thus, the total number of packets in link e is given as

$$\sum_T P_{e,T} \leq P_e(0) + C_e(t_0, t) + \mathcal{F}(t) - C_e(t_0, t) \leq P_e(0) + \mathcal{F}(t). \tag{2.37}$$

For $h = h - 1$, we have a monotonically non-decreasing function $N(t)$, such that $N_i(t) = \mathcal{F}(t)$, where i increases from $0, \ldots, h-1$, we show that $P_i(t) \leq N_i(t)$. Consider the similar case of a link e at an arbitrarily fixed time-slot t, such that $t_o \leq t$, which suggests there are no packets waiting to take the link e. Now, the only packets waiting to traverse the link e are in the hops $0 \leq i \leq h-1$ at time t_0. Based on our assumption, we have the first batch of packets bounded by the monotonically increasing function:

$$\sum_{i=0}^{h-1} N_i(t_0) \leq \sum_{i=0}^{h-1} N_i(t). \tag{2.38}$$

Further, we have the next batch of packets given as $\sum_\Psi \mathcal{Q}_\Psi(t_0, t)$. The TSN_u policy considers the following packets, when there are $h-1$ packets to move in the link e; the following h packets were not processed. This means that within the first $h-1$ hops, there were $\sum_\Psi \mathcal{Q}_\Psi(t_0, t)$ new packets, plus $\sum_{i=0}^{h-1} N_i(t)$ preceding (old) packets. Essentially, the least number of packets going to h hops away from the source is $\max\left\{0,\ C_e(t_0, t) - \sum_{i=0}^{h-1} N_i(t) - \sum_{i=0}^{h-1} \sum_T \mathcal{Q}_T(t_0, t)\right\}$. We can therefore write $P_{e,T}$ as

$$P_{e,T}(t) \leq \sum_{i=0}^{h-1} N_i(t) + \sum_T \mathcal{Q}_T(t_0, t) - \left[C_e(t_0, t) - \sum_{i=0}^{h-1} N_i(t) - \sum_{i=0}^{h-1} \sum_T \mathcal{Q}_T(t_0, t)\right]. \tag{2.39}$$

By further approximation,

$$P_{e,T}(t) \leq 2\sum_{i=0}^{h-1} N_i(t) - \sum_{i=0}^{h-1}\sum_{T} \mathcal{Q}_T(t_0,t) - C_e(t_0,t). \tag{2.40}$$

From Equation (2.32), we can summarize,

$$P_{e,T}(t) \leq 2\sum_{i=0}^{h-1} N_i(t) + \mathcal{F}(t). \tag{2.41}$$

Accordingly,

$$P_h(t) \leq 2\sum_{i=0}^{h-1} N_i(t) + \mathcal{F}(t) \tag{2.42}$$

and

$$N_h(t) = 2\sum_{i=0}^{h-1} N_i(t) + \mathcal{F}(t). \tag{2.43}$$

We observe that the size of the physical queue is given as $\sum_e \mathcal{V}_e(t) = \sum_{h=1}^{k-1} P_h(t)$. From Equation (2.43), for all hops, $P_h(t) \leq N_h(t)$. Thus,

$$\lim_{t\to\infty} \frac{\sum_e \mathcal{V}_e(t)}{t} = 0. \tag{2.44}$$

From, Equations (2.43) and (2.44) and comparing with Neely (2010), we conclude that the physical TSN network is rate stable.

2.4 Performance Evaluation

2.4.1 General Evaluation Set-up

We used Python 3.6 to build our simulator and built a custom controller for replicating the Qcc standard control environment. There are no well-defined queue disciplines available for the Qcc standards, thus we model our own queue structure for the simulation. The physical links have a capacity of 1 Gbps and follow a directional graph (which is the routing flow), see Figure 2.2 for an illustrative example. We set the duration of a base period (TSN cycle time) to 1 ms, with 70 time-slots per base period (unless otherwise noted).

Algorithm 1: TSN_{u}

Input: Initialize all the GCL queues to be zero

For every time-slot,

1) **Topology construction**: Establish the links costs for an incoming request

2) **Routing**: For a unicast traffic $source - sink$ flow, find the minimum weight spanning tree and the weighted shortest path in the network graph G. Update the routing logic based on

$$\sum_{\gamma} \hat{\mathcal{Q}}_{\gamma}(t) \sum_{e} \hat{\mathcal{V}}_{e}(t)(1_{\{e \in \Psi_{c}(t)\}} \mid \hat{\mathcal{V}}(t), a(t)). \tag{2.45}$$

3) **Admission**: Update the following for all admissions:

$$2\mathbb{E}\left[\sum_{\gamma} \hat{\mathcal{Q}}_{\gamma}(t) \sum_{e} \hat{\mathcal{V}}_{e}(t) 1_{\{e \in \Psi_{c}(t)\}} \mid \hat{\mathcal{V}}(t), a(t)\right] - 2\mathbb{E}\left[\sum_{e} \hat{\mathcal{V}}_{e}(t)\tau(t)p_{e}a_{e}(t) | \hat{\mathcal{V}}(t), a(t)\right]. \tag{2.46}$$

4) **Schedule Preparation**:

$$\sum_{e} (\hat{\mathcal{V}}_{e}(t)\tau(t)p_{e}a_{e}(t) \mid \hat{\mathcal{V}}(t), a(t)) \tag{2.47}$$

5) Update the GCQ and forward the packets based on the prepared schedule.

6) Update the virtual queues based on the rate stability of Equation (2.42).

We run our TSN_u policy with a logarithmic utility function $ln(\cdot)$. We consider $\mathcal{Z}_\gamma(r_\gamma) = \ln(1 + (r_\gamma))$. For illustrating the method of how the max-flow min-cut is achieved, we consider an example topology with $J = 9$ TSN switches, see Figure 2.2. For our utility maximization evaluations, J is a strictly positive constant that is varied up to $J = 40$ TSN switches. For a given number J of TSN switches, the evaluation network graph is randomly generated with the NetworkX tool in Python. From the illustrative topology in Figure 2.2, we observe the following paths from Source 1 to Destination 1: $1 \to 2 \to 3 \to 4 \to 5$, $1 \to 8 \to 5$, and $1 \to 9 \to 5$. Next, from Source 2 to Destination 2, the path is $3 \to 6 \to 5$. Now, we use the 3 cuts from Source 1 to Destination 1 which makes $r_{\gamma 1} = 3$; analogously, from Source 2 to Destination 2, $r_{\gamma 2} = 1$. Accordingly, the theoretical utility values are $\mathcal{Z} = \ln(4) + \ln(2) = 2.02 \approx 2$. As this example topology illustrates, the achieved utility varies with the number of nodes J. Generally, as the number of max-flow paths increases, there are more options for routing and scheduling the flows. These max-flow paths are cut as shown in the example topology, i.e. the cuts are the numbers of paths available from source to sink. Thus, when the number J of nodes increases, the number of paths increases, and the number of scheduled flows will commensurately increase.

Throughout, we consider the dynamic scenario of Section 2.3.1. with Poissonian generation of scheduled traffic flows requests. Flows expire after one base period and generate frames of size 100 bytes to give a prescribed relative traffic load λ.

We conduct 100 independent replications of the simulation for each evaluation scenario. The resulting 95% confidence intervals are less than 5% of the corresponding sample means for all performance metrics and are omitted from the plots to avoid visual clutter.

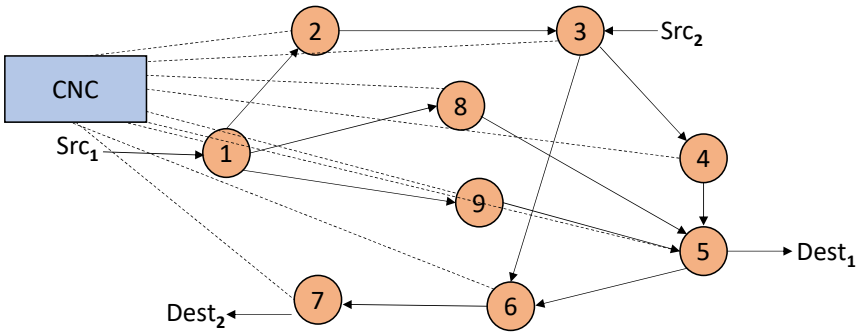


Figure 2.2: Illustrative Network Topology with J = 9 Tsn Switches.

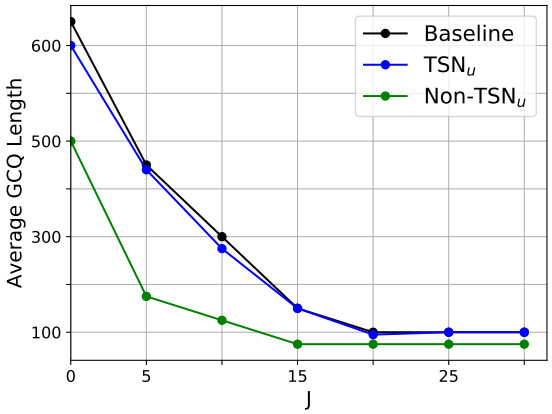


Figure 2.3: Tsn Queue Length (in Bytes) as a Function of Number J of Nodes; Traffic Load $\lambda = .6$, Fixed

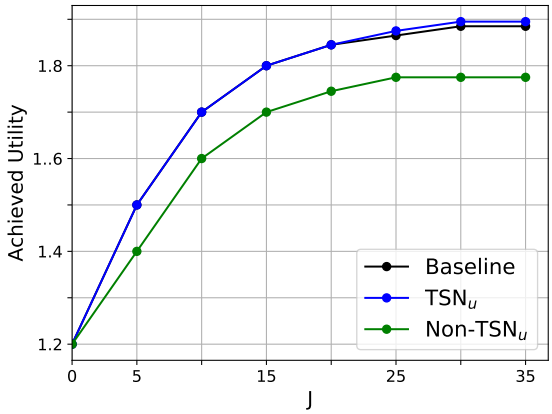
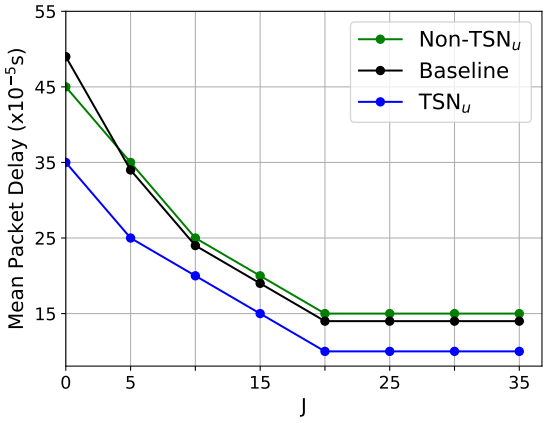


Figure 2.4: Achieved Utility as a Function of Number J of Nodes; Traffic Load $\lambda = .6$, Fixed

2.4.2 Comparison of TSN$_u$ with Non-TSN$_u$ and Virtual Queue Baseline

Our initial comparisons are made towards achieving a better admissibility as that obtained in Nasrallah *et al.* (2019), where admission control and routing are not jointly optimized. We refer to the approach from Nasrallah *et al.* (2019) as Non-TSN$_u$. Further, we conduct a baseline comparison against the virtual queue based approach in Neely (2010) that achieves near-optimal results to provide a justification for our model.



Packet Delay with Respect to Number of Switching Nodes J; For Fixed Traffic Load $\lambda = 0.6$.

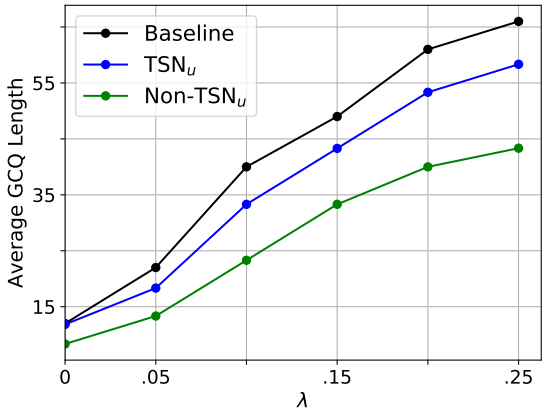


Figure 2.5: Average Gcq Length W.R.T Arrival Rate λ; For Fixed Number of Nodes $J = 9$.

Figure 2.3 shows the Gate Control Queue (GCQ) length, represented by $\mathcal{F}(t)$ in the analysis, as a function of the number of nodes J. We observe from Figure 2.3 that Non-TSN$_\text{u}$ has lower queue occupancies than the two optimized approaches. This is because Non-TSN$_\text{u}$ relies only on the local physical queue length differences for a given TSN output port and employs a heuristic approach without a notion of service time per queue; hence, Non-TSN$_\text{u}$ does not capture network congestion accurately. In contrast, both the baseline approach and the proposed TSN$_\text{u}$ approach accurately capture the physical network congestion by considering a common virtual queue for all the outgoing ports of a TSN switch.

Fig. 2.4 shows the achieved utility as a function of the number J of nodes in the topology. We observe the expected behaviour of increasing network utility with increasing number J of network nodes. Importantly, this evaluation demonstrates the throughput-optimality of the TSN$_\text{u}$ policy since TSN$_\text{u}$ and baseline achieve nearly the same utility.

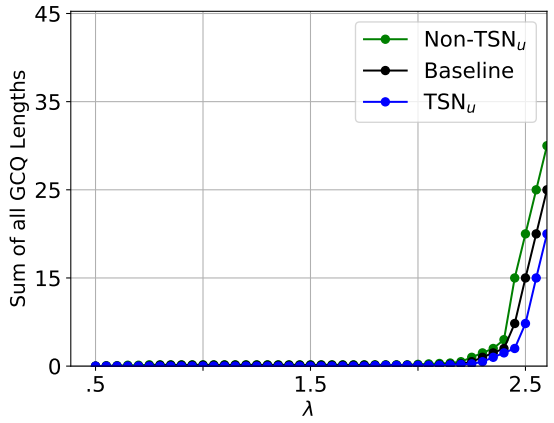


Figure 2.6: Sum of All Gcq Lengths as a Function of Traffic Load λ.

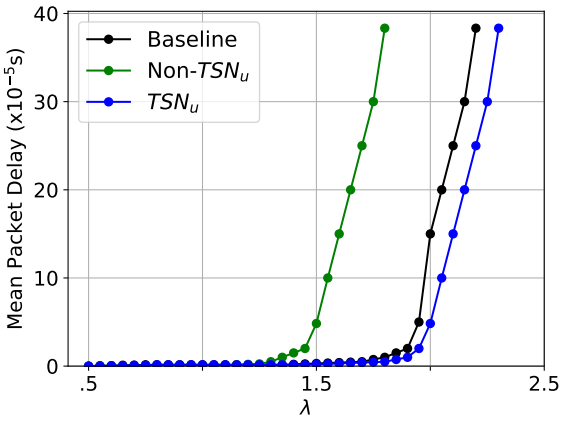


Figure 2.7: Mean Packet Delay as a Function of Traffic Load.

Figure **??** shows the mean packet delay of the scheduled traffic as a function of the number J of TSN switches for a fixed traffic load λ. We observed that the higher latency of Non-TSN$_u$ can be attributed to the packet cycling in the network, whereas in the proposed TSN$_u$ model, packets only traverse through acyclic paths.

Figure 2.5 shows the comparisons of the average gate control queue lengths as a function of the traffic arrival rate. In this comparison, Non-TSN$_u$ produces smaller queue lengths as compared to the other approaches because a single physical queue per TSN switch output port is considered, whereas the TSN$_u$ and baseline approaches consider a virtual queue per TSN switch. For TSN communication, the use of a virtual queue can enable a highly decongested queue, i.e., a queue that transmits packets to all routes acyclically to maintain low delays. Fig. 2.5 therefore, provides a vital insight as to how a virtual queue can continue to satisfy optimal rates compared to physical queues. Importantly, by routing over optimal acyclic paths, TSN$_u$ achieves shorter packet delays than Non-TSN$_u$, as examined further below in Fig. 2.7.

Figure 2.6 shows the behavior of sum of queue lengths inclusive of BE and ST as

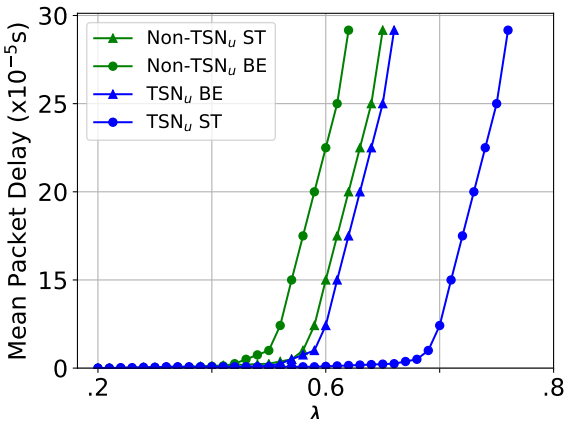


Figure 2.8: Non-TSN$_u$ vs TSN$_u$ under Different Load Conditions

a function of the traffic load λ. In Figure 2.7, we compare the mean packet delay of the three policies as a function of the traffic load λ. Around $\lambda \approx 1.5$, the non-TSN$_u$ policy begins to saturate. The baseline policy saturates next. Further, an arrival rate of $\lambda \approx 2$ is supported by TSN$_u$ which demonstrates the usefulness of joint optimization of routing and scheduling, which has a direct implication on queueing delay. As Non-TSN$_u$ explores all possible paths to route packets to the destination, we can observe a higher delay, which is also the reason why a higher total queue length (as deduced from Little's Law, a higher sum queue length results in higher delay). The main reason why Non-TSN$_u$ behaves in this manner is because, the route estimation of Non-TSN$_u$ explores all available paths; instead, TSN$_u$ only takes the non-cyclic routes which are optimal, reducing the latency substantially.

Fig. 2.8 shows how the BE and ST traffic separation enables dynamic control of the GCQ. Essentially for an arrival rate λ above 0.6, we observe that the Non-TSN$_u$ traffic saturates. Non-TSN$_u$ is not able to support a higher arrival rate, mainly because it lacks optimal control. In contrast, the TSN$_u$ policy implements optimal

control which substantially improves the performance.

2.4.3 Comparison of TSN$_u$ with TSSDN

Set-Up

As there is no openly available implementation of TSSDN Nayak *et al.* (2017), we implemented the TSSDN optimization rules in a customized control algorithm called TSSDN*. Primarily, the TSSDN* algorithm considers all the ILP rules of TSSDN, but uses a pruning methodology inspired by the general approach of Nayak *et al.* (2017). As the approach in Nayak *et al.* (2017) mainly uses the results of the previous time-slots and computes the schedule based on the previous observations, we can reduce the algorithm to a knapsack dynamic programming problem.

In terms of the computational effort, we note that due to the NP hardness of the static routing and the associated TSSDN approximations Nayak *et al.* (2017), TSSDN* requires on the order of 100 seconds of computation time on a contemporary PC for typical scenarios with on the order of 10^4 flows. In contrast, in our evaluations, the proposed TSN$_u$ requires on the order of 50 μs per flow for 70 time-slots per base period. This is mainly because the routing in TSSDN is based on checking the validity of the time-slots along the entire path every time a flow is registered; whereas TSN$_u$ routing is based on simple max-flow paths, which are not cyclic.

For the comparison, we consider a base period (TSN cycle time) of 1 ms, as defined in the ILP based solver in Nayak *et al.* (2017) and the SMT based solver used in Craciunas and Oliver (2016). In Nayak *et al.* (2017), up to 50 time-slots are granted within a base period; TSN$_u$ follows a similar timing structure with up to 70 time-slots in a base period.

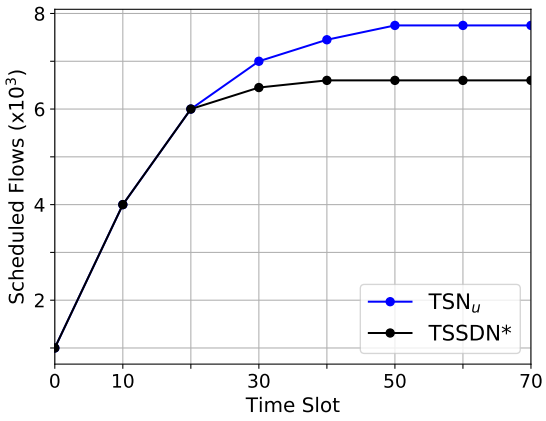


Figure 2.9: Number of admitted scheduled flows for proposed TSN_u and TSSDN* Nayak *et al.* (2017)as a Function of Number of Time-slots per Scheduling Cycle (Base Period); Fixed Parameters: $J = 9$ TSN switches, Traffic Load $\lambda = 0.6$.

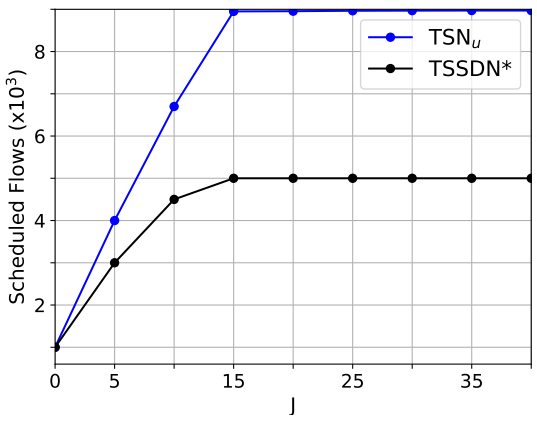


Figure 2.10: Number of Admitted Scheduled Flows as a Function of the Number J Of Nodes (Tsn Switches); Fixed Parameters: Traffic Load $\lambda = 0.9$, 70 Time-slots per Base Period.

Comparison Results

Figure 2.9 shows the numbers of admitted scheduled flows as a function of the number of time-slots per base period for a moderately sized network of $J = 9$ TSN switches. We observe that the numbers of admitted scheduled flows saturate with increasing number of time-slots per base period for both TSSDN* and TSN_u. The key difference is that TSN_u achieves about 15% more admitted scheduled flows for a high number of time-slots per base period than TSSDN*. Furthermore, Figure 2.10 indicates that the performance gap in terms of the number of admitted scheduled flows between TSN_u and $TSSDN^*$ increases as the number J of TSN switches in the network increases to about $J = 15$ and then saturates. The overall advantage of TSN_u results from

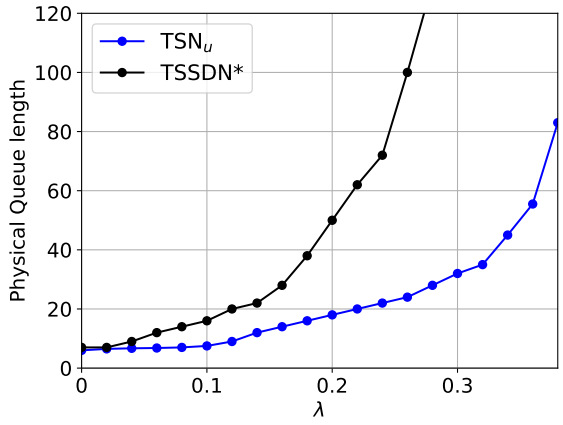


Figure 2.11: Physical Queue Length as a Function of Traffic Load λ; Fixed Parameters:: $J = 40$ nodes, 70 time-slots per base period.

jointly optimizing the admission control, routing, and scheduling. More specifically, the TSN_u route-selection consists of assigning a dominating set of J to every incoming flow. This allows multiple routing paths, based on the order of feasibility. On the other hand, for such dynamic scenarios, $TSSN^*$ resorts to a time-slot first approach, wherein time-slot sizes matter much more than the links. This means that depending on the traffic timings, some flows will have more options to be accommodated, while other flows may be rejected since the routing options are not fully optimally explored. In contrast, TSN_u optimally explores both the available routes and time-slot options for accommodating flows.

Fig 2.11 compares the physical queue lengths of TSN_u and $TSSDN^*$ for a large network of $J = 40$ TSN switches for increasing traffic load λ. We observe from Fig 2.11 that TSN_u achieves substantially shorter queue lengths than TSSDN*, particularly for heavy traffic loads. The shorter TSN_u queue lengths are mainly due to TSN_u exploring all the feasible routing paths; whereas, $TSSDN^*$ is time-slot dominant i.e., if there are available time-slots in one path, $TSSDN^*$ will continue to schedule on

that path until the path is completely saturated, while alternate paths may be left under utilized. The shorter TSN_u queue lengths translate into shorter queueing delays compared to TSSDN.

Overall, the TSN_u performance advantages are due to the rate-stability offered by the TSN_u framework and its focus on optimal joint control of admission control, routing, and scheduling. Generally, TSSDN* strives to minimize the number of links over which the flows are scheduled. However, TSN_u minimizes the scheduling re-attempts of TSSDN* by exploring multiple a cyclic paths that are stored in the Qcc data-base which leads to short queue lengths. Thus, TSN_u achieves a higher network utility owing to the smaller network-wide queue lengths. In summary, the key design difference why TSN_u outperforms the state-of-the-art TSSDN* approach Nayak *et al.* (2017) is the joint optimization of the admission control, route selection, and scheduling in TSN_u.

2.5 Conclusion and Future Work

We have proposed the TSN_u framework for jointly optimizing admission control, routing, and scheduling in time sensitive networking (TSN). The TSN_u framework can provide optimal control in Industrial Internet of Things (IoT) settings, such as factory floors. We have shown how TSN gating constraints impact the TSN_u policy and have conducted a detailed analysis of the TSN_u policy. Our simulation evaluations have indicated substantially increased flow admissibility and reduced queue lengths compared with a state-of-the-art policy.

There are numerous directions for future research that can build on the TSN_u time sensitive industrial IoT framework developed in this article. One direction is to examine strictly deterministic schedules based on deterministic network calculus Le Boudec (2018); Zhao *et al.* (2018) in order to satisfy flows in edge cloud based scenarios that

are pertinent in the Industry 4.0 realm, such as real time factory automation and robotics. The present study did not employ any artificial intelligence or machine learning techniques and rather relied on conventional optimization techniques. An interesting direction for future research is to examine whether a dual-subgradient approach can reduce the time complexity of the proposed TSN optimization framework Another direction is to explore TSN in the wider context of network function virtualization (NFV). Through NFV, TSN services could be provided in a flexible scalable manner via network slicing.

Chapter 3

NETWORK FAULTS AND RECOVERY

3.1 Introduction

Emerging technologies, such as the industrial Internet of Things (IoT), autonomous vehicular networks, and smart healthcare systems, are critical time-sensitive applications Al Ridhawi *et al.* (2018); Gavriluţ and Pop (2020). Machines in industrial automation have started to implement time-sensitive control loops that require a strict delay bound. However, with the advent of Industry 4.0 and the Industrial IoT the new challenge of managing the time-sensitive traffic has arisen Baccarelli *et al.* (2017); Raptis *et al.* (2019). In order to address the real-time communication needs of these applications Belliardi, R., et al. (2018), the IEEE Time Sensitive Networking (TSN) Task Group has developed the IEEE 802.1Qbv standard Bello and Steiner (2019). The IEEE 802.1Qbv standard includes the Time Aware Shaper (TAS) which relies on synchronized time cycles and gate time periods for high-priority scheduled traffic Nasrallah *et al.* (2019). TAS strives to provide on-time packet services.

To cope with dynamic network scenarios, various reconfiguration mechanisms need to be considered that optimally allocate network resources. Some recent studies, such as Feng *et al.* (2022); Gutiérrez *et al.* (2017); Kehrer *et al.* (2014), have considered using path protection mechanisms via redundancy of the data transmissions, i.e., sending multiple data copies into the network, thus pursuing a data-oriented perspective. In contrast, in this study we pursue a network-oriented perspective focused on the links since link failures will disrupt all transmissions over a failed link, i.e., sending multiple data copies over the same path with a failed link will not overcome

a link failure. Also, a link focus avoids the resource waste from transmitting multiple data copies and can utilize links more efficiently.

There are two categories of protection while considering the links, namely:

1. Path protection where two paths primary and backup path are assigned to each connection. When there is a primary path failure traffic is routed through back-up path;

2. Link protection where an alternate link is computed which takes the re-routed traffic during a failure.

We focus on path protection. We find ways to protect a network with probabilistic path failures

In-order for a network to operate in its completeness, it is important to provide sustainable communication paths. Hence, we target in this paper the protection of paths in scenarios where links fail with certain probabilities that may not be equal to one so as to design a sustainable Industry 4.0 communication network solution.

There are several factors that come into play which can cause multiple link failures Many studies such as Tapolcai *et al.* (2017) Shen *et al.* (2005), have proven that the disjoint path problem in the context of SRLG is NP-complete. The key difference in our study is that we consider network failures on a factory floor that may not be deterministic. Therefore, we model the SRLG failure based on a probability index that may not necessarily be one. To further explain this scenario, consider an example of a fire emergency on a factory floor; then the wires closer to the fire are destroyed with a higher probability than the wires that are far away from the fire.

While establishing key management policies in the network, Software Defined Networking (SDN) control has played a key role in recent years. Numerous studies, such as Barakabitze *et al.* (2020); Castillo *et al.* (2020); Gerhard *et al.* (2019); Kellerer

et al. (2019); Leng *et al.* (2019); Pinheiro *et al.* (2017); Yang and Yeung (2020); Guck *et al.* (2016), have shown varying benefits of remote monitoring of network events. Primarily, the orthogonality of the control-plane and the data-plane enables a programmatic independence that allows efficient resource management. Generally, path computation issues can be very well managed via the SDN control strategies Wang *et al.* (2018) and we accordingly employ SDN.

We use an SDN controller to implement the 802.1Qcc variables (including the flow instantiation parameters and the 802.1Qbv gate control parameters) Bello and Steiner (2019). The SDN controller cohesively communicates with the data-plane TSN switches, which are time synchronized, e.g., via the IEEE 1588 Precision Time Protocol. For a topology change, we resort to the SDN controllers response to change. Understandably, the requirement to enforce time-bounded packet latencies may be abandoned in a disaster situation. Therefore, our aim does not incorporate achieving bounded packet latencies during path recovery.

The main contributions of this article are:

i) We design a novel solution called TSN_{u1} to provide path protection in case of correlated network link failures in Industry 4.0 scenarios. This TSN_{u1} framework relies on an SDN controller to react to failures.

ii) We formulate the correlated link failure problem as a non-linear programming objective of finding primary and backup paths. In doing so, our path-protection approach protects against multiple failures.

iii) We design a set of approximation algorithms to find close-to-optimal solutions in multi-failure scenarios, such as a factory floor emergency where multiple links fail and demonstrate via simulations the benefits of our TSN_{u1} model in Industry 4.0 scenarios.

The remainder of this paper is structured as follows. The related work is reviewed in Section 5.2. Section 3.3 introduces the design and formulation of our framework. Section 3.4 presents the theoretical analysis, while Section 3.6 presents the simulation evaluation and comparison with state-of-the-art approaches. Section 3.7 provides concluding remarks and outlines future research directions in the real-time communication domain, such as Industry 4.0.

3.2 Related Work

Efforts in the past have mainly divided reliability and sustainability based studies in Time Sensitive Networks into two broad domains. The first domain is time redundancy, the studies Raagaard *et al.* (2017) Balasubramanian *et al.* (2020) approach the problem based on the reconfiguration of gate control list schedules. In Raagaard *et al.* (2017), a TSN agent is designed that is aware of the traffic conditions at every node in the network. While Balasubramanian *et al.* (2020) relies on an optimized approach via route maintenance, whereby the objective is similar to that of Raagaard *et al.* (2017) where new traffic flows can be accommodated by maintaining the queues in the ports and checking the feasibility of the schedule. Further, in Steiner (2010) designs an SMT solver based approach for time-triggered traffic that consists of internal fault recovery methods for adding flexibility at the time of re-executions during failure. Similar approaches have been considered in Craciunas and Oliver (2016) Nayak *et al.* (2017) where a time based redundancy enables a faster run-time reconfiguration of the network.

The second approach is space redundancy, which selects a simplified error model where singleton errors can occur with a predefined setting of least known minimum arrival times, see the 802.1CB TSN standards. In the present study, we modify this approach via the concept of a Shared Risk Link Group. We note that according to

the 802.1CB standards, the simplified error calculation may not always produce an optimal result. For instance, during a failure event, the packets which are in transit cannot be recovered if the link fails. Further, if we consider a typical factory scenario, there could always be the case where one link failure may be correlated with another link failure. Further, there need not be a specific inter-arrival time before another fault occurs. Accordingly, we consider a probabilistic scenario where a failure can occur based on a particular event probability that can encompass realistic events, such as man-made or natural disasters in the factory. Therefore, this study is one of the first to prove that probabilistic failures in TSN networks can be recovered in a reliable manner to maintain the sustainability of a TSN network.

We employ the Software Defined Networking (SDN) paradigm McKeown *et al.* (2008)Said *et al.* (2019) and the OpenFlow interface, that is utilized in the IEEE 802.1cc TSN standards. According to IEEE 802.1cc, a time synchronization function ensures that all other functions are executed within predetermined deadlines. However, it has been established in all previous studies that in order to maintain the reliability of the time sensitive systems during faults, recovery takes precedence over timeliness i.e., the evaluation metric changes from reducing latency to quality of protection. That is, how well is the network prepared to recover from a particular fault? In the next section we elaborate our system model and eventually show how we achieve a better protection quality compared to state-of-the-art models, such as RFT-TSN Feng *et al.* (2022), baseline Lagrangian models Álvarez *et al.* (2019), or the satisfiability modulo theories (SMT) model Craciunas and Oliver (2016).

Table 3.1: Summary of Key Notations.

Notat.	Meaning
$G(V, E)$	Graph with set V of nodes and set E of links (l, m)
$\|E\|$	Cardinality of set of links E
$p_{l,m}$	Failure probab. of link (l, m), abbrev. p for set of links E
$\kappa_{l,m}$	Link weight for link $(l, m) \in E$
$H_1(p, z)$	Failure probability of path z for link failure probs. p
$\mathcal{F}(t)$	Gating function
$L(\cdot)$	Lagrangian function for approximation
$\rho\chi$	Lagrangian multiplier vectors
z, w	Primary and back-up paths
β	Binary variable to replace the product zw

3.3 Model Overview

In this section we elaborate the problem and the model that we will use throughout this paper. We consider a graph $G = (V, E)$, where V is a set of nodes and E is the set of links. A link (l, m) in set E originates from node l, $l \in V$ and ends at node m, $m \in V$.

We follow a single source *src* to sink *dst* node pair as, for instance in Balasubramanian *et al.* (2020), whereby (*src*, *dst*) $\in V$. We find two paths, namely the primary path z and the back-up path w from (*src*, *dst*) with minimum JFP. Let the indicator variable $z_{l,m} = 1$, if the primary path is via (includes) the link (l, m); otherwise, $z_{l,m} = 0$. Analogously, we define a back-up path with the indicator variable $w_{l,m}$. For brevity, we will write all these variables by dropping the indices, e.g., instead of $z_{l,m}$ we will write z.

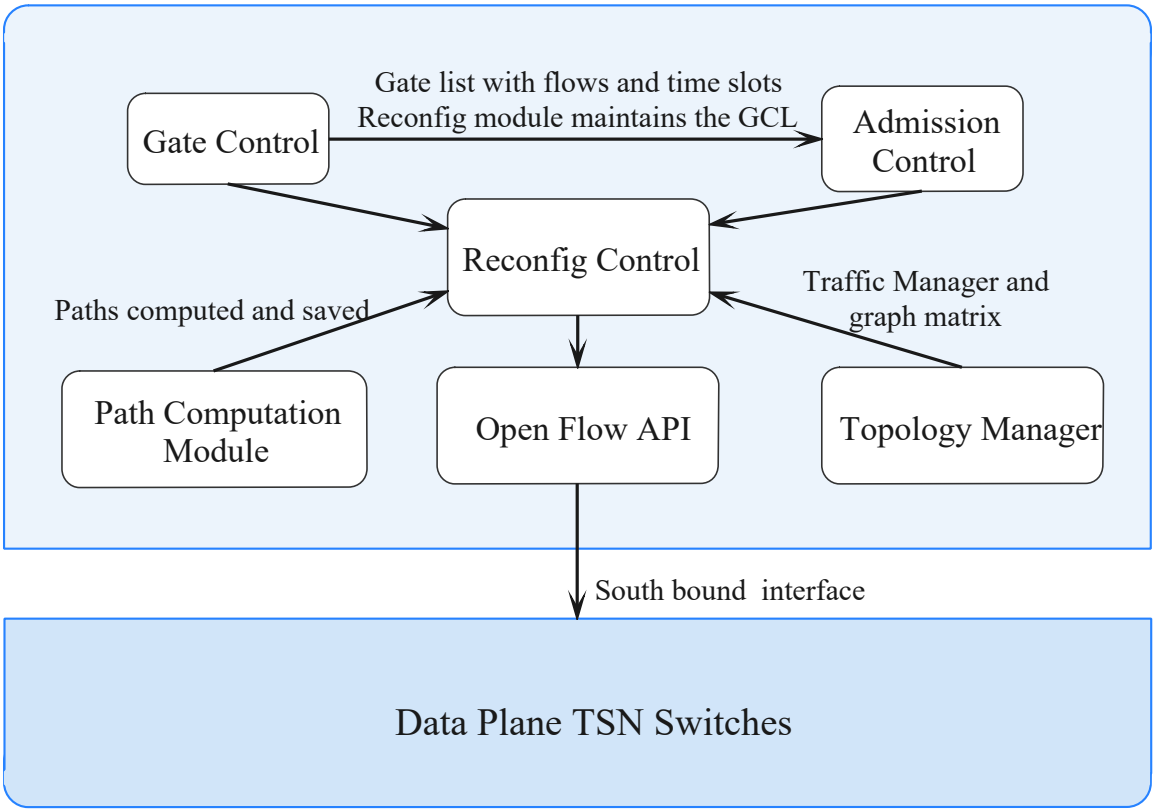


Figure 3.1: Illustration of Architecture and Operation of Proposed TSN_{u1}.

3.4 Recovery for Independent (Non-Shared) Link Failures

We first develop a model for independent link failures and how the SDN controller can create a fall-back mechanism for such independent link failures. More specifically, for independent link failures, the links of the primary and back-up paths are not shared, i.e., the two paths are link-disjoint.

3.4.1 Link Failure Model and Recovery Goal

Let $p_{l,m}$ be the probability of a link failure for link (l, m), then the probability that the link survives is $1 - p_{l,m}$.

The recovery goal is to minimize the JFP. In particular, for a network with given

link failure probabilities $p_{l,m}$, our TSN$_{u1}$ algorithm finds the primary (z) and back-up (w) paths that minimize the JFP. During normal operation, the *src* to *dst* packet traffic flows over the primary path z and meets the TSN timliness requirements. If a link failure occurs that disrupts the primary path z, then the *src* to *dst* packet traffic flows is re-routed via the pre-computed back-up path w. The re-routed traffic may no longer meet the TSN timeliness requirements as priority is given to maintaining a basic network connectivity via the back-up path and not to reserve resources that would ensure that the rerouted traffic still meets its timeliness requirement. A protection scheme with TSN level resource reservations is left for future research. If both primary and back-up paths fail due to the link failures then the *src* to *dst* packet traffic flow is disrupted, i.e., the event of a joint failure has occurred. Our optimization goal is to minimize this joint failure probability (JFP).

3.4.2 Path Failure Probability Model and Minimization

Let $z_{m,l} = 1$ be an indicator variable that is set to one when the primary path traverses the link (l, m); and is set to zero otherwise. The product form $\prod_{l,m}(1 - p_{l,m}z_{l,m})$ gives the sustainability of the path that is composed of the considered links (l, m). Maximizing the path sustainability corresponds to:

$$\min_{z} 1 - \prod_{l,m}(1 - p_{l,m}z_{l,m}) \tag{3.1}$$

$$s.t \sum_{l:(l,m)\in E} z_{l,m} - \sum_{l:(m,l)\in E} z_{m,l} = \begin{cases} 1, & l = src \\ -1, & l = dst \\ 0, & \text{otherw. } \forall l \in V. \end{cases} \tag{3.2}$$

Following Shen *et al.* (2005), the constraint (3.2) requires that the set of links selected for the path z forms a path from *src* to *dst*. Since the problem formulation in an integer non-linear program, we need to reduce it further to a linear program with an

assumption as shown below. We convert the above non linear program to a solvable linear problem, by assuming that the link failure probabilities $p_{l,m}$ are not equal to one, i.e., we assume that the $p_{l,m}$ values may approach one without actually taking on the value one:

$$p_{l,m} \in [0, 1), \forall (l, m). \tag{3.3}$$

Optimal Path Choice Theorem: Based on the assumption in Eqn. (3.3), the objective function in Eqn. 3.1 can be reduced to an equivalent ILP with the objective function:

$$\min_{z} - \sum_{l,m} z_{l,m} \log(1 - p_{l,m}). \tag{3.4}$$

subject to the link constraints in Eqn. (3.2).

Proof: The objective function in Eqn. 3.1 can be written as $\max \prod_{(l,m)}(1 - p_{l,m} z_{l,m})$. Taking the logarithm of the objective function gives

$$\max \sum_{l,m} \log \left(1 - p_{l,m}, z_{l,m}\right). \tag{3.5}$$

Applying the log identity for the binary variable $z_{l,m}$ gives

$$\log(1 - p_{l,m} z_{l,m}) = z_{l,m} \log(1 - p_{l,m}). \tag{3.6}$$

Note that $\max f(x) = \min[-f(x)]$, completing the proof.

This Optimal Path Choice Theorem shows that the path with the least failure probability is the least-cost path with link costs (weights) set to $-\log(1 - p_{l,m})$. If the link failure probabilities $p_{l,m}$ are small, then

$$-\log(1 - p_{l,m}) \approx p_{l,m}. \tag{3.7}$$

Note that the greedy Algorithm 2 needs to go through only two iterations Find z, find next best path, find w.

Algorithm 2: SDN Assisted Path Finder (TSN_{u1})

Input: Initialize link weights $\kappa_{l,m} = -\log(1 - p_{l,m})$ (or $\kappa_{l,m} = p_{l,m}$ for small $p_{l,m}$)

Output: Shortest prim. path z and back-up path w;

```
Find shortest primary path z;
while finding the next best path disjoint from z do
    if directed edges found then
        Check if used by z;
        If True Remove such edges;
Find second shortest back-up path w;
```

3.4.3 Path Computation with Disjointedness Constraint

Let $H_1(p, z)$ [and $H_1(p, w)$] denote the failure probability of a path z [and w], respectively, for a given set p of link failure probabilities $p_{l,m}$. For link-disjoint paths z and w, the failures are mutually independent as there is no common link shared by z and w. For the independent path failures, the joint failure probability (JFP) can be represented as $H_1(p, z) \cdot H_1(p, w)$. Thus, we can formulate the minimization of the JFP as:

$$\min_{z,w} H_1(p, z) \cdot H_1(p, w) \tag{3.8}$$

$$s.t. \;\; z_{l,m}, z_{m,l} \text{ Constraint as per Eqn. (3.2)} \tag{3.9}$$

$$s.t. \;\; w_{l,m}, w_{m,l} \text{ Constraint as per Eqn. (3.2)} \tag{3.10}$$

$$s.t. \; z_{l,m} + w_{l,m} \leq 1, \; \forall (l, m) \in E. \tag{3.11}$$

The first two constraints follow Shen *et al.* (2005) and enforce that the sets of selected links for the paths z and w forms path from *src* to *dst*, analogous to Eqn. (3.2). The

constraint Eqn. (3.11) enforces that there is no shared link between z and w. If a pair of paths z and w satisfies the constraints (3.9)–(3.11), then the paths z and w form a disjoint path pair. We will transform this non-linear program into a linear program via the following assumption.

Assumption: The length of the shortest path from *src* to *dst* be q and the link failure probabilities be uniform, i.e.,

$$p_{l,m} = p \quad \forall (l,m). \tag{3.12}$$

Length Lemma: The link probability p needs to satisfy link failure probability p satisfies $1 - \frac{1}{2^{1/q}} \leq p \leq 1$, where q is the length of the shortest path from source to destination so as to enable a concave minimization.

With the assumption of uniform link failure probabilities, the problem in Eqn. (3.8) becomes a concave minimization. We proceed to design a heuristic algorithm to solve the concave minimization problem.

Proof: Defining the number $Z = \sum_{l,m \in E} z_{l,m}$ of hops of the primary path z and the number $W = \sum_{l,m \in E} w_{l,m}$ of hops of the secondary path w, we note that the entire path z survives with probability $(1-p)^Z$, while path z fails with probability $1-(1-p)^Z$. Multiplying the path z failure probability $1-(1-p)^Z$ with the corresponding failure probability $1-(1-p)^W$ for path w gives the objective function from Eqn. (3.8) as:

$$h(Z,W) = 1 - (1-p)^Z - (1-p)^W + (1-p)^{(Z+W)}. \tag{3.13}$$

The proof of the concavity of the objective function in Eqn. (3.13) can be shown as per Peressini *et al.* (1988) as follows. Consider S to be the Hessian of $h(Z,W)$. Then, concavity exists when $S_{11} \leq 0,\ S_{22} \leq 0$, and $S_{11}S_{22} - S_{12}S_{21} \geq 0$. We have the Hessian given by the factor $e^{(Z+W)\log(1-p)}\ \log^2(1-p)$ multiplied by

$$\begin{bmatrix} 1 - e^{-W\log(1-p)} & 1 \\ 1 & 1 - e^{-Z\log(1-p)} \end{bmatrix}. \tag{3.14}$$

By definition, $p < 1$, $Z > 0$, and $W > 0$, thus $1 - e^{-Z\log(1-p)} < 0$ and $1 - e^{-W\log(1-p)} < 0$, satisfying the concavity condition on the diagonals. The determinant condition can be expressed as $e^{Z\log(1-p)}e^{W\log(1-p)} \leq 1$. We can evaluate this further, to show that:

$$e^{Z\log(1-p)}e^{W\log(1-p)} \leq 2e^{q\log(1-p)} \leq 1. \tag{3.15}$$

The first inequality in Eqn. (3.15) is due to $Z \geq q, W \geq q$; the assumption $1 - \frac{1}{2^{1/q}} \leq p$ gives the second inequality. From Peressini *et al.* (1988), we can ascertain that $h(Z, W)$ is concave on Z and W and the original objective function in Eqn. (3.8) can be re-written as

$$1 - (1-p)^{\sum_{l,m} z_{l,m}} - (1-p)^{\sum_{l,m} w_{l,m}} + (1-p)^{\sum_{l,m}(z_{l,m}+w_{l,m})}, \tag{3.16}$$

which shows the concavity and convexity is preserved under the linear mapping composition as per Boyd *et al.* (2004). Hence, the problem is a concave minimization.

3.4.4 Path Selection Algorithm

In order to develop the path selection algorithm we rely on the majorization theory. According to this theory, a vector x is said to be majorized by another vector y if:

$$\sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} y_i, \;\; k = 1, 2, \ldots, n-1, \tag{3.17}$$

and $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$ which can be represented by $y \succ x$. The majorization theory shows how evenly the vector is distributed, e.g., $x = [2, 2, 2]$ is evenly distributed over the 3-tuple vector with a sum $(2 + 2 + 2)$ and is majorized by the vector $[1, 1, 4]$ since majorization does not depend on the order of the elements of the vectors, i.e., the statement $[1, 1, 4] \succ [2, 2, 2]$ is equivalent to $[4, 1, 1,] \succ [2, 2, 2]$. With this premise we define another condition for minimization called the *Schur Concavity*.

A function $h : K \subseteq R^n \rightarrow R$ is Schur concave if $h(z) \geq h(w)$ for any $z, w \in K$ such that $w \succ z$. It is Schur convex if $h(z) \leq h(w)$ for $w \succ z$. Our goal now reduces

to show that our objective function is Schur concave and we develop a heuristic algorithm to find a path. We continue to assume a uniform link failure probability i.e., Eqn. (3.12) continues to hold. Defining $Z = \sum_{l,m} z_{l,m}$ and $W = \sum_{l,m} w_{l,m}$ as the number of hops taken in the primary and backup paths, respectively, the objective function Eqn. (3.13) can be re-written as:

$$h(Z, W) = \left[1 - (1-p)^Z\right] \cdot \left[1 - (1-p)^W\right]. \tag{3.18}$$

Although, a failure might not always be within the seven hop delay constraints (as at the time of crisis, the goal is to re-instate proper connection without worrying about the delay requirement, it should be noted that the total number of packets serviced should still continue to follow the Gating function dynamics examined in Balasubramanian *et al.* (2020). Primarily, the topology manager gives the total number of hops which need to be traversed for a particular reconfiguration to take place; while the gate control engine provides the time-slot within which the reconfiguration should take place. However, the gating function hands over the control to path computation modules when link failure happens. We recall (Balasubramanian *et al.*, 2020, Lemma 1):

Lemma 1: *A non-negative function called the gating function $\mathcal{F}(t)$ bounds the total number of packets serviced, which exists for every chosen TSN path Ψ. The definition of $\mathcal{F}(t)$ is given by $\sup_{0 \le t' \le t} \mathcal{F}(t')$, whereby, $\mathcal{F}(t)$ is monotonically increasing in t and acts as a bound.*

With this premise, the problem can re-written as: minimizing the function $h(Z, W)$, subjected to the gating constraints $\mathcal{F}(t)$, with additional constraints of $Z = \sum_{l,m} z_{l,m}$ and $W = \sum_{l,m} w_{l,m}$. Such an objective function will be Schur concave.

Theorem: If we are given that the function $h(Z, W)$ in Equation (3.18) is continuously differentiable and symmetric, then the function is said to be Schur concave

when $0 \leq p < 1$.

Proof: The proof of this theorem is based on the Schur criterion. If a function $h : (a_1, a_2)^n \to R$ is symmetric and continuously differentiable, it is said to be Schur concave on $(a_1, a_2)^n$ iff

$$(z_j - z_k)\left(\frac{\partial h(z)}{\partial z_j} - \frac{\partial h(z)}{\partial z_k}\right) \leq 0 \ \forall 1 \leq j \leq k \leq n, z \in (a_1, a_2)^n. \tag{3.19}$$

Evidently the function is symmetric and differentiable as the function values are unchanged when the Z and W terms are exchanged. Applying this to the function $g(Z, W)$, we have

$$\begin{aligned}(Z - W)\left(\frac{\partial h(Z,W)}{\partial Z} - \frac{\partial h(Z,W)}{\partial W}\right) \\ = -\log(1-p)(Z-W)\cdot\left[(1-p)^Z - (1-p)^W\right] \\ \leq 0. \end{aligned} \tag{3.20}$$

This shows that $h(Z, W)$ is Schur concave. From the Schur concavity definition we know that if the function $h(Z, W)$ is Schur concave, then it can be minimized at the unevenly distributed points. A pair with unbalanced paths is preferred because its joint failure probability may be lower than that of a balanced pair. Similarly, a Schur convex function is minimized at evenly distributed points. For instance, if we have a $Z = 3$, $W = 3$ pair and a $Z = 2$, $W = 4$ pair, then by the concavity property, we have $h(3, 3) \geq h(2, 4)$

as $(2, 4) \succ (3, 3)$. We acknowledge that our optimization approach requires that the sums of Z and W, i.e., the values for $Z + W$ are the same for the different considered path options. In practice, different path options could have different sums $Z + W$ of hops, extending the optimization approach to such scenarios with different sums $Z + W$ is a direction for future research.

When applied to the joint probability Eqn. (3.13), we find that the symmetric/balanced pair is more unreliable compared to the unbalanced values. Further, we base our heuristic algorithm design based on the strategy that a good-bad path pair (e.g., a pair with one path having 0 failure probability and another one having 0.2 failure probability will still have a 0 joint failure probability) is better than a "medium-medium" path pair (i.e., a pair with both paths having 0.1 failure probability will give a joint failure probability of 0.01). This is the key strategy behind our heuristic algorithm development that there would be at least one best path or path having the least (minimum) failure probability. Here, we also make a mild assumption that the path chosen does not contain cycles, and if there are cycles then they would be of zero length (i.e., the cycles can be removed without affecting failure probabilities). Thus, it suggests that the chosen paths z and w do not contain cycles.

Connectedness Lemma: If we have an *n-connected* graph, where $n \geq 2$, removing a source-to-destination path in the graph does not disconnect the source and destination nodes Boyd *et al.* (2004); Roughgarden (2010); Wolsey and Nemhauser (1999) Further, our algorithm seeks to find source-destination paths such that after the first path is found, then there is at least one other *src* to *dst* path.

3.4.5 Alternative Solution

We design another heuristic algorithm that leverages an ILP approximation of the problem. A key point here is that we will observe many higher order terms which would involve products of more than two failure probabilities. In such scenarios, i.e., where the failure probabilities are very low, $p_{l,m} \ll 1$, such terms can be neglected, therefore, the ILP can be approximated by going through the analogous steps that led from Eqn. (3.8) to Eqn. (3.13), but now without assuming uniform link failure

probabilities, i.e., by considering heterogeneous link failure probabilities $p_{l,m}$:

$$H_1(p,z)\cdot H_1(p,w) = 1 - \prod_{l,m}(1-p_{l,m}z_{l,m}) - \prod_{l,m}(1-p_{l,m}w_{l,m}) + \prod_{l,m}(1-p_{l,m}z_{l,m})\prod_{l,m}(1-p_{l,m}w_{l,m}). \quad (3.21)$$

We define the binary variable $\beta_{l,m}^{a,b}$ to be set to one when $z_{l,m} = 1$ and $w_{a,b} = 1$; otherwise, $\beta_{l,m}^{a,b}$ is set to zero. This means that $\beta_{l,m}^{a,b} = 1$ indicates that link (l,m) is part of the primary path and link (a,b) is part of the back-up path. Expanding Eqn. (3.21) (while neglecting the higher order terms) gives:

$$H_1(p,z)\cdot H_1(p,w) = \sum_{l,m}\sum_{a,b} p_{l,m}p_{a,b}z_{l,m}w_{a,b}. \quad (3.22)$$

We use the binary variable β to replace the zw product. Thus, minimizing the JFP $H_1(p,z)\cdot H_1(p,w)$ corresponds to

$$\min \sum_{l,m}\sum_{a,b} p_{l,m}p_{a,b}\beta_{l,m}^{a,b} \quad (3.23)$$

$$s.t.\ \beta_{l,m}^{a,b} \geq z_{l,m} + w_{a,b} - 1\ \forall (l,m),(a,b) \in E. \quad (3.24)$$

Along with the constraint (3.24), the constraints (3.2)–(3.11) are inherited here

Now, we make use of Lagrangian relaxation on the constraints (3.2)–(3.11) to simplify the problem further. NOting that Eqn. (3.2) is unimodular, thus linear program relaxation has an integral optimal solution following the rules in Wolsey and Nemhauser (1999), whereby we denote ρ and χ for the respective associated Lagrangian multiplier vectors:

$$L(z,w,\beta,\rho,\chi) = \sum_{l,m}\left(\rho_{l,m} + \sum_{a,b}\chi_{l,m}^{a,b}\right)z_{l,m} + \sum_{l,m}\left(\rho_{l,m} + \sum_{a,b}\chi_{a,b}^{l,m}\right)w_{l,m} + \sum_{(a,b),\,(l,m)}\left(p_{l,m}p_{a,b} - \chi_{l,m}^{a,b}\right)\beta_{l,m}^{a,b}. \quad (3.25)$$

Algorithm 3: Alternate Solution With Lagrangian Approximation (LA): via updates of Lagrangian multipliers and keeping the best path pair at every iteration.

Input: Initialize: $i = 0$, $\rho_{l,m}(0) = p_{l,m}$, $\chi_{l,m}^{a,b}(0) = p_{l,m}p_{a,b}$ $\forall (l,m)(a,b) \in E$

Output: Best path pair (z, w)

while iteration index $i < I$ **do**

- Initialize for min. JFP for (a,b) superscripts $\kappa_{l,m} = \rho_{l,m}(i) + \sum_{a,b} \chi_{l,m}^{a,b}(i), \forall (l,m) \in E$;
- Current shortest path found at $z(i)$;
- Initialize for (l,m) superscripts $\kappa_{l,m} = \rho_{l,m}(i) + \sum_{a,b} \chi_{a,b}^{l,m}(i),\ \forall (l,m) \in E$;
- Current back-up path found at $w(i)$
- $$\beta_{l,m}^{a,b}(i) = \begin{cases} 1, \text{ if } \chi_{l,m}^{a,b}(i) > p_{l,m}p_{a,b} \\ 0, \text{ otherwise;} \end{cases}$$
- $\rho(i+1) = \rho(i) + \delta_i(z_{l,m}(i) + w_{m,l}(i) - 1)$;
- $\chi_{l,m}^{a,b}(i+1) = \chi_{l,m}^{a,b}(i) + \delta_i(z_{m,l}(i) + w_{a,b}(i) - \beta_{l,m}^{a,b}(i) - 1)$.
- Initialize $(z^{\text{chosen}}, w^{\text{chosen}}) = (z(i), w(i))$, if $p(i) < p_{\text{high}}$, where $p(i)$ is the joint failure probability of $(z(i), w(i))$; $i++$;

The relaxed problem now becomes:

$$\min_{z,w,\beta} L(z, w, \beta, \rho, \chi). \tag{3.26}$$

This problem is subjected to constraints Eqn (3.2). Additionally, we notice the total unimodularity property of the constraint matrix. A matrix is said to be totally unimodular when the determinant of the sub-matrices is 0, 1, or -1. Thus, as the constraint matrix is totally unimodular, the linear program (LP) relaxation has an integral solution that is optimal. This is also solvable in polynomial time. We solve

this problem with the primal-dual approach, see Algorithm 3 . In this solution, z and w are the optimal shortest paths and $\beta_{l,m}^{a,b}$ is optimal when the $\beta_{l,m}^{a,b} = 1$ if $\chi_{l,m}^{a,b} > p_{l,m}p_{a,b}$; and 0 otherwise. For a maximum number I of iterations, we let δ_i, $i = 1, 2, \ldots I$, denote the positive diminishing step sizes.

In Algorithm 3, we can observe that the updates of the Lagrangian parameters are carried out in Lines 5 and 6. Further, the chosen path or the best path is always saved through the iterations. This algorithm considers the disjointedness. Further, we show that the objective function in Eqn. (3.22) provides an upper bound on the JFP. The proof of this intuitive and can be stated as follows for a failure event $\Omega_{l,m}$ of link (l, m):

Proof of upper-bound: For independent link failures, i.e., $(l, m) \neq (a, b)$,

$$Pr(\Omega_{l,m} \cap \Omega_{a,b}) = Pr(\Omega_{l,m})Pr(\Omega_{a,b}) = p_{l,m}p_{a,b}. \tag{3.27}$$

Then,

$$\begin{aligned} H_1(p, z) \cdot H_2(p, w) =& \; Pr\left(\cup_{(l,m)\in z}\Omega_{l,m}\right) && (3.28)\\ & \cdot Pr\left(\cup_{(a,b)\in w}\Omega_{a,b}\right) \\ =& Pr\left(\cup_{\substack{(l,m)\in z,\\ (a,b)\in w}} \left(\Omega_{l,m} \cap \Omega_{a,b}\right)\right) && (3.29)\\ \leq & \sum_{(l,m)\in z,\, (a,b)\in w} Pr\left(\Omega_{l,m} \cap \Omega_{a,b}\right) && (3.30)\\ = & \sum_{(l,m)\in z,(a,b)\in w} p_{l,m}p_{a,b}. && (3.31) \end{aligned}$$

Note that Eqn. (3.29) for the outer union can be interpreted as finding the common independent failure links for all the links in paths z and w and taking their sum. The inequality (3.30) follows from the conventional union bound (Boole's inequality). Further, the last equality, i.e., Eqn. (3.31), follows from Eqn. (3.27). Overall, this proof shows that via linear approximation we can find an upper bound on the JFP of independent (disjoint) paths.

3.5 Recovery from Shared Link Failures

3.5.1 Link Failure Model and Recovery Goal

In this section we evaluate the probabilistic scenario where there is a possibility of simultaneous link failure. In other words, this is a situation when there is no disjointedness constraint, i.e., a link (l, m) may be shared by both the primary path z and the backup path w. Then, the link (l, m) failure is said to be a simultaneous link failure of z and w.

3.5.2 Path Failure Model

Let $H_s(p, z, w)$ represent the probability that both paths z and w fail due to the failure of a shared link and let $H_{ts}(p, z, w)$ represent the probability that both paths z and w fail due to failures of non-shared links. The probability that both paths z and w fail can be expressed as:

$$H_s(p, z, w) + [1 - H_s(p, z, w)] \cdot H_{ts}(p, z, w). \tag{3.32}$$

Defining J_{zw} as the set of links that are shared by paths z and w, i.e., $J_{z,w} = \{(l, m) \in E : z_{l,m} = 1, w_{l,m} = 1\}$, and recalling that link (l, m) survives with probability $1-p_{l,m}$, i.e., all shared links survive with probability $\prod_{(l,m)\in J_{z,w}} (1 - p_{l,m})$, the probability that both paths z and w fail due to the failure of one (or multiple) shared links is:

$$H_s(p, z, w) = 1 - \prod_{(l,m)\in J_{z,w}} (1 - p_{l,m}) \tag{3.33}$$

$$= 1 - \prod_{(l,m)\in J_{z,w}} (1 - p_{l,m} z_{l,m} w_{l,m}) \,. \tag{3.34}$$

If we consider a vector w, then the complement with respect to the set E of all links $\hat{w}$ represents the links that are not selected for the back-up path w. Effectively, the intersection of the set z of links in the primary path and the set $\hat{w}$ of links that are

not in the backup path is the set of links in the primary path z that are *not* shared with the back-up path w. Thus, the probability that path z fails due to the failure of non-shared links is equivalent to the probability that links in the intersection of z and $\hat{w}$, i.e., that links shared by z and $\hat{w}$ fail. This failure probability can be expressed as $H_s(p, z, \hat{w})$. Analogously, $H_s(p, \hat{z}, w)$ represents the failure of the backup path w due to the failure of non-shared links. In such a scenario, the total failure of non-shared links can be given as $H_{ts}(p, z, w) = H_s(p, \hat{z}, w) \cdot H_s(p, z, \hat{w})$.

Thus, the overall minimization of the JFP can be formulated as:

$$\min H_s(p, z, w) + [1 - H_s(p, z, w)] \cdot H_{ts}(p, z, w) \tag{3.35}$$

$$s.t. \text{ Constraint Eqn. (3.2) for } z \text{ and } w. \tag{3.36}$$

We can approximate Eqn. (3.35) by analogously following the steps Eqn. (3.1) to Eqn. (3.7) for the first term. The second term is the joint fail. prob due to the fail of links that are not shared. The formulation is a standard ILP problem and since its NP complete, we approximate it as shown in equation 37. This is the low failure probability regime that we consider so, the $\hat{\beta}_{l,m}$=1, only if $z_{l,m}$ and $w_{l,m}$ =1 i.e. link is shared.

$$\min_{z,w,\beta,\hat{\beta}} \sum_{l,m} p_{l,m}\hat{\beta}_{l,m} + \sum_{l,m}\sum_{a,b} p_{l,m}p_{a,b}\beta^{a,b}_{l,m} \tag{3.37}$$

$$s.t. \text{ Constraints Eqns. (3.1) and (3.4)} \tag{3.38}$$

$$\hat{\beta}_{l,m} \geq z_{l,m} + w_{l,m} - 1 \;\; \forall (l,m) \in J \tag{3.39}$$

$$\hat{\beta}^{a,b}_{l,m} \geq z_{l,m} - w_{l,m} + z_{a,b} - w_{a,b} - 1 \; \forall (l,m),(a,b) \in J. \tag{3.40}$$

Note that the constraints (3.39) and (3.40) exploit vector arithmetic, i.e., $\vec{\hat{w}} = \vec{1} - \vec{w}$; thus, vector $\hat{w}$ only includes the links that are not chosen for path w. Hence, the probability that path z fails due to the failure of non-shared links equals the

probability that both $\hat{w}$ and z fail due to the failure of both $\hat{w}$ and z. Thus, the 1's in constraint (3.39) [resp. (3.40)] can be interpreted as the probability that path w (resp. z) fails due to the failure of non-shared links.

In constraint (3.39), $\hat{\beta}_{l,m} = 1$ only when $z_{l,m} = w_{l,m} = 1$, which is the condition sharing a link. The first term in Equation (3.37) is the JFP due to the shared link failure. In constraint (3.40), $\hat{\beta}^{a,b}_{l,m} = 1$, when $z_{l,m} = w_{a,b} = 1$ and $z_{a,b} = w_{l,m} = 0$, which means that the links (a, b) and (l, m) are used by paths z and w, but these links are not shared. Thus, the second term of Eqn. (3.37) models the JFP due the failures of non-shared links.

Analogous to the problem complexity in Section 5.2, the problem specified in Eqns. (3.37)–(3.40) is NP complete. Therefore, we design a greedy solution in Algorithm 4. The key difference in this algorithm is that for a backup path w, the weight of each link is set to the JFP because of the failure of link (l, m) and the links in z. Thus, there are two cases considered:

1. If link (l, m) is not selected as the primary path z, then its weight $\kappa_{l,m}$ is set to the product of the failure probability of link (l, m) and the approximated failure probability is $\sum_{(a,b)} p_{a,b} z_{a,b}$ of z.

2. If link (l, m) was selected by path z, then for a joint failure to occur, even path w would have had to select the link (l, m), so the weight of link (l, m) is $\kappa_{l,m} = p_{l,m}$. The shortest path will minimize the JFP and will be used as the backup path w.

Clearly, if the link (l, m) is shared, then its weight should be set $p_{l,m}$ which the first order value, which is higher than the second order weight in the non-shared scenario. Through this we observe that links will low failure probability are almost always shared.

Algorithm 4: Shared Link Greedy Algorithm (TSN_{u2} without disjointedness)

Input: Initialize: $\kappa_{l,m} = p_{l,m}$

Output: Shortest paths z and w with min JFP

Calculate the $p_{l,m}$ when $z_{l,m} = 0$ and 1, for $\kappa_{l,m} = p_{l,m}$;

Find shortest path z;

if $z_{l,m} = 0$ **then**

    $\kappa_{l,m} = p_{l,m} \cdot \sum_{(a,b)} p_{a,b} z_{a,b}$;

if $z_{l,m} = 1$ **then**

    $\kappa_{l,m} = p_{l,m}$;

Search for shortest path w;

3.6 Evaluation

3.6.1 Simulation Setup

Compared Approaches and Benchmarks

We compare the Algorithms 1, 2, and 3 designed in this paper with a brute-force solution implemented via a Python optimzer called Gurobi Optimization solver, which we call "Brute Force" and with the state-of-the-art. RFT-TSN approach Feng *et al.* (2022). We refer to our Algorithm 3 via Lagrangian approximation as "LA". We call our two greedy algorithms which approximate the ILP as TSN_{u1} (Algorithm 2) and TSN_{u2} (Algorithm 4). As we will see eventually that both greedy algorithms TSN_{u1} and TSN_{u2} perform very similarly, we resort to use of TSN_{u1} in all the plots unless specified. We evaluate the protection quality via the path failure probability and the algorithm run-time. For the LA based iterative algorithm, we chose a maximum

number of $I = 2 \cdot 10^4$ iterations, with a step size of $10^{-9}/\sqrt{i}$ here $i,\ i = 1, 2, \ldots, I$, denotes the iteration index.

In order test if we could potentially maintain time-lines even at the time of failure we prepare a set up as follows. As before, we have time slot chosen for each flow from the set 10,20,50,100μs, this is also the deadline for completing the flow or the length of the flow. The size of bytes transported as payload is 1500 bytes and specify zero propagation delay. The TTE structure has a flag which is set to 0 for time-triggered flow, 1 for non-time triggered flow. Finally, we randomly chose flows from [1,500] set of flows.

Network Model

We generated over 100 random graphs with a maximum node degree of five. Further, we always check for 3-connectedness. If 3-connectedness does not exist, then we discard the graph. Set distances between two nodes(its again a construct NetworkX provides, you can set distances manually. The difference however is brought by connectedness. We choose only 3-connected graphs again to maintain the flow deadlines. Each flow deadline is chosen from $\{10, 20, 50, 100\}\,\mu$s at random). Two nodes are connected if distances between each other is less than 0.5 We choose a maximum node degree of 5. We consider single source to destination pair, whereby source and destination are uniformly randomly selected.

Link Failure Simulation

We assign the link failure probabilities

$$p_{l,m} = \epsilon[\nu + (1 - \nu)U], \tag{3.41}$$

where U is an independently uniformly randomly drawn rational number between (0,1) for each link (l, m). A small ν value corresponds to a low failure probability regime, whereas large ν values correspond to a high failure probability regime. We vary the value of ν between 0 and 0.9; whereby $\nu = 0$ implies a uniform random link failure probability regime. Furthermore, ϵ is a fixed factor between [0,1].

For the simultaneous (shared) link failures, 20 correlated link groups are uniformly randomly generated for every network topology graph and their failure event probabilities are set to uniformly distributed random numbers so that their failure probabilities sum to one.

Simulation Process

Each independent simulation replication proceeds as follows. We first generate the network.

We conducted 100 independent replications for each parameter setting and condition or approach and report averages of the values obtained for the 100 independent replications (which involved 100 random network topologies). The 95% confidence intervals on the reported averages are less than 5% of the corresponding sample means and are not plotted to avoid visual clutter.

3.6.2 Results

Independent (Non-shared) Link Failures

Shared Link Failures

In Figure 3.2, We evaluate the variations of the lower probability regime and higher probability regime by changing ν from 0 to 0.9 and observing the path failure probability. We see that the probability based path selection approximates the ILP accurately.

We compare the two approaches TSN_{u1} with the brute-force GUROBI solver solution along with the Lagrangian approximation (LA).

We plot Figure 3.6 with the disjointedness constraint and observe the joint failure probability achieved by TSN_{u1} and LA. We see that GUROBI solver always achieves the best possible path and the TSN algorithm and LA achieves the same protection levels. However, without the disjointedness constraints, the brute force solution still performs better than others but LA begins to deteriorate pretty sharply. One important observation to make here is the performance of our greedy algorithms which perform as well as the GUROBI brute force solution. This fits our claim made in section 3 that for path preservation, it is important that the primary and back up paths pair has the best path. Further, for the solution obtained via Lagrangian approximation we see that performance deteriorates for larger networks. This is mainly because, LA requires more iterations to find a solution. This is what we observe in Figure 3.5.

In terms of run time it is obvious that GUROBI takes an exponential time to solve, whereas the greedy solutions are relatively faster, as examined in detail in Figure 3.10. We observe that when we have a probability assigned for the SRLG i.e., $p_{l,m} \in (0.5, 1)$ and $p_{l,m} \in (0, 10^{-}3)$ for high probability regime in Figure 3.4 and low probability regime in Figure 3.3, respectively. In this scenario, the GUROBI performance is equally well in low and high probability regimes. This is intuitive because the approximation obtained by the GUROBI solver is the lower bound for the JFP. The greedy algorithms perform failure probability adjustment before making the choice for the second path to minimize the JFP. This is the reason why TSN_{u1} has a bit higher failure probability when disjointedness constraints are taken into account.

In Figure 3.7 for a test case we generated 20 SRLG for every graph and the

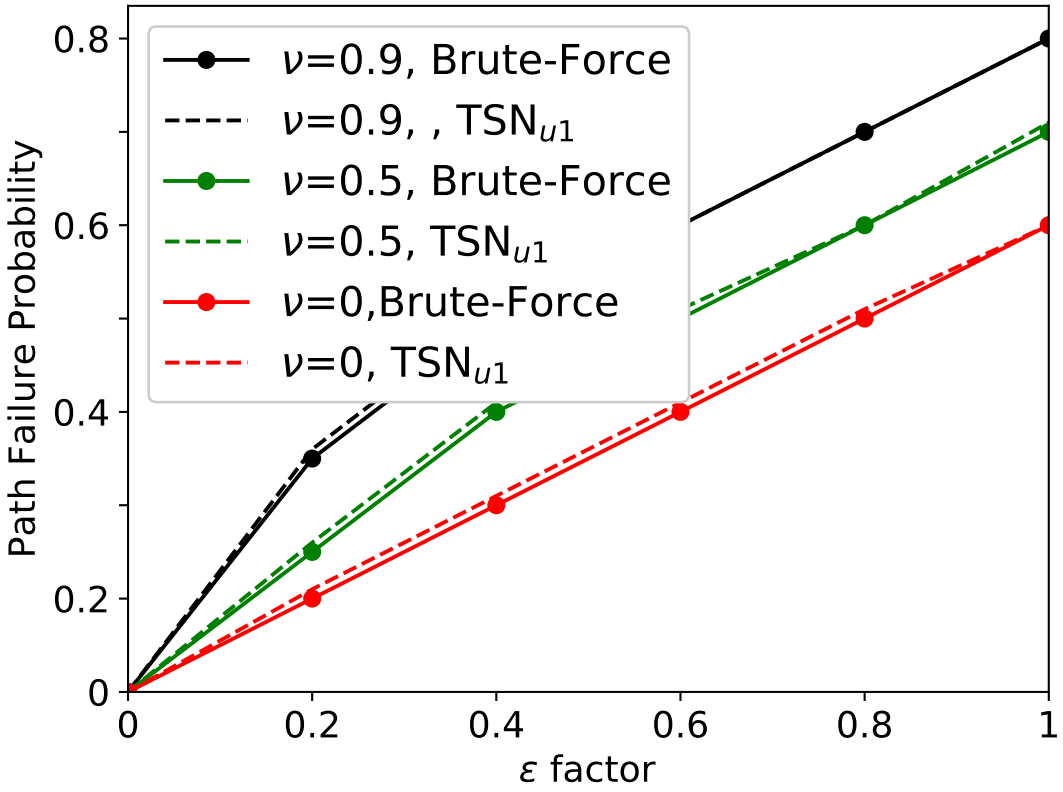


Figure 3.2: Comparisons of Path Failure Probability for Brute-force Ilp Solution and TSN$_u$1.

failure event probability are distributed as elaborated before $p_{l,m} \in (0.5, 1)$ and $p_{l,m} \in (0, 10^{-}3)$ for high probability regime and low probability regime, respectively. It was observed that the JFP was decreased by removal or relaxation of the disjointedness constraint. A very well justified claim is successfully shown to be true here as the greedy algorithm outperforms state of the art joint shortest path approach when introduced in traditional TSN setting of Balasubramanian *et al.* (2020). This is because of the inclusion of the best path as explained before.

We also observe in Figure 3.7 that greedy algorithms perform very similarly and still provide a better protection than the traditional one shortest path per cycle

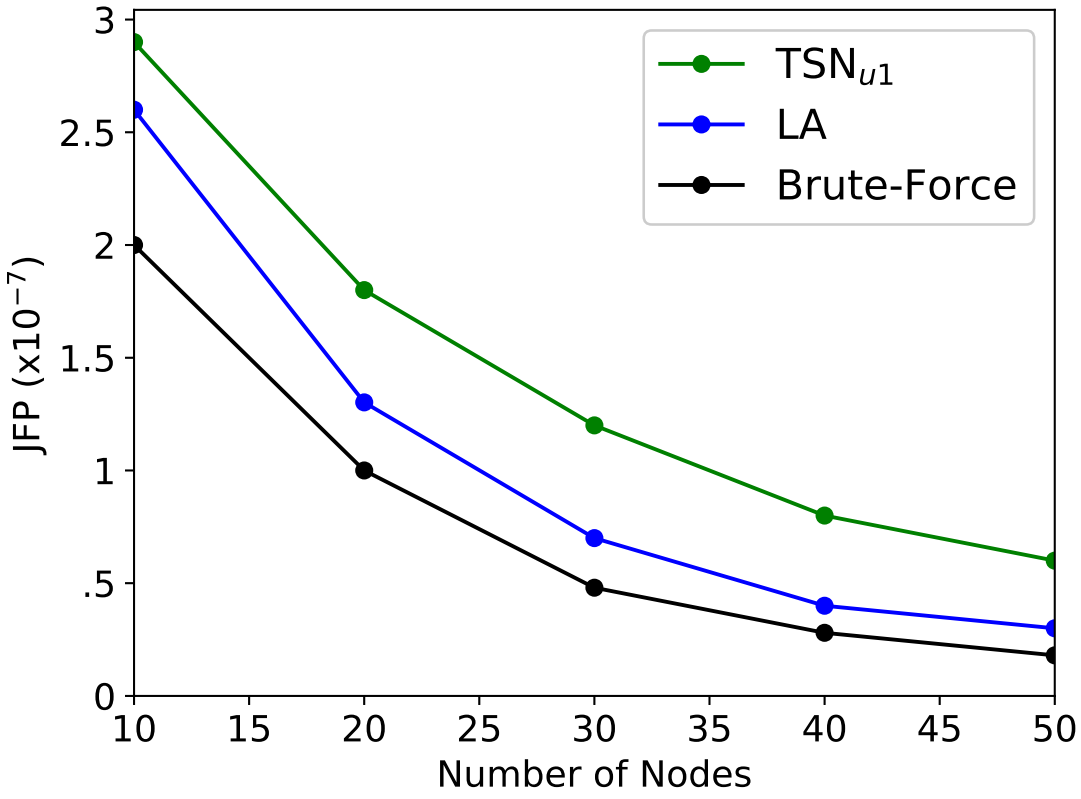


Figure 3.3: Jfp for Independent Link Failures (with Disjoint Primary and Back-up Paths) in Low Probability Regime ($\nu = 0, \epsilon = 10^{-3}$).

approach known as "one-by-one" shortest path instead of pair-wise allocation. This is justified because our TSN$_{u1}$ and TSN$_{u2}$algorithms adjust the failure probability before making a choice of the second path w. This also leads to less wait time for re-connection. Further, the choice of the second path is such that the overall joint failure probability is reduced, hence out performing the traditional algorithm. Further, in Figure 3.9 we observe the change in utilization of the link given as the ratio of total flows to the bandwidth which is set at 100 Mbps. The $RFT - TSN$ represents the reproduced algorithm mentioned in Feng *et al.* (2022). The key constraints in the $RFT - TSN$ include the reservations based on queue IDs. Due to this we have an

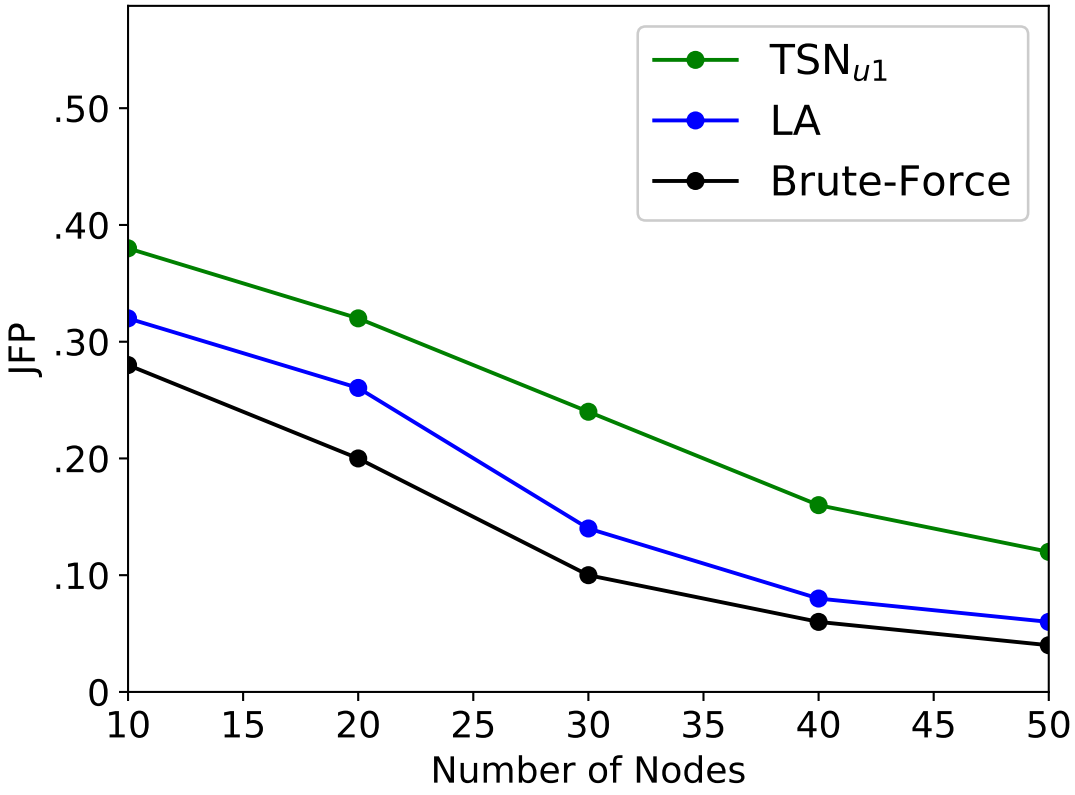


Figure 3.4: Jfp for Independent Link Failures (with Disjoint Primary and Back-up Paths) in High Probability Regime($\nu = 0.5, \epsilon = 1$)

entire interval reserved that allows only the queue IDs which are different to pass through the gates. This involves searching and sorting that adds overheads to the links which have been utilized. However, with TSN_{u1} we are always ready with a backup link which is a singular assignment rather than searching from a pool of links.

In Figure 3.10 we observe that the execution time is much higher for the $RFT - TSN$ this is justified because of the isolation constraints of the model. This means in-order to prevent the TT flows from overlapping there is an isolation introduced. However, in TSN_{u1} there are no flow based isolation which could cause overheads at the time of failures because we do not change the allocated slots after the scheduling is

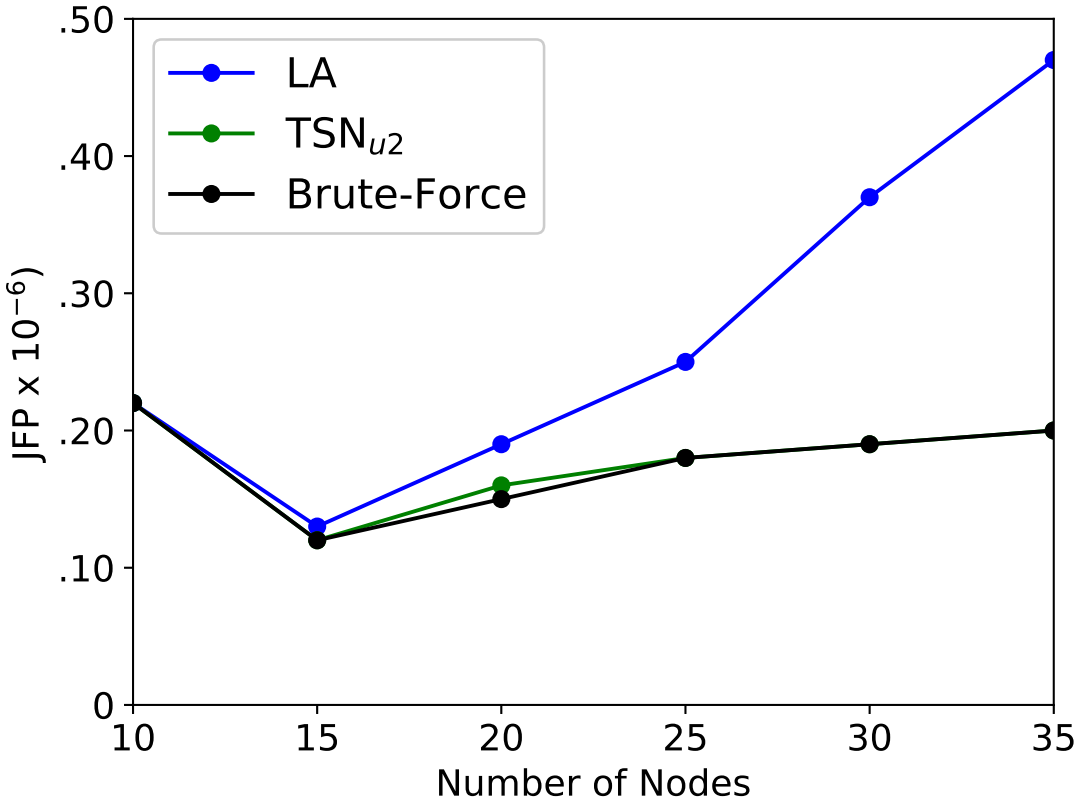


Figure 3.5: TSN_{u2}-model Without Disjointedness Constraints $p_{l,m} = 0.5$

done. Our design choice and belief is that once flows have been allocated to respective time-slots, changing them at run-time can cause issues related to overlapping routes and hence should be avoided. Additionally, this choice is justifiable owing to the much less constraints which need to be satisfied unlike the $RFT - TSN$.

In Figure 3.11 we observe the drawbacks of sending extra bits of CRC that causes further increase in the amount of resources that is wasted. That is another queue is allocated for the CRC (ACK or NAK), so the transmission is reserved again for these messages to be successfully.

Through these experiments there was one factor which was very clear, the space and time redundancy strategies come with a trade-off. This trade-off affects the

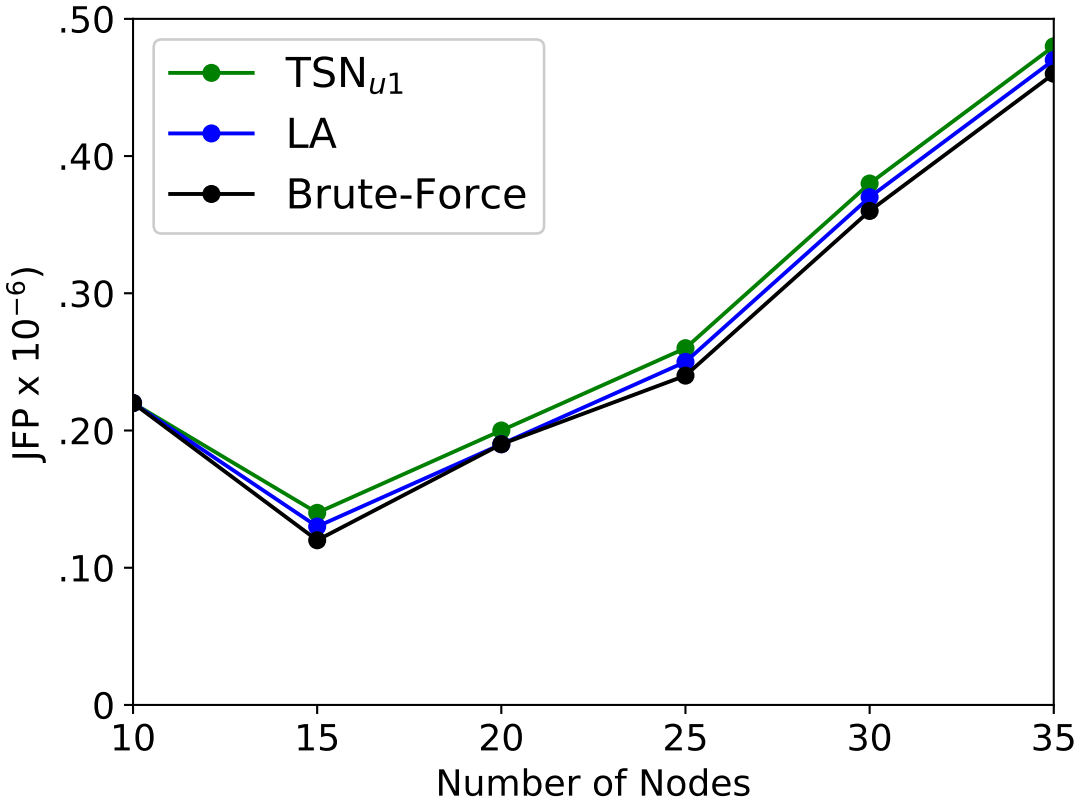


Figure 3.6: TSN_{u1} with Disjointedness Constraints $p_{l,m} = 0.5\ \forall l, m \in E$

same questions as for Fig. 5.

very core of time sensitive industrial networks. That is, the time sensitivity or the time boundedness. We notice that although the delay parameters are not completely compliant with the TSN boundaries, fault tolerant mechanisms might sometimes play out as expected while considering scheduled traffic. In Figure 5.7, we observe these characteristics. When the injection rate is low, the overall delay appears to be not so drastic i.e. remains below 10 cycles. However, we can observe the shortcomings of schemes like LA as injection rate increases to 0.6 This is mainly because of the wait times for a link to reconnect when there is no back-up path or when the overall "pair-wise" path allocation is not found. This causes a higher communication delay.

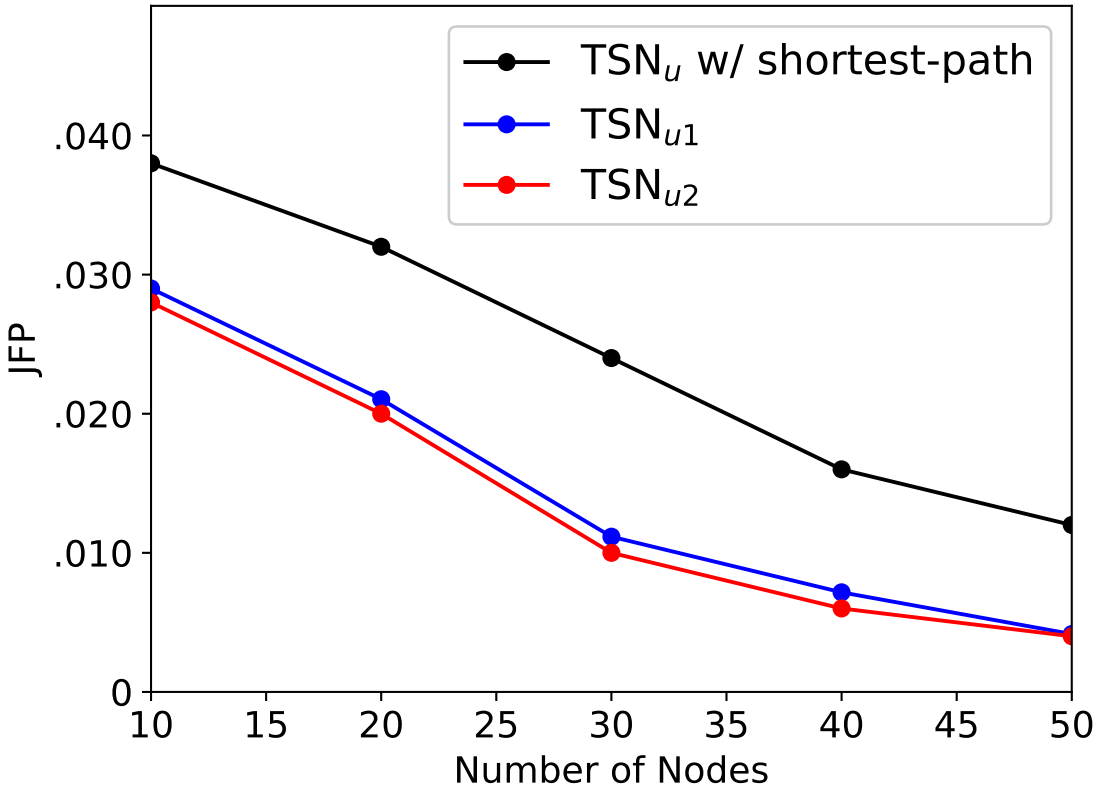


Figure 3.7: Comparing Failure Probability of Greedy Algorithms TSN$_{u1}$ and TSN$_{u2}$ vs. one-by-one shortest path injected in Balasubramanian *et al.* (2020) labeled TSN$_u$ with shortest path.

3.7 Conclusions and Future Work

In this paper, we deeply investigated sustainability in the IoT 4.0 scenario. We observe various path preservation problems with correlated failures in a factory floor. To this end, it is important that efficient strategies are developed to take into account complex behaviors. Primarily, the policies developed should not simply provide protection via disjoint paths as these may not be always reliable. We show event where the unreliability of such policies are observed and provide robust solution. In our

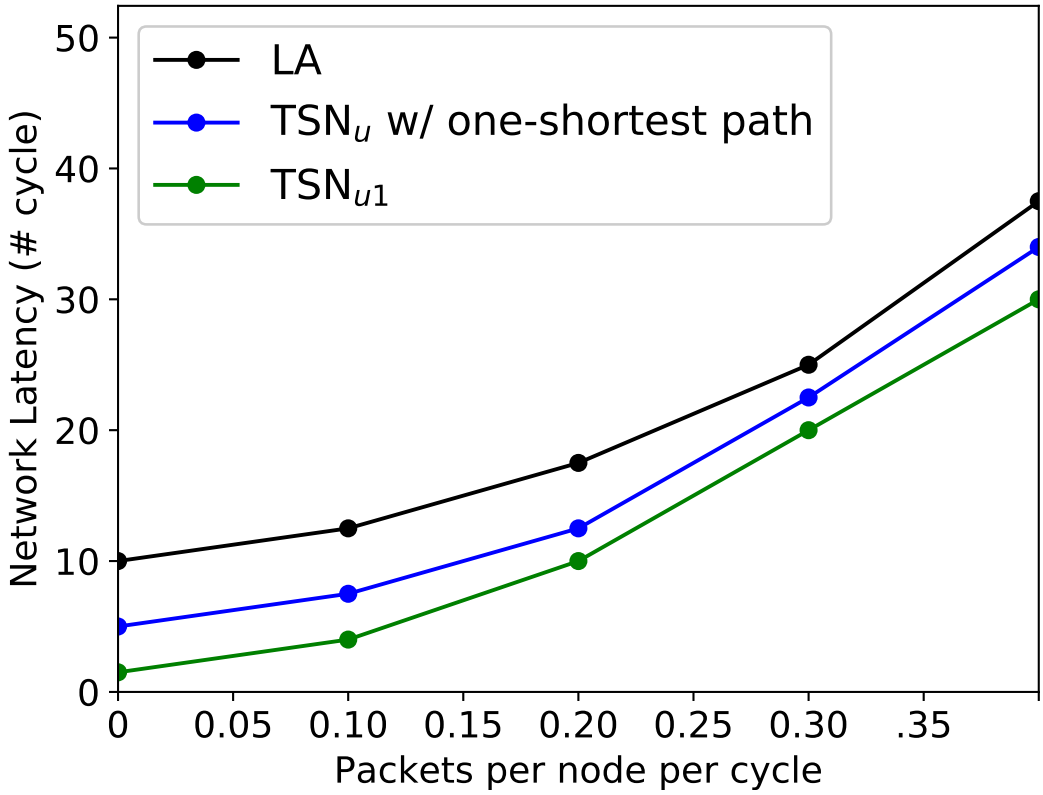


Figure 3.8: Latency Comparison

solution we consider minimum joint failure probability. In this model we consider shared risk link group protection schemes and show how Time sensitive networks gating parameters affect the overall system. Using linear approximations we developed heuristics that show reliable performance.

In the future it will interesting to find out how graph convolutional networks can enable predictive routing under similar constraints of disjointedness. Further, as graph convolutional networks (GCNs) enable a semi-supervised network, it will be our endeavour to find out if through that semi-supervised time sensitive networks can be realized.

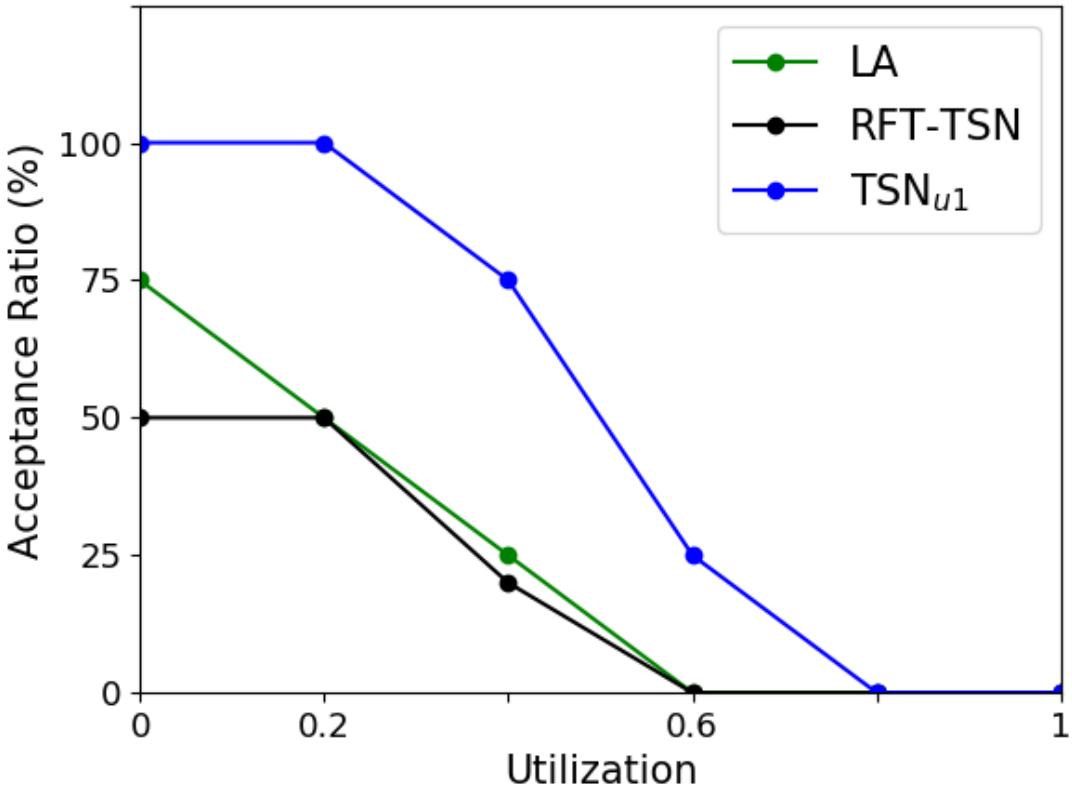


Figure 3.9: Acceptance Ratios as Function of Utilization

$$\nabla^+(S) = (l, m) \in E : l \in D, m \in S \tag{3.42}$$

$$\nabla^-(S) = (l, m) \in E : l \in S, m \in D. \tag{3.43}$$

Since, the graph is bidirectional, $|\nabla^+(S)| = |\nabla^+(S)|$, e.g., if we consider a simple src-dst path Q, then $|\nabla^-(S) \cap Q| - |\nabla^+(S) \cap Q| = 1$. Suppose that

$$|\nabla^-(S) \cap Q| = l + 1. \tag{3.44}$$

This means the path Q touches $l + 1$ number of links in $|\nabla^-(S)|$. Since the graph is bidirectional, we also have $|\nabla^+(S) \cap Q| = l$, which implies at least l number of links

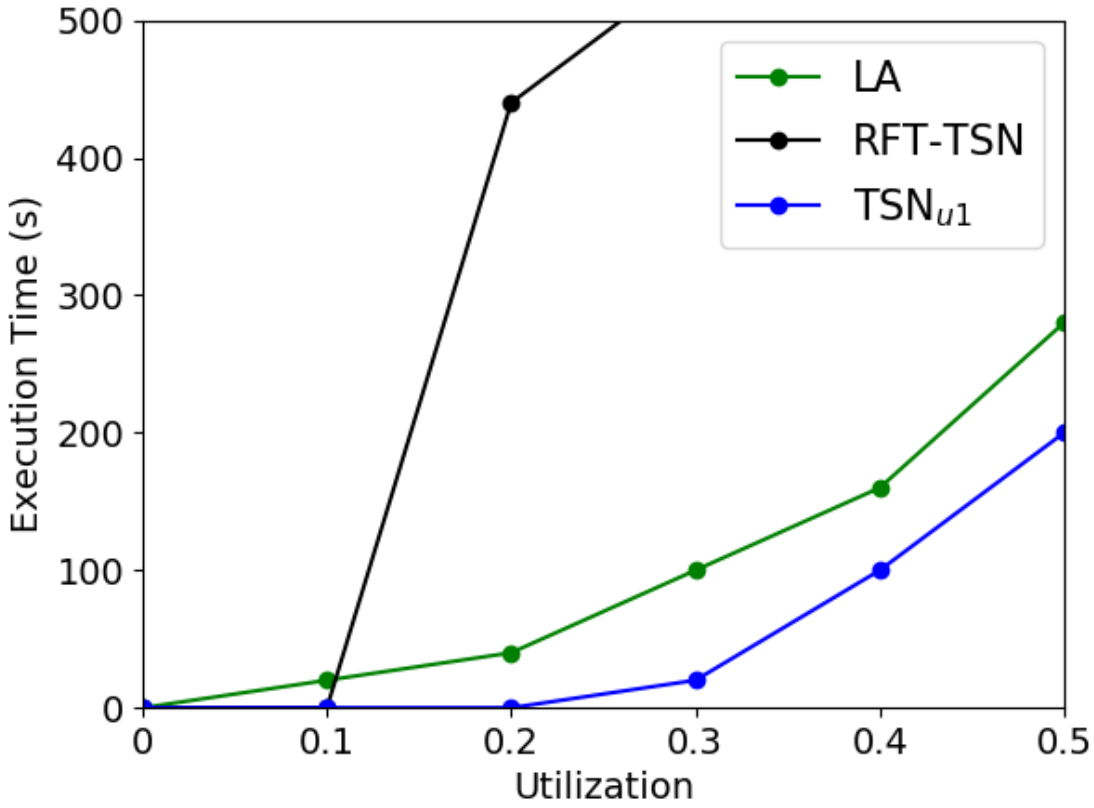


Figure 3.10: Comparison of Execution Times

in $|\nabla^-(S)|$ are *not* touched by the path Q. Further, $l \geq 1$ ensures that at least one link that is going out from the source will remain after all links have in Q have been removed. If $l = 0$, then $|\nabla^-(S) \cap Q| = 1$, means that only one link is removed. Since, we have the connectedness such that $|\nabla^-(S) \geq 2$, we have at least one outgoing link that is always present, i.e., at least one link will survive, thereby enhancing the sustainability.

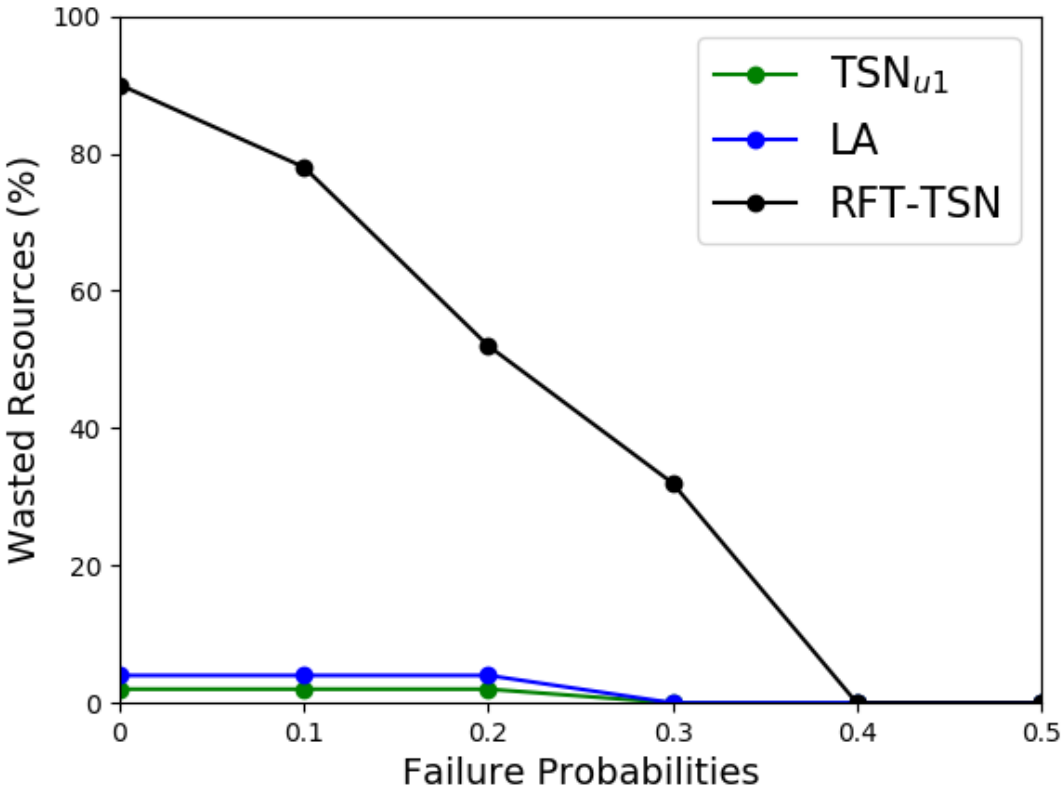


Figure 3.11: Resource Wastage with Failure Probabilities

Chapter 4

FEDCO: NETWORK CONTROL PLANE FOR THE EDGE

4.1 Introduction

In order to de-congest the backhaul network, the management of content caching has been closely investigated in the last decade. Further, reducing the delay for the last-mile users has been a major research focus. To this end, placing popular contents closer to the user so that frequently requested files can be retrieved faster, has been a key endeavour for many engineering applications that have been developed post-2015. There are two prevalent models studied in the edge caching paradigm, namely the costly edge data center (EDC) model Brik *et al.* (2020) and the low cost 5g-D2D model Balasubramanian *et al.* (2019). Both of these models deployments have their own pros and cons Long *et al.* (2018); Balasubramanian and Karmouch (2017); Balasubramanian *et al.* (2019). In general, there is broad agreement that MEC deployments are costly, especially when they serve demand of multimedia services in upcoming 5G networks.

Recent research has begun to involve the mobile equipment (MEs) nodes in the multimedia delivery loop via Device-to-Device (D2D) communications. In this paradigm, MEs lend their own storage resources Habak *et al.* (2017)Li *et al.* (2017a) to serve each other via D2D links, thereby flexibly enhancing the edge cloud capability and scalability. In such models, all mobile devices already have a 5G subscription and MAC-IDs which are known centrally by a base station (owned by an MNO). The client requesting a service forms a D2D resource composition using the idle resources of close-by devices. Note that MEs increasingly feature acceleration modules

for network-intensive functions Linguaglossa *et al.* (2019); Shantharama *et al.* (2020). Such a resource-rich environment of MEs replacing the expensive MEC is called Mobile Device Cloud (MDC) Ferrer *et al.* (2019). However, despite the MDC promises, privacy issues are still prevalent. Further, users uploading contents to a central datacenter inherit the limitations of the existing cloud infrastructure, such as single point of failure, latency, redundancy, and security risks. Motivated by the need to comprehensively address these issues, we propose a model that carries the benefits of edge entities with an underlying Federated Learning (FL) technique.

Recent efforts such as in Wang *et al.* (2019), study the case of distributed machine learning algorithm wherein parameters are distributed across multiple edge nodes. In this model, raw data transmission to a centralized location is not considered important rather, the gradient descent results from each local node are updated and averaged at a central place. The study Wang *et al.* (2019) shows that using such an environment results in close to optimal results in terms of training time with a given resource budget. Extrapolating this distributed FL to an MDC environment, we show in our model that the users at the lowest layer may request services from a 5G-D2D MDC (i.e registered with the MNO) or an EDC collocated with the 5G-NR (New Radio) base station. Having multiple content placement locations, not only forms a convenient business model, but also empowers the customers to select judiciously the QoS level that they would like to pay for. As most of the QoS related metrics depend on where the popular content items are placed, a close examination of the content placement is needed. Further, as both of these service locations (i.e., the EDC and the MDC) are registered with the MNO, the question of where the content with a high demand should be placed influences the MNO's profit margin.

A controller which is situated at the edge, e.g., within a backhaul architecture King *et al.* (2019); Shantharama et al. (2018), makes the key decisions of content placement.

As shown in Figure 4.1, when each user processes the FL from the controller for training the model with the local data, the main goal of the end user is to upload only the necessary model parameters' updates. After updating the parameters, the recommendation for the content is registered in the controller. This recommendation is then used to place the content in an appropriate resource block (i.e. a device cloud or an EDC). The edge controller is responsible for aggregating the information from the users and the choice of the most popular content. This translates into understanding the QoS levels of the user, thereby resulting in the maximization of the MNO's revenues. Additionally, by keeping the locally trained data with the end-user, security risks are minimized. In line with these two objectives, the contributions of this paper are:

- We design a hierarchical architecture for multimedia content delivery and caching management where the MNO places the 5G-D2D compositions in different strategic areas along with other edge computing entities, allowing users to select their offloading points.

- We formulate a business model for the MNO to maximize its revenues with the key constraint of satisfying the user's QoS. The management is supported by a FL algorithm that predicts the users' demands for specific content in order to provide the exact location for its placement.

- Extensive simulations were conducted to show the effectiveness of the proposed model compared to two state of the art models, namely Random Caching Xu *et al.* (2018) and Edge-Boost Balasubramanian *et al.* (2019).

The rest of the paper is organized as follows. Section 5.2 delineates the novelty of the proposed solution with a brief discussion of the most recent related work.

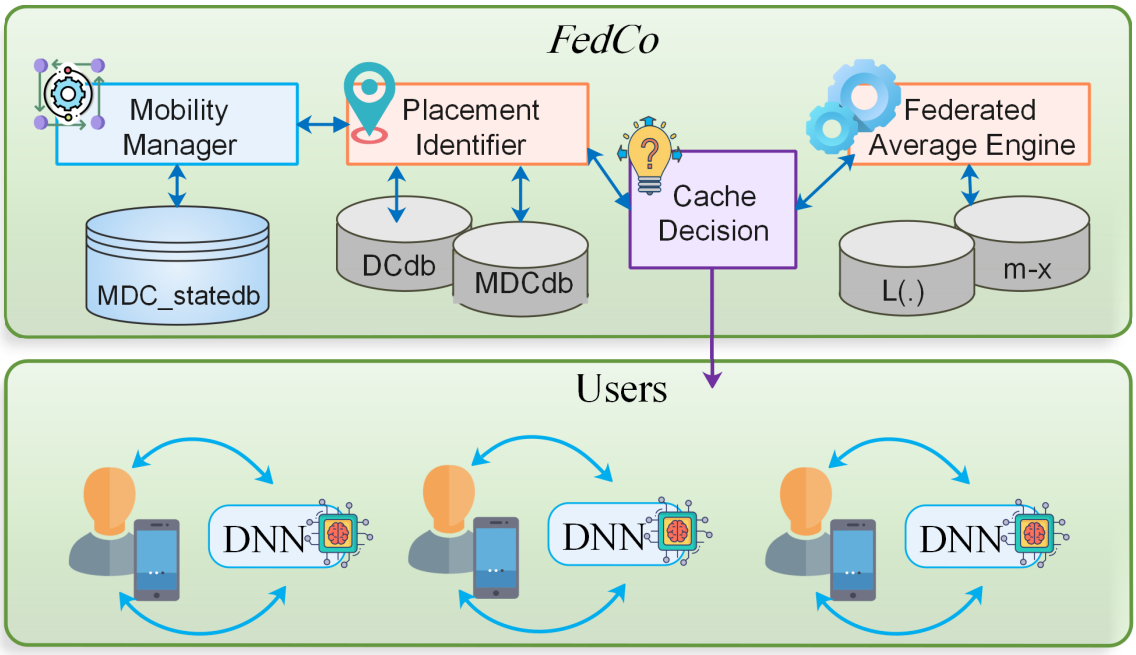


Figure 4.1: When Each User Downloads the Fl Framework from the Controller For Training the Model with the Local Data Marked as Red Arrow, the Main Goal of the End User Is to Upload Only the Necessary Parameter Updates.

Section 4.3 presents the system design and the associated problem formulation. Section 4.4 presents the performance evaluation and Section 5.6 provides concluding remarks.

4.2 Related Work

4.2.1 Software Defined Network (SDN) Paradigm

It is challenging to address the difficulties in a network whose mobility characters are not known. In Wang *et al.* (2020) Wang et al. capture the ubiquity of operations at the edge that encompasses state of the art models. Authors discuss the key notions behind the merger of software defined networks, artificial intelligence and Deep Learning implementation at the edge. An SDN usage in wireless network is proposed

in Detti *et al.* (2013). In this architecture, a cluster head is chosen that acts as a controller. Similar to these models, our control plane logic utilizes the OpenFlow protocol to feed control information via the south bound interface to the data plane. Figure 4.2 shows how the collocation of the BS-controller assists in reliably placing content.

4.2.2 Edge Caching Models and Federated Learning

Extensive studies have recently been conducted for replacing the edge caching capacity by device storage resources via D2D links Ferrer *et al.* (2019)Mehrabi *et al.* (2019). For instance, the authors in Zhang *et al.* (2018) have utilized mobile vehicles as smart caching agents to offload the caching tasks from the BS using a vehicular edge structure. However, the random vehicular movements had not been considered. Neglecting the vehicular movements significantly impairs the caching demand estimation, which in turn negatively affects the caching performance. Similarly, studies Wu *et al.* (2018) Li *et al.* (2017b) proposes a distributed caching framework based on the D2D assisted caching paradigm. The main difference between the two is that in Li *et al.* (2017b) a delay-aware caching algorithm over D2D links is proposed that locates the best carrier. The key idea is to minimize the transmission delay and improve the throughput. These solutions regulate the caching capacity but ignore the demand variations.

In Park *et al.* (2017), an overlay structure is employed to effectively search for content providers in D2D networks without having a reliable content popularity estimate. This would in turn result in a sub-optimal revenue generation for the operator. In Deng *et al.* (2020), Deng et al. discuss a roadmap for how edge computing and the interdisciplinary fields of AI and Machine Learning together brought about change in various communications and computations aspects at the edge. The main insights

relate to how devices resource constraints can be overcome training machine learning models at the edge. However, this work ignores some key aspects of revenue generation that is directed towards the MNO who is deploying such a service.

4.2.3 Related Federated Learning (FL) Research

FL is a branch of machine learning that enables decentralization of the training model by letting the end users be part of the training and prediction loop. An over-the-air computation model that exploit the super-position property of wireless channels to aggregate data has been proposed in Yang *et al.* (2020). This is one of the preliminary models that provides a close observation of applying FL in wireless networks. The study Sacco *et al.* (2020) targets a unique FL problem in *challenged networks*, where an LSTM model is in place to take the inputs. The output obtained here is the routes that are feasible in disaster management scenarios or challenged scenarios as the authors define it. However, this model cannot be reused in cases where we need to search for computation sites to determine satisfactory QoS levels for users. The study Yu *et al.* (2018) proposed a FL based proactive content caching (FPCC), which is a hierarchical architecture where users upload only the requisite updates to the edge server and keep all the remaining sensitive data on their devices. However, due to the complex nature of the model, a scalable deployment might not be possible. In contrast, our model is purely based on the probability of outcomes that is judged via a predictive user-behaviour model.

In Chen *et al.* (2021), Chen et al. study FL training models in wireless networks. In this work, the authors study the various network parameters that come into play while the local FL models are transmitted to the Base Station (BS) that aggregates them and maintains the global FL model. They formulate an optimization problem capturing wireless network parameters and users choices. We take inspiration from

this work for the MDC scenario where there are network factors that come into play while uploading the local FL models to the central controller situated at the base station. The key difference from that work is that our interest is towards producing a seamless content placement/retrieval environment for cloud users which not only benefits the MNO but also provides a better Quality of Experience (QoE) to the user. To that end, while training, we ensure that our algorithm produces optimal results while minimizing losses.

Zhao et al. in Zhao *et al.* (2020), elaborate on the use of Federated Radio Access Networks where the combination of edge computing and AI is explained. Fundamentally, authors study the use of loss functions at the time of training and the commensurate reduction in prediction accuracy over a period of time with learning enabled Radio Access Networks. Similar to this Vu et al. Vu *et al.* (2020) proposes a FL supported MIMO framework that optimizes the local accuracy, transmit power, processing frequency and data-rate. The key idea behind this paper is to study the complex non-convex behavior of the training time, computation and transmission of the computed values. The two FL models cited above have similarity in training time and accuracy predictions but differ in the overall complexity. On the other hand, although we evaluate a similar scenario with a central controller collocated with the base station, our interest is driven by multiple hierarchical controllers. The training time reduction caused by the presence of these controllers is much better compared to the other models.

Park et al. in Park *et al.* (2019) show theoretically and via simulations the different blocks of an ML based network edge, discussing how training and inference occurs when devices share their local training models with the base station in wireless networks. Similar to the above two models, a distributed, low-latency and reliable ML model is proposed that incorporates different neural network architectural splits.

While our work focuses on training and accuracy, at the same time we ensure acceptable placements of the content for the other devices to retrieve from. This not only reduces the latency but also reduces cost for the users, due to the decentralized nature of the system.

In Chang *et al.* (2019) Chang et al. propose an adaptive content delivery framework that is built on a Q learning technique for video streaming at the edge. The authors prove how quality of experience and fairness improves with such a design. Further, this work considers HTTP Adaptive Streaming (HAS) video chunks for its use cases, which enables the various adaptation schemes (such as Buffer-based adaptation, Rate- Based Adaptation and DASH based adaption) implemented in the paper.

In Li *et al.* (2020) Li et al. propose a deep neural network framework called Edgent that adaptively partitions a job among different computation entities. Authors make use of a predictive strategy that enables low latency communication with the edge that works seamlessly in static and dynamic scenarios. Specifically, they propose an online point detection algorithm that accommodates changing bandwidth conditions in the last mile networks. However, the execution plan proposed for the MNO is costly as the edge intelligence is distributed across devices that is outside the control of the operator. Further, job partitioning might not always result in proper resource retrieval as there are many false positives which occur in the very beginning of the training period.

4.3 FedCo Model

4.3.1 Design

The proposed model is depicted in Figure 4.2. At the control plane, we have four important modules namely, the *Mobility Manager*, the *Placement Identifier*, the

Cache Decision making module, and the *Federated Averaging* module. The mobility manager plays the role of maintaining seamless sessions between the devices forming the MDCs. The state of a device is stored in the $Mdc\_statedb$. Every time a device moves out of a location, a state variable associated with the device is updated. The placement identifier keeps a log of where a particular content is placed, i.e., MDC or the EDC. It makes use of the on-going procedural calls to the MDCdb and the DCdb to understand the popularity of a content and manages the placement. The Cache decision is made based on the inputs received from the *Federated Averaging* module, that holds and computes the two important quantities namely the $L(\cdot)$ and m_s in our FL framework. In the data-plane, the users compute the results via a Recurrent/Deep Neural Network (DNN) (popularly called RNN) algorithm that will be uploaded to the control plane once computation is complete. We do not discuss the DNN algorithm (i.e. LSTM) implementation in depth due to space limitations.

4.3.2 System Model

We consider a typical 5G scenario where the end-users specify their demands to the controller. The content is stored in a set of virtual resources in the EDC or it can be split among the 5G-D2D MDC composition Balasubramanian *et al.* (2019). The concern is that the users request content delivery and the MNO is responsible for placing the content and providing the requisite service to the user. More specifically, the user u requests a content file j which could be placed either in an edge server or across an MDC composition. The MNO, which is housed along side the central controller, is responsible for placing the content across these resource points. The request from the end user u is for one such content file. We define three popularity classes $c \in \mathcal{C}$ for these content files:

Table 4.1: Summary of Key Notations.

Notat.	Meaning
$\mathcal{R}$	Set of resource points (locations)
j	Content item (e.g., a video segm.) to be placed
o_j	Amount of space in storage units consumed by a content item j
$\mathbf{V}$	Vector of content popularities
u	User
d_u	Deadline specified by a user u
$T_{u,r,j}$	Total time spent for serving a request
$\mathcal{P}(e_u)$	Price fct., i.e., price paid by the user depending on the QoS
$L(\cdot)$	Weighting function
$m(\cdot)$	Probability of attaining the highest QoS that is achievable for r resource units
$\chi_{(\cdot)}$	Prospect pairs

1. High demand, High popularity, $(c = 3)$
2. Average demand, Average popularity, $(c = 2)$
3. Low demand, Low Popularity, $(c = 1)$.

We assume that the MNO has some (external to our model) prediction that gives the popularity level c_j for each content file j, represented by a vector of content popularities $\mathbf{V}$.

We form a resource pool with the set of available resource points. We refer to the set of resource points (locations) as $\mathcal{R}$ and each resource point $r \in \mathcal{R}$ corresponds to a virtual resource which has some content items (files) in storage (see Table 5.2 for a summary of notations). Resource point $r \in \mathcal{R}$ has a storage capacity $s_r,\ s_r \leq s_{\max}$

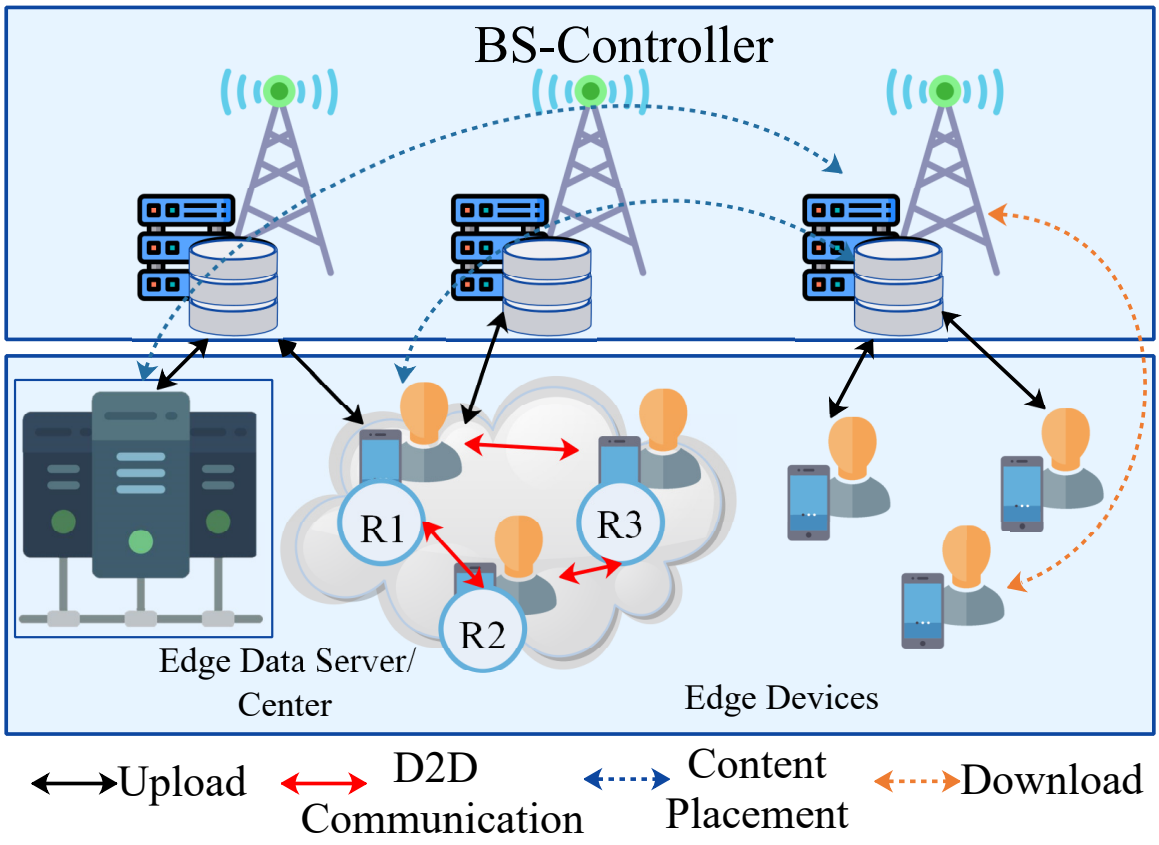


Figure 4.2: *FedCo* Federated Learning Controller Framework:The Edge Computing Units (Including the Idle Device Resources (R_1, R_2, R_3) and Data-center Servers) Are the Service Locations for Content Placement Which the Users Request

in terms of number of storage units. Placing a content item (object) j in any of these resource points requires (consumes) o_j storage units.

4.3.3 MNO Revenue Optimization Model

Now, we are considering this problem from an MNO's perspective: content placement and the resource allocation has associated costs, such as speed of computation and storage on a specific resource unit. To satisfy a customer's request, the MNO has to satisfy the end users QoS requirements. As each of these requests has a strict deadline to be met, the MNO has to complete the computation prior to a deadline.

Specifically, each user u is mapped to a deadline d_u. Once a content file j is retrieved, the user u may use it for a time $\tau_{u,r}$ during which o_j units are dedicated for this transaction on the resource pool. Denoting by $q_{u,r,j}$ the queuing time of the user u at the resource location r, the total time spent serving user u is $T_{u,r,j} = \tau_{u,r} + q_{u,r,j}$ In scenarios where requests for two contents of the same class occur, then the choice is dependent on the resource point r which can retrieve the content with the lowest queuing time. A key element here is that once the content delivery is complete, we map the task completion time to the request's deadline. The QoS satisfaction makes the customer pay more to the MNO for a future service.

In order to map this price division we define four main QoS classes for the user with κ_1 being the best QoS to κ_4 being the worst QoS. For each user u, the experienced QoS level e_u is defined based on the difference between the given deadline requirement d_u and the total service time $T_{u,r,j}$.

If $d_u - T \geq 2d_u$ then $e_u \in \kappa_1$. If $\frac{2d_u}{3} \leq d_u - T \leq 2d_u$, then $e_u \in \kappa_2$. If $\frac{d_u}{2} \leq d_u - T < \frac{2d_u}{3}$, then $e_u \in \kappa_3$. Finally, if $d_u - T \leq d_u/2$, then $e_u \in \kappa_4$.

These different QoS levels are mapped to different price levels v_1 to v_4 as per the service satisfaction. More specifically, the price function $P(e_u)$ represents the price paid by the user based on the service satisfaction level. We define this price function as

$$P(e_u) = \begin{cases} v_1, \text{ if } e_u \in \kappa_1 \\ \vdots \\ v_4, \text{ if } e_u \in \kappa_4. \end{cases} \tag{4.1}$$

As the value of e_u shows the exact user demands that need to be satisfied and its price, the FL prediction of e_u is our goal.

The model represents the multi-factorial dependence on the type of content placed, the deadlines met, and the QoS that the end user perceives. Let, $n_{u,r,c} =$

$\begin{cases} 1, \text{ if } u \text{ retrieves content of pop. class } c \text{ from res. } r \\ 0, \text{ otherwise.} \end{cases}$. A reasonable operator revenue model is:

$$H = \sum_{u \in \mathcal{U}} \sum_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} n_{u,r,c} \big(d'_u \mathcal{P}(e_u) - \chi_{r,c} \big) \tag{4.2}$$

where the binary variable $d'_u \in \{0, 1\}$ is set to 1 when the deadline time associated with a user u is met, i.e., d'_u is set to 1 when the deadline d_u associated with user u is satisfied. The cost $\chi_{r,j}$ represents the MNO cost price in deploying and provisioning a content caching resource unit at resource point r for popularity class c.

The accurate prediction of the user demand plays an important role in the MNO revenue maximization. It is important to note that the mobile end users who register for these services have a highly variable behaviour. It is therefore hard to predict a user's level of desired QoS.

In order to maximize the profit earned by the MNO, the goal is to place the content appropriately for maximizing the user's QoS which is dependent on producing the profit. Since higher levels of QoS can be achieved if the content placed is retrieved and processed faster within the completion times, to achieve the goal we need to accurately predict the demands of the users for the content. To realize this prediction, we make use of the FL framework. Therefore, the revenue maximization can be expressed as:

$$\max_{n_{u,r,c} \ \forall \ u,r,c} H \tag{4.3}$$

$$\text{s.t.} \quad \textstyle\sum_{j \in \mathcal{A}_r} o_j \leq s_r, \ \forall \ r \in \mathcal{R} \tag{4.4}$$

$$T_u \leq d_{c_u} \ \forall u \in \mathcal{U}; \ c_u \in \mathcal{C}. \tag{4.5}$$

The set $\mathcal{A}_r$ represents the contents j that are already allocated in resource location r. The first constraint ensures that all content items j that are on location r fit with their sizes o_j into the available space s_r. Thus, the first constraint enforces compliance

with the resource capacities of the various nodes of the different resource pool types (EDC and 5G-D2D MDC), i.e., the content is placed based on the knowledge of the resource point capacities. The second constraint ensures that for a user u who requests a content file j belonging to popularity class c_u, the time $T_{u,r,j}$ spent to serve a request from user u should be less than or equal to the deadline d_{c_u} specified by the user for the class c.

4.3.4 FL Model

As the problem is hard, mainly due to the presence of the last constraint, we use the FL framework to provide a heuristic solution. The FL technique performs machine learning based training on the end users and the output is then aggregated in a central location. The local controller is the first aggregation location. Each user u that requests a content file j belonging to popularity class c maintains historical data in the vector $\mathbf{B}_{u,c}$. This vector stores the information of the content that was recently requested. Each user is passed this vector in order to predict the next content it will request. The end user is now responsible for solving the ML model having a loss function that needs to be minimized. As noted in Yang *et al.* (2020), if we have a function $f_l(x)$, where l is a data sample, the output of $f_l(x)$ represents the error in data while training the model. If the users in $\mathcal{U}_c$ have requested a content belonging to class c, the loss function can be re-written from Wang *et al.* (2018) as

$$F_{u,c}(z) = \frac{1}{|\mathbf{B}_{u,c}|} \sum_{u \in \mathbf{B}_{\mathcal{U},c}} f_u(z). \tag{4.6}$$

The objective reduces to finding z^*. It is given as the $\arg\min F(z)$ which we solve with gradient descent. The presence of a hierarchical FL model enables training at the users locally with the aim of aggregation at the edge controller. The key here is the communication between the user's training model and the aggregation at the

controller within a specific number of iterations. We design the following Algorithm 5 using the steps in gradient decent to update the quantity $z_{i,c}(\alpha)$, where α is the number of iterations. We have,

$$z_{u,c}(\alpha) = \hat{z}_{u,c}(\alpha - 1) - \beta \nabla F_{u,c}(\hat{z}_{u,c}(\alpha - 1)), \tag{4.7}$$

where β is the learning rate and $\hat{z}_{u,c}(\alpha-1)$ represent the global aggregation. Once this information is passed to the edge controller we have the weighted average calculated as

$$z(\alpha) = \frac{\sum_{u \in \mathcal{U}} |\mathbf{B}_{u,c}| z_{u,c}}{\sum_{u \in \mathcal{U}} |\mathbf{B}_{u,c}|}. \tag{4.8}$$

Algorithm 5: *FedCo*

Input: Users $\mathcal{U}_c$ requesting a service of class c

Output: Weighted Average $z(\alpha)$

1 Calculate the $z_{u,c}(\alpha)$; $z_{u,c}(\alpha) = \hat{z}_{u,c}(\alpha - 1) - \beta \nabla F_{u,c}(\hat{z}_{u,c}(\alpha - 1))$

2 Calculate the weighted average $z(\alpha)$; $z(\alpha) = \frac{\sum_{u \in \mathcal{U}} |\mathbf{B}_{u,c}| z_{u,c}}{\sum_{u \in \mathcal{U}} |\mathbf{B}_{u,c}|}$

3 Placement Strategy:

Input: Content Popularity Vector $\mathbf{V}$, QoS levels e_u

Output: Content Placement Location

4 Calculate : $L(v_{\kappa_1,c,j,r})$

5 Update: $m(x_{u,c,r,j})$

6 Repeat Calculation of $L(\cdot)$ until all content items have been placed.

The main benefit of FL is inheriting the advantages in securing the privacy of the end user. To that end, we have the user updating the quantity z_u. Further, using gradient enables optimal resource consumption, hence would mitigate negative impacts on the battery usage

4.3.5 Decision Making

An important step above is the decision making process. To this end, a real-life decision making process via the theoretical design called Prospect Theory has been defined in Sanjab *et al.* (2017). In strongly uncertain scenarios, where the customer demands for a particular content may fluctuate, prospect theory helps to model the customer behavior in a reasonable manner. According to this theory, we have a set of possible prospects which are the outcomes and the probabilities related to these outcomes. In association with our model, these outcomes are the request completion times $T_{u,r,j}$ and each resource point r implies the presence of a prospect. If a user u requests a content item j of popularity class c in resource point $r \in \mathcal{R}$, the number of prospects is computed as the total available resource points. Hence, the prospects can be written as a combination of the following pairs:

$$\mathcal{X}_{u,c,1} = (m(x_{u,c,1,j}), L(v_{\kappa_1,c,j,1})) \tag{4.9}$$

$$\mathcal{X}_{u,c,2} = (m(x_{u,c,2,j}), L(v_{\kappa_1,c,j,2})) \tag{4.10}$$

$$\vdots$$

$$\mathcal{X}_{u,c,\mathcal{R}} = (m(x_{u,c,|\mathcal{R}|,j}), L(v_{\kappa_1,c,j,|\mathcal{R}|})). \tag{4.11}$$

Note that we use κ_1 to establish the probability of having the highest QoS for a specific area for a set of requests; specifically, κ_1 stands for the highest QoS level, while κ_4 stands for the worst QoS.

The quantity $m(x_{u,c,r,j}) = \frac{1}{\delta_{u,c,r,j} v_{\kappa_1,c,j,r}}$ is the probability of attaining the highest QoS on resource computation point r for a content request c. It is easy to observe the intuition behind the ratio of the number of users allocated to a resource requesting a content c experiencing the highest QoS and the total users in accessing the same resource queued for the content c. The $L(\cdot)$ function is the weighting function defined

as in Sanjab *et al.* (2017):

$$L\left(v_{\kappa_1,c,j,r}\right) = \frac{v^{\gamma}_{\kappa_1,c,j,r}}{v^{\gamma}_{\kappa_1,c,j,r} + (1 - v^{\gamma}_{\kappa_1,c,j,r})^{1/\gamma}}. \tag{4.12}$$

Note that prospect theory is based on evaluating a prospect. For our problem setting, the outcomes of evaluation are represented by the completion times of a request, and the service areas (MDC, EDC) are the prospects.

The definition of the value function can be written as $m(x_{u,c,r,j}) =$
$\begin{cases} x^{\omega}_{u,c,r,j}, \text{ if } x_{u,c,r,j} \geq 0 \\ -\zeta(-x^{\theta}_{u,c,r,j}), \text{ if } x_{u,c,r,j} < 0, \end{cases}$ with $0 \leq \omega, \theta \leq 1$ and $\zeta \geq 1$. Thus, the final objective can be written as $\arg\max_r(m(x_{u,c,r,j}), L(v_{\kappa_1,c,j,r}))\ \forall u \in \mathcal{U}$. The next step after the FL framework is executed to define the content popularity $\mathbf{V}$ vector. Each content c that belongs to the vector $\mathbf{V}$ requiring o_j amount of resources where it is executed. Each of the requests arriving at this resource point accepts the content such that it minimizes the cost of the operator. Further, the perceived QoS levels play a key role in mapping the user to the content.

As observed before, each user u that requests a content is allocated to a resource point r. If there are multiple requests arriving at the resource point, we evaluate the time-lines/deadlines of completion. Only the highest QoS levels that are guaranteed for a particular u are accepted and others are rejected. If there are more resources available on the resource r, then we allocate the content on that resource. The choice of the placement that guarantees the highest QoS level is chosen. However, if the QoS levels are the same, we iterate over the value functions $L(\cdot)$ for breaking the ties. After each allocation, the queues are updated for each resource point whether the choice is an EDC or 5G-D2D MDC. The user does not know the exact location of the content as only a set of resource points $\mathcal{R}$ is visible to the user.

4.4 Performance Evaluation

This section first presents the performance evaluation set-up of our experiments and the data-sets generated for carrying out the simulations. We then compare our model with two state-of-the-art models, namely Random Caching by Xu et al. Xu *et al.* (2018) and Edge Boost by Balasubramanian et al. Balasubramanian *et al.* (2019).

4.4.1 Set-up

We use the Python simulator Mininet-Wifi for our simulations. A custom POX controller is used for control plane decision making of the OpenFlow switches (learning and forwarding). We use a pre-processed LSTM model with 4 layers and 128 neurons. We will provide the accuracy analysis for this choice Section 4.4.4. As multimedia caching is an important use-case of 5G, we generate video samples of varying sizes following the DASH standards Y. Li, et al. (2008). We set the base-station user communication range to 500 m. We simulate a total of 250 users, each user has a 5G-D2D communication module for MDC communication with the physical parameters communication range of 150 m and bandwidth of 30 Mbps. Each video segment is divided such that the chunks are 2 s long with a bit-rate of 4000 kbps, resulting in a chunk size of 1 MB. The arrival rate of each video request follows a Poisson distribution over [2, 20]. When the requested content is determined, the corresponding segments will be consequently accessed by the ME. The simulation time is set to 1000 s and 95% confidence intervals are evaluated.

4.4.2 Evaluation

In this section, we compare the proposed FedCo model with *Random Caching* Xu *et al.* (2018) and *Edge-Boost* Balasubramanian *et al.* (2019). Initially, we consider a static (non-mobile) scenario in this subsection to get a basic understanding of the performance, and then bring in mobility in Subsection 4.4.3. In the random caching model, the placement strategy is based on random content selection. In the Edge-Boost model, content placements are deployed with MDCs as the primary resource. In the FedCo model, however, the choice of placement is based on a pool of resources with high computation resource points of the EDC as well as the MDC. In Figure 4.3, the overall revenue is higher while using the FedCo module mainly because of the prediction technique applied on the content. More specifically, Figure 4.4 shows the accrued revenue by FedCo with varying communication rounds. Due to the learning model of FedCo, the requests that are learnt over a period of time enable the MNO to properly place the contents in appropriate locations, thereby producing adequate QoS for the end users. It is very clear that accurate predictions have led to accruing more profit for the MNO, and when the FedCo learning is removed, intuitively profits go down. In Figure 4.5 we can observe the Cache Hit Ratio (CHR), which is defined as the total number of requests generated to the total number of requests satisfied from caches. The caching space of each resource unit ranges from 1%–5%, arbitrarily containing MDC compositions and EDC resource points. We can observe that the CHR is higher for FedCo than for the benchmarks. Further, a larger caching size increases the CHR. However, Edge-Boost and Random Caching lack the prediction of the cached content; therefore, their accuracy is lower by a margin of over 10% compared to FedCo.

The Average Access Latency (AAL) performance is also better for FedCo as shown

in Figure 4.6. AAL is defined as the time interval between the request generated by the user to the first packet received for the requested content. AAL is relatively low for FedCo due to the exact content retrieval which means the extra time required to search the content is saved. Further, the FedCo module empowers the user by allowing exact mapping via the prospect theory value function. The outcome of this AAL evaluation also suggests a better QoS for the users as low access latency means better QoS.

In Figure 4.7, content placement failure is observed. This is the dropping probability of the requests which cannot be completed within the deadline due to placement errors. FedCo has the least percentage error in placement; although, Edge-Boost was expected to perform better in this case because of the stricter assignments with the MDC. The users who are already using the MDC for content retrieval can do so without any service disruption. This outage probability is minimal for the FL controller due to the current knowledge of the state of the devices. Thus, the controller consistently places content between the two main resource blocks from the resource pool $\mathcal{R}$. Figures 4.4 and 4.8 show the need for a large number of communication rounds for having a close to optimal CHR. The CHR becomes around 0.5 after 30 rounds for 100% of users participating while its over 0.3 and 0.4 for 60% and 20% participating users. In Figure 4.4, the revenue relationship with the communication rounds is clear. This variability indicates that higher information exchanged during the learning process shows that the model has a higher effectiveness. The appropriate information learnt is actually establishing the fact that the content has been placed suitably which directly translates into MNO profit. Thus, we see over 40% gain in all cases within the prescribed resource limits of $\mathcal{R}$.

In Figure 4.9, we calculate the time required to train our RNN/D-RNN model. We use an LSTM and consider a case where we have just one FedCo controller that

takes over 70% more time compared to using 30 controllers. The main reason for this reduction with multiple FedCo controllers is that the traffic load is spread across more FedCo agents which perform training on smaller parts of the data.

4.4.3 MDC Management with Mobility

We consider a case of mobility in Figure 4.10 for moving speeds between [1, 5] m/s. We observe from Figure 4.10 that using multiple controllers results in a constant exchange of messages between the controllers. We observe that for lower mobility scenarios there are fewer messages exchanged, mainly because the devices which are registered with one controller have already been passed to the nearby controller even before the movement occurs. Further, we see that the FedCo convergence time changes with the increase in the number of MEs. In Figure 4.11, we observe that as the number of MEs increases, more data is used to train the LSTM model. This means that more training data samples are available and the time taken by FedCo to converge decreases. On the contrary, in state-of-the-art models, such as EdgeBoost and Random Caching, we do not observe this reduction, mainly because the control is distributed among the data-plane devices, which in turn results in a longer time to convergence.

4.4.4 Loss Function Evaluation

Figures 4.12 and 4.13 show how the loss function values, specifically, the values of the general linear regression loss function used for prediction of FL algorithms Chen *et al.* (2021) values vary with the number of MEs and progressing iterations. We note that unlike FedCo, the EdgeBoost and Random Caching benchmarks, do not have prediction capability. In order to enable the loss function comparison, we generalized the loss function for EdgeBoost and Random Caching as follows. The loss func-

tion normally determines the classification errors. However, EdgeBoost and Random Caching models do not conduct classifications, they only select devices based purely on a search mechanism and provide a decision based on this search. This search for devices and placing the content implicitly follows the result of the search, which is essentially a "general" way to place content. Thus, there is going to be a loss associated with the search (depending on the input data and output data); similar to FedCo, where the loss function assesses the correctness of the prediction.

Firstly, in Figure 4.12 we see that as the number of users registering with the FedCo controller increases, the loss function values show a commensurate reduction, with FedCo achieving lower loss function values than the EdgeBoost and Random Caching models. This is mainly attributed to the fact that as the number of generated data samples increases, LSTM can use more data for training itself. This leads to lower training loss. As the number of MEs reaches 15, the training loss reduces faster, which shows that the initial increase in the MEs registering causes most of the reduction in the training loss, with FedCo achieving lower loss function values than the state-of-the-art EdgeBoost and Random Caching models. Similarly, in Figure 4.13, we see that as the iterations continue increasing, the loss function reduces by a substantial margin. This is because, as time progresses, there are enough data samples to approximate the loss function gradient. Another reason for this reduction is that there are more MEs contributing to the global FedCo controller. This enables a more extensive training, resulting in lower training loss. Overall, FedCo performs better, i.e., achieves lower loss function values, mainly because Edgeboost and Random Caching primarily search for devices to optimize the QoS, irrespective of the cost/revenue maximization objective. This comparison shows the balance between meeting user QoS and MNO revenue maximization that FedCo strives to achieve.

Finally, Figure 4.14 shows the prediction accuracy of all three considered con-

trollers. The prediction accuracy is defined as the ratio of the number of correct predictions to the total number of predictions made. We observe that FedCo achieves a better prediction compared to the state-of-the-art models. This shows how a future device is chosen for placing a content in the MDC scenario. We observe a noticeable increase in prediction for FedCo mainly because of its federated averaging algorithm that creates an aggregation of the local MEs' models which is later useful for prediction.

4.5 Conclusion

A Federated Learning (FL) framework named *FedCo* to predict the user demands of a particular edge content has been proposed. FedCo provides suitable content management across different placement sites. In doing so, we show how revenue maximization can be achieved for a Mobile Network Operator (MNO) that enables 5G services, such as 5G-D2D Mobile Device Cloud (MDC) and 5G-Edge Data Center (EDC) access for content caching. The paper also provides theoretical evaluation of user demand behavior modelling via prospect theory to justify how revenue maximization can be achieved by closely investigating user behavior. Finally, we show via performance comparisons how FedCo outperforms two state-of-the-art designs while considering multimedia content delivery use-case. Over 40% profit margin is accrued while the FedCo framework is deployed for 5G mobile edge content caching scenarios.

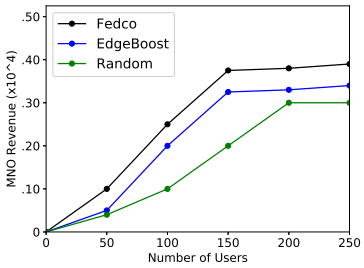


Figure 4.3: MNO Revenue Accrued

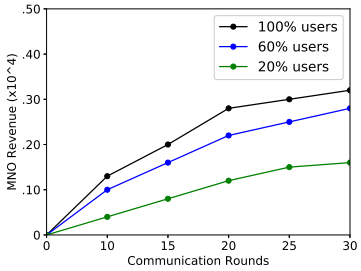


Figure 4.4: FedCo MNO Rev. Diff. Pc. of Fedco Participating Users

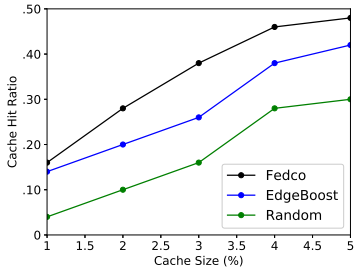


Figure 4.5: Cache Hit Ratio

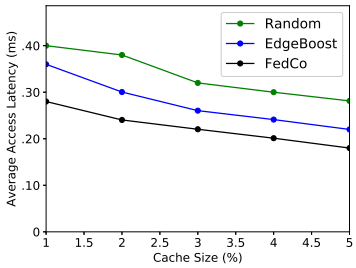


Figure 4.6: Average Access Latency

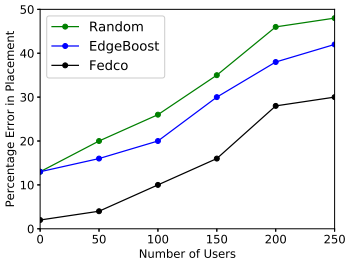


Figure 4.7: Percentage of Error in Placement

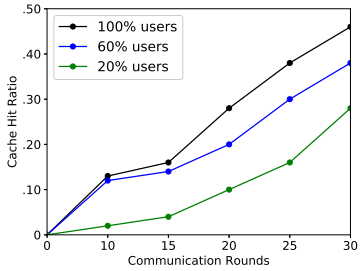


Figure 4.8: FedCo Cache Hit Ratio

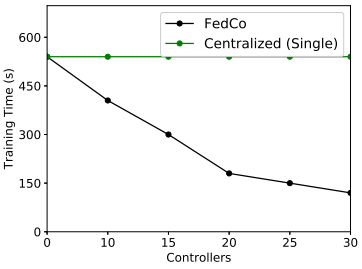


Figure 4.9: Training Time

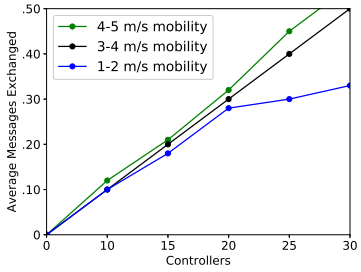


Figure 4.10: Fedco Cnt. Mesg. Inter. for Diff. Mob.Speeds

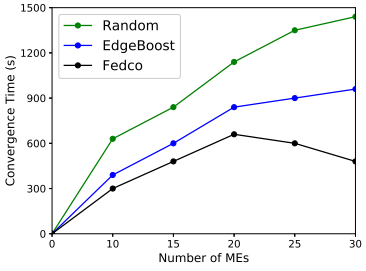


Figure 4.11: More Usr. Reg. w/ the Fedco Cnt, More Samples Trained

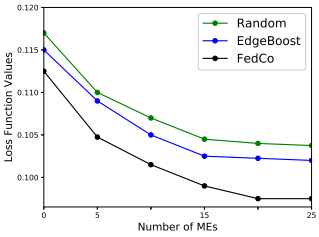


Figure 4.12: as the Number of Samples Increases, Fedco Training Loss Reduces And Remains Unchanged

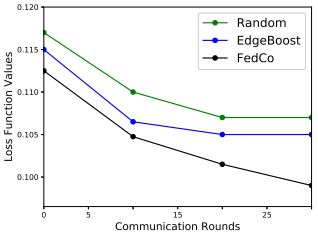


Figure 4.13: Fedco Training Loss Has a Significant Impact with Increase in Communication Rounds

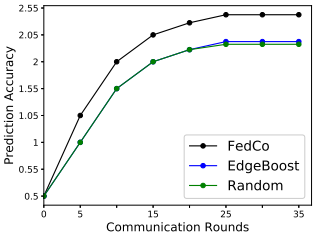


Figure 4.14: Fedco Accuracy Outperforms Traditional Approaches

Chapter 5

CONTROL PLANE FOR VEHICULAR NETWORKS

5.1 Introduction

With the advent of 5G wireless systems, vehicular networks with interactions between the vehicles and the road-side infrastructure at different levels of automation and connectivity have become feasible, enabling the Internet of Vehicles (IoV) paradigm Bréhon-Grataloup *et al.* (2022) Balasubramanian *et al.* (2022); **?**. The data that each vehicle in an IoV produces, i.e., the data that is exchanged over the wireless network between the vehicle and the base stations, or the so-called Road Side Units (RSUs), can be exploited for predicting the vehicular movements on the road network as well as the data traffic on the wireless network. Two key elements of the 5G IoV are vehicle mobility and RSUs connected to a Mobile Network Operator (MNO) X. Kong, et al. (2022). Mobility refers to the movements of a vehicle among the sub-areas of a given wireless cell (zone) or from one zone or RSU to another. MNOs are the infrastructure providers that have the authority over the network resources (such as computation, storage and bandwidth) which are connected to the backbone network. The MNO manages the resource allocation and QoS requirements of the vehicular requests. Each vehicle comprises of a combinations of sensing and communication technologies, which generate datasets that characterize the vehicle behaviors. These raw datasets can be shared with the RSUs once a vehicle establishes a connection with an RSU. In turn, these datasets can be monitored by the MNO. The MNO can then exploit these datasets for spatio-temporal prediction modeling, i.e., for predicting and managing the movements of the vehicles on the roads as well as the data

packet traffic on the wireless network.

A characteristic of common real-life vehicular networks is that there are periodic vehicle movement patterns, e.g., daily (diurnal) commutes to school, university, or work, that can be remembered, i.e., the routes taken to specific locations can be learnt and stored. Adaptive machine learning algorithms can model and represent these patterns to improve vehicular traffic flows and the data packet transport in wireless networks. These learned representations can then enable a wide range of management mechanisms for the vehicular traffic and the wireless network traffic. The focus of the research on edge intelligence in the IoV is currently to advance these representations and the VeNet design seeks to advance this representation research. Prior studies, such as Posner *et al.* (2021); Otoum *et al.* (2020), have addressed prediction problems in vehicular networks, whereas the machine learning frameworks explored in Liang *et al.* (2019, 2020) follow a time-consuming reinforcement learning approach. However, none of these prior studies considered the spatial characteristics of the vehicular data across the sub-areas within a given cell of a cellular wireless network or across the neighboring cells. Our key innovation is to consider the spatial data characteristics across sub-areas and cells to improve the accuracy of the vehicular data representation, and consequently the predictions based on the vehicular data.

Time series data produced by wireless systems and its usage in state-of-the-art models, such as Support Vector Regression (SVR) and Auto Regressive Integrated Moving Average (ARIMA), are very common. These models have shortcomings which limit their use in the IoV. ARIMA drifts naturally to the mean of the past series data which hinders capturing and understanding temporal changes. SVR takes user-defined inputs which need to be pre-calculated. Both of these models use historical data without considering the constantly changing spatial data values that are typical in the IoV.

Another issue with autonomous vehicles in the IoV is the performance degradation due to obstructions in the environment. Vehicles are equipped with sensors, such as Light Detection and Ranging (LIDAR) and visual cameras, but these rely on line-of-sight detection. Therefore, a vehicle does not know what is happening several cars ahead of it for proper planning. Past research has suggested to cooperatively use the vehicles to increase situational awareness Kim *et al.* (2015). However, currently very little is known about the behaviors of such cooperative sensing scenarios in terms of performance scaling. Problems arise from the geometry of the obstructed environments that dictate the coverage area of a sensor. We conduct stochastic geometric modeling for our learning model to account for the line-of-sight obstacle detection.

In this paper, we propose the VeNet framework as groundwork for addressing two key IoV challenges, namely managing the (on-road) vehicle traffic and the wireless (data packet) network traffic. To aid the solution of these management problems, our model predicts the vehicle traffic and the network traffic using the data collected at the MNO. The predicted traffic patterns are useful for managing both the on-road vehicle traffic as well as in-network data packet traffic.

Our VeNet framework utilizes an SDN-based VeNet controller in the base-station for vehicle monitoring and data processing. The VeNet controller is deep learning-based and follows a Multi-Layer Perceptron (MLP) based learning that improves prediction when large datasets are fed as inputs. More specifically, the SDN-VeNet controller includes a hybrid stacked autoencoder (AE) consisting of multiple local AEs for fine-grained spatial data modeling and a central AE, as well as a Long-Short Term Model (LSTM) for temporal prediction. We develop a novel training algorithm for the stacked AE. We conduct measurements with a set of Raspberry Pi vehicles with embedded sensors (including speed, motion, and location sensors) on a tracked studio consisting of six zones (cells) and two RSUs to obtain experimental data.

Our data analysis indicates significant spatio-temporal correlations in the data. The evaluation of the proposed VeNet controller indicates significant improvements in the prediction accuracy of the proposed stacked multiple local AEs–central AE learning model compared with state-of-the-art baseline models.

In summary, the main contributions of this paper are:

1. We design the VeNet learning model consisting of a hybrid stacked AE (multiple local AEs and one central AE) along with a novel training algorithm. To the best of our knowledge, the VeNet learning model, which supports seamless handovers of mobile vehicles between the learning models at the various RSUs, is the first hybrid learning model for vehicular networks.

2. We develop a stochastic geometry model for the detection of obstacles based on the analysis of the vehicle camera-to-obstacle geometry.

3. We extensively evaluate VeNet with a hybrid approach consisting of experimental measurements with Raspberry Pi vehicles on a tracked studio area as well as simulations. The evaluations demonstrate that compared to the ARIMA and SVR models, VeNet improves the prediction of the learning model (errors reduced to approx. three quarters of the ARIMA and SVR errors) for reduced control signalling bitrate as well as reduced energy consumption on the vehicles.

The remainder of this article is structured as follows. Section 5.2 reviews the related state-of-the-art models. Sections 5.3 and 5.4 introduce the VeNet system model and the deep learning based prediction in VeNET. Section 5.5 presents the simulation framework and results. Section 5.6 draws the conclusion and outlines future research directions.

5.2 Related Work

This section reviews the existing state-of-the-art in the two main related areas, namely software-defined IoV and deep learning techniques in the IoV, which have typically been studied separately. To the best of our knowledge, the proposed VeNet approach along with the FedCo approach Balasubramanian *et al.* (2021a) are the first to conduct and implement deep learning (DL) in the context of an IoV based on software-defined networking (SDN).

5.2.1 Software-Controlled Vehicular Networks

Rasouli and Tsotsos Rasouli and Tsotsos (2020) survey the autonomous vehicle-pedestrian interactions to enable the autonomous vehicles to create a prediction-based environment. Communication based on visual perception and communication based on pedestrian (obstacle) behaviors are demonstrated to play major roles in the design of autonomous vehicles. VeNet uses the combination of both these communication modes since obstructions in the vehicle environment require successfully combining the two modes for creating a good prediction model.

Aljeri et al. Aljeri and Boukerche (2021) propose a distributed software-defined vehicular network based on switch-enabled RSUs. Similar to Ethernet back-bone vehicular networks, the discovered nodes and topology information are stored at the RSUs. VeNet does not store topology information to avoid the cost of the topology databases.

Samarakoon et al. Samarakoon *et al.* (2020) study resource allocation in vehicular networks where extreme events affecting queue lengths are observed for ultra low latency communication. Federated learning is employed to estimate the tail distribution of queue lengths, while a Lyapunov optimization model manages the federated

learning delays over wireless links. However, Samarakoon et al. Samarakoon *et al.* (2020) do not consider the spatial or temporal correlations, which enable the low-loss prediction in VeNet. Additionally, VeNet, unlike other approaches, e.g., Du *et al.* (2020); Liu *et al.* (2021), trains with raw data available without any excessive time spent on data labelling.

Wisitpongphan et al. Wisitpongphan *et al.* (2007) examine vehicular adhoc network (VANET) routing. Although, the Samarakoon et al. Samarakoon *et al.* (2020) study is analytically focused, the routing in Samarakoon *et al.* (2020) is similar to the VeNet routing. Specifically, Samarakoon *et al.* (2020) predicts the location where data is sent based on the previous history. Similarly, VeNet maintains a database that collects the data (location, traffic sensed, and images of the traffic) and refreshes itself every few minutes while consistently keeping the vehicle IDs and the zone from which the data was received. However, Samarakoon *et al.* (2020) does not explore fast prediction mechanisms. In contrast, VeNet employs a novel data modelling approach that caters to fast and accurate IoV data representation.

Tong et al. Liu *et al.* (2018) address the problem of optimal social welfare for a vehicular network, which is based on the amount of sensing data received by a central RSU or base station. Network control stability is examined via a backpressure control policy that is distributed among the participating vehicles. A theoretical analysis shows how optimality is reached without providing details about the inter-vehicle locations or mobility based information. However, the creation and maintenance of the inter-base station communication pipes which are integral to the control are not explicitly considered. For VeNet, we find that using and understanding the data that is exchanged between vehicles enables an efficient communication control among the participating vehicles while improving the spatio-temporal data modelling and prediction.

Ning et al. Ning *et al.* (2021) design an online offloading framework that leverages Lyapunov optimization for near-optimal offloading. The offloading is divided into RSU-peer offloading and vehicle-to-RSU offloading and evaluations focus on global optimums. However, in VeNet, we also focus on finding the local optimal values for information exchanges. To that end, VeNet performs additional computations for gathering the neighbourhood information. While Ning *et al.* (2021) does not consider a spatially diverse environment, VeNet extends the framework in Ning *et al.* (2021) by considering spatial diversity.

5.2.2 Deep Learning in Vehicular Networks

The distributed learning framework by Xiao et al. Xiao *et al.* (2022) considers the vehicle velocities and positions that fluctuate with learning time. The study Xiao *et al.* (2022) formulates a min-max optimization problem which optimizes the on-board computation, transmission power, and overall model accuracy. Although, this formulation follows a non-linear programming approach, the sub-problems target the same resource allocation problems as VeNet. While Xiao *et al.* (2022) lacks multi-layered neighbourhood support with additional zonal information exchanges, VeNet exploits these features to achieve improved accuracy and a model representation without the need for heuristic searching.

Generally, timely traffic information is vital for successful prediction frameworks. The prediction of obstructions, e.g., from pedestrians, requires exact local regional topology information. Lv et al. Lv *et al.* (2015) study traffic flow prediction that considers spatio-temporal correlations via a greedy layer-wise AE. Similarly, a stacked AE is a building block of the VeNet traffic representation. However, VeNet uses a hybrid AE consisting of a centralized AE and multiple local AEs. The local AEs obtain information from neighbouring zones and append that location-specific regional

information to the representation of the central AE.

Similar to Lv *et al.* (2015), spatio-temporal dependencies have been modelled by Zhao et al. Zhao *et al.* (2020) with a temporal Graph Convolutional Network (GCN) that combines a gated recurrent unit with GCNs. The GCNs learn the entire topology of a region (urban road network) so as to capture the spatial and temporal dependencies. However, the GCN-based approach Zhao *et al.* (2020) inherits the complexity of graph modelling in convolutional neural networks. In contrast, the layered stacked AE in VeNet reduces the complexity and increases the data-modelling efficiency.

Pokhrel and Choi Pokhrel and Choi (2020) propose a training model that is based on a block chain consensus mechanism. The focus of Pokhrel and Choi (2020) is on a renewal reward, whereby the machine learning model updates each node in a distributed fashion. The considered parameters include the re-transmission limit, block size, and arrival rate. The approach in Pokhrel and Choi (2020) predominantly depends on different block chain mechanisms that impose key bottlenecks. In contrast, VeNet considers these parameters in a block chain agnostic manner that avoids the block chain overheads The seamless communication set-up in VeNet ensures that re-transmissions are rare.

Dongdong et al. Ye *et al.* (2020) present a selection based Deep Neural Network (DNN) model wherein models are selected from a pool of locally trained models in vehicles. A central server collects these results from the vehicles and applies a two-dimensional contract theory for communicating between the vehicular clients and the central server. Similar, to Balasubramanian *et al.* (2021a,b), the central server improves the accuracy with federated averaging. However, in contrast to Ye *et al.* (2020); Balasubramanian *et al.* (2021a,b), VeNet holistically considers inter-zone and intra-zone characteristic features. This holistic approach improves the prediction, see

Sec. 5.5.

Du et al. Du *et al.* (2020) and Li et al. Li *et al.* (2021) discuss a wide range of vehicular IoT (VIoT) designs with deep learning methods. Similarly, Li *et al.* (2021) explores the challenges of vehicular data computation in VIoT scenarios and proposes a novel data-sharing scheme using a deep Q-network with federated learning for trustworthy and efficient data sharing. However, neither of these studies Du *et al.* (2020); Li *et al.* (2021) consider the impact of the training time and performance improvements due to hybrid modelling which are key aspects of VeNet.

Hong et al. Liu *et al.* (2021) propose a security framework that offloads a training model to the edge devices, namely the RSUs and the vehicles. The key idea in Liu *et al.* (2021) revolves around distributed federated learning for preserving privacy in the IoV. However, Liu *et al.* (2021) ignores the need to consider changes in neighboring zones. This could potentially result in privacy and security breaches since different zones in the IoV usually depend on one another when a vehicle moves from one zone to another. The vehicle needs to be aware of changes in the context and environment for a good representation.

Liu et al. Liu *et al.* (2022) employ federated learning to reduce the uplink and downlink communication overheads via three operating modes, namely customized, partial, and flexible. The partial participation mode Liu *et al.* (2022) mitigates the uplink congestion via selective vehicle participation. In contrast, VeNet does not include partial participation in order to comprehensively collect the neighbourhood information. However, VeNet employs simple learning methods in the SDN controller to create isolated uplink and downlink control communication links that do not interfere with the uplink and downlink data transmission channels thus obviating the need for a complex congestion control algorithm.

Chen et al. Chen *et al.* (2021b) use pre-defined road information and pre-processing

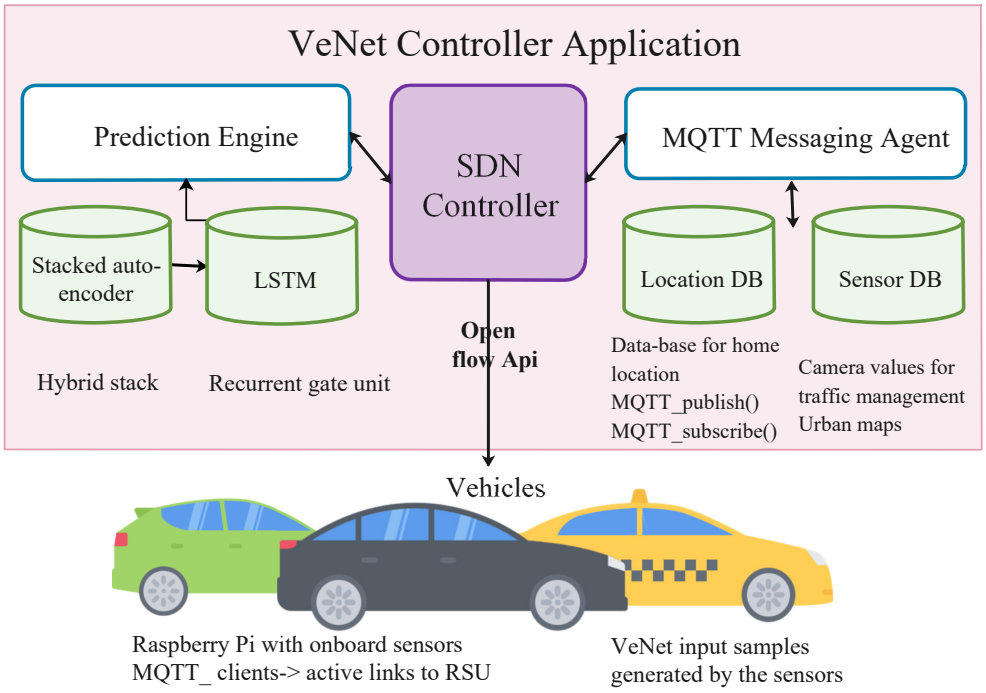


Figure 5.1: Venet Architecture: The Sdn Based Venet Controller in a Road Side Unit (Rsu) Predicts the Vehicle Locations and Network Traffic with A Hybrid Stack (Multiple Local Aes, Central Ae). The Mqtt Messaging Agent Tracks the Vehicle Mobility

that forms data clusters in a learning system employing a single AE. The representations are post-processed to detect and remove anomalies, which is computationally expensive. In contrast, we design a novel hybrid stacked AE consisting of multiple local AEs and a single central AE operating in conjunction with an LSTM.

FedCo Balasubramanian *et al.* (2021a) utilizes distributed federated learning to preserve the privacy of the local information. However, the coordination of the federated learning by a central controller delays the learning process. In contrast, in VeNet local AEs append their learned representation to the central AE representation, to improve the representation accuracy while mitigating delays.

5.3 VeNet System Model

5.3.1 Architecture

The VeNet architecture consists of a VeNet Controller implemented with an SDN contoller, e.g., the Python-based SDN controller (POX), that operates at the RSU, as illustrated in Figure 5.1. The controller maintains the network status status and uses the SDN control channel to communicate with the moving vehicles. The Message Queuing Telemetry Transport (MQTT) Messaging Agent sends pings to the vehicles to maintain the network connectivity and to update the location and sensor databases (DBs). The MQTT Messaging Agent continuously updates the controller about the current vehicle positions. The controller maintains separate uplink and downlink traffic channels to the vehicles for the control signaling and for the payload data to mitigate congestion. The Prediction Engine relies on a stacked AE consisting of several local AEs and a central AE which pass the output to the LSTM model. The LSTM model makes the final prediction and feeds into the controller. The multiple local AEs are running at the RSU. The "responsibility" for a given zone (e.g., cell of cellular communication system) or sub-area within a given zone is split among the different local AEs based on historical spatio-temporal information.

5.3.2 SDN-VeNet Controller Interaction

This sub-section explains how the components in the proposed VeNet framework interact with each other, as further illustrated in Figure 5.1:

1. The data from a given (target) zone or sub-area (depending on the spatial granularity supported by the local AEs) and its neighbours form a *grouped*-data. This includes uplink and downlink network traffic loads and vehicle locations. The AE is stacked hierarchically as a hybrid stack consisting of a central AE

and multiple local AEs.

2. The central AE takes the *grouped*-data as input and produces an encoding. This encoding is passed to the local AEs. Each local AE generates another (local) encoding, which is appended to the global representation.

3. The appended encoding is passed to the LSTM for the final prediction. Together, these modules form the prediction engine. The SDN controller receives the output and uses the MQTT messaging agent to broadcast messages and to receive the status from the vehicles.

4. The vehicle IDs, which are received, are updated. If new samples are received, then the process is repeated. If not, the controller handles all the previously predicted information from the Prediction Engine autonomously.

5.4 VeNet Prediction

There are many models that use historical data from only the target zone for centralized temporal modelling. However, these "target-limited" models do not take the spatial correlations (related to base stations or RSUs near the target) into account which makes them inaccurate for vehicular networks. Thus, we need to design a model that takes the historical data along with the spatially separated cells (or sub-areas) into consideration for making accurate predictions. To that end, our model mainly consists of an LSTM and AEs, which conduct unsupervised learning. We divide the AEs into a central AE and multiple local AEs so as to improve the generalization while gathering data from multiple cells (zones) or multiple sub-areas within a given cell (or zone). The following steps are carried out:

- The uplink/downlink data include multiple features, such as sensor load from the target cell and its neighbours. The central AE is fed with these features and

these features are encoded. After encoding by the central AE, this outcome of encoding is partitioned and fed to the respective local AEs, and appended to the central AE's encoding.

- The information from the central AE and the local AEs is fed to the LSTM for the final prediction. The LSTM updates hidden states based on recurrent gates to control past information which assists in modelling long historical dependencies.

Next, we propose a training algorithm for such a hybrid stacked AE model. To the best of our knowledge, none of the state-of-the-art models makes use of such a hybrid AE model for IoV training; our study is the first to train a hybrid AE model in a vehicular network.

5.4.1 Hybrid Model Training

In order to capture the local spatial correlations, i.e., the localized data characteristics, it is important to use multiple local AEs, each dedicated to a zone (or sub-area of a zone). The central AE keeps the encoding obtained from the overall dataset and trains itself. Further, having multiple local AEs reduces reconstruction loss. Therefore, a trained central AE with multiple local AEs can improve the data representation. Additionally, almost all AEs are operationally placed in stacks, i.e., multiple hidden units. Therefore, having a single stacked AE would lead to a very large dataset size that is hard to train (due to excessive computation requirements). Instead, a decentralized model allows moderate training sizes. Also, the central AE and the local AEs can be trained in parallel, independent of each other. A well-known model, such as Vincent *et al.* (2008), could generally be used for training a centralized model. However, in scenarios where some features are not inherited in the central

model, Vincent *et al.* (2008) performs poorly; therefore, we do not use Vincent *et al.* (2008).

We proceed to define novel encoding functions $E(Z^\ell)$ and decoding functions $D(R^\ell)$ in the local AEs. The encoding function $E(\cdot)$ processes the input layer Z^ℓ, i.e., the layer ℓ out of a total of L layers. Specifically, with the sigmoid activation function $\sigma(\cdot)$, with W_l^ℓ denoting the weight matrix of layer ℓ of the local AE, and with b_l^ℓ representing the bias for layer ℓ of the local AE, the encoding result R^ℓ is

$$R^\ell = E(Z^\ell) = \sigma(W_l^\ell \cdot Z^\ell + b_l^\ell). \tag{5.1}$$

The encoding result R^ℓ is passed to the decoding function $D(\cdot)$. Specifically, for layer $\ell = 1$,

$$D(R^1) = \sigma\left(\hat{\bar{W}}_c^1 \cdot (\hat{W}_c^1 \cdot Z^1 + \hat{b}_c^1) + \hat{\bar{b}}_c^1 + \bar{W}_l^1 \cdot R^1 + \bar{b}_l^1\right), \tag{5.2}$$

while for layers $\ell = 2, \ldots, L$,

$$D(R^\ell) = \sigma\left(\bar{W}_l^\ell + \bar{b}_l^\ell\right). \tag{5.3}$$

Note that the decoding function $D(\cdot)$ establishes the connection between the local AEs and the central AE, whereby $\hat{\bar{W}}_c^1$ and $\hat{\bar{b}}_c^1$ in Eqn. (5.2) represent the trained weights and biases for decoding at the central AE.

The central AE weight matrices for decoding and encoding are transpose to another, i.e., the weight matrix in $D(\cdot)$ is the transpose of the matrix in $E(\cdot)$. We call the data with local information of target cell and their neighbouring (surrounding) cells as $\mathcal{G}(m_{u,v,t})$, i.e., while $m_{u,v,t}$ gives the input data corresponding to the target zone u, v, the $\mathcal{G}(m_{u,v,t})$ gives the aggregated information from zone u, v and its surrounding zones in time slot t.

Algorithm 6 summarizes the main training steps in VeNet, which are presented in an concise form due to space constraints. We refer to the publicly available VeNet

Algorithm 6: VeNet Training Algorithm

Input: Trained central model $\hat{W}_c^\ell$, $\hat{b}_c^\ell$ $\hat{\bar{b}}_c^\ell$; Dataset $\mathcal{M}$

Output: Trained loc. mod. weights, biases $\hat{W}_l^\ell, \hat{b}_l^\ell, \hat{\bar{b}}_l^\ell$

1 **Init** $Z_l^1 = \text{NULL}$;

2 **for** $t \in \text{range}(0, \text{time}) : Z_l^1 = Z_l^1 \bigcup \mathcal{G}(m_{u,v,t})$

3 $\bar{b}_l^1 = 0;\ W_l^1 = 0;$

4 $W_l^1, b_l^1, \bar{b}_l^1 = \arg\min_{W_l^1, b_l^1, \bar{b}_l^1} \rho\left(Z_l^1, D_l^1(E_l^1(Z_l'^1))\right);$

5 $Z_l^2 = E_l^1(Z_1'^l);\ \ell = 2;$

6 **for** layer $\ell \leq L$: $W_l^1, b_l^1, \bar{b}_l^1 = \arg\min_{W_l^\ell, b_l^\ell, \bar{b}_l^\ell} \rho(Z_l^1, D_l^1(E_l^1(Z_l^1)));$

$Z_{l+1} = E_l(Z_l);\ \ell = \ell + 1;$

7 $\hat{W}_l^\ell, \hat{b}_l^\ell, \hat{\bar{b}}_l^\ell = \arg\min_{W_l^\ell, b_l^\ell, \bar{b}_l^\ell} \rho(Z_l'^1, \bar{Z})\ \forall \ell \in 1, 2, \ldots, L$

8 **end for**

9 **return** $\hat{W}_l^\ell, \hat{b}_l^\ell, \hat{\bar{b}}_l^\ell\ \forall \ell \in 1, 2, \ldots, L$

source code for the full algorithmic and implementation details Venkatraman (2021). Typical of an AE, the Z' value is the distorted version of Z. The input data and initialization are specified in Lines 1–3. In particular, we generate the input data for the first layer $\ell = 1$ in the local AE, i.e., we initialize the $\bar{b}_l^1 = 0$ and $W_l^1 = 0$.

Then, we pre-train the first layer $\ell = 1$ with the partial input in Line 4. We use the Stochastic Gradient Descent (SGD) to minimize the reconstruction loss; however, any optimization mechanism, such as RMSProp could be used. The AE reconstruction loss is represented as $\rho(\cdot)$, based on this loss the training is carried out in the first layer. The reconstruction loss is given by Bengio *et al.* (2006) as:

$$\rho(Z, U) = \sum Z \cdot \log(U) + (1 - Z) \cdot \log(1 - U). \tag{5.4}$$

We use the gradient descent approach for optimizing the loss available on tensor flow. Line 5 generates the input Z_l^2 for the next layer $\ell = 2$.

From layer $\ell = 2$ to layer L, the pre-training process is carried out in Line 6. Once this is over, i.e., all layers have been pre-trained. Then the local and central AE values are fine-tuned in Line 7. All weight matrices and bias variables are updated Line 7 and we obtain the reconstructed input. The searches in Lines 4, 6, and 7 are conducted by partitioning the dataset into two subsets and then iterating the search over the subsets (with linear complexity in the cardinality of the dataset $\mathcal{M}$).

Note that the local AEs cannot be trained without the central AE because the decoding values rely on the trained central AE values. As the first layer $\ell = 1$ takes input from the central AE, we will be able to see a difference from the first layer and remaining layers in the local AEs. Layers $\ell = 2$ to L are trained differently based on the output from the first layer. For a given centrally trained AE, all local AEs can be trained in parallel. Importantly, the local and central AEs can have different numbers of layers.

Studies on distributed AEs and stacked AEs typically focus on the mean values of the past time series data, which is not fitting for a highly varying scenario, such as vehicular traffic. Our framework discovers an appropriate representation for the given raw data and stores the representation.

Next, similar to Balasubramanian *et al.* (2021a), we train the LSTM model to represent the time series information. The LSTM model accommodates long-term time dependencies by incorporating recurrent gates that allow current states to update historical values Yu *et al.* (2019).

For prediction of the future location values $m_{u,v,t+1}$ for a given zone (u, v), the past time-slot values are taken as input. For a particular time slot t, the values of $m_{u,v,t}$, central AE representation of $\mathcal{G}(m_{u,v,t})$, and local AE representation of $\mathcal{G}(m_{u,v,t})$ are appended together in a vector. The LSTM unit will predict the value of $m_{u,v,t+1}$, thus forming the complete time sequence of vectors. Given the linear computational

complexity of the underlying AEs and LSTM, the VeNet learning and prediction is overall linear in the cardinality of the dataset $\mathcal{M}$

5.4.2 VeNet Stochastic Geometry Analysis

This section analyzes how collaborative sensing overcomes obstructions in the environment in VeNet. Typically, cameras have a major limitation in terms of visibility in platooning scenarios where one vehicle follows after another. A vehicle typically does not know what is happening two or three cars ahead; this lack of information can be addressed with collaborative sensing. We define obstructing objects on a 2-D plane following a Poisson point process for the object location Loc_i: $\delta = \{Loc_i | Loc_i \in \mathbb{R}^2\}$. With J_i denoting the shape of an obstructing object i, the region C_i that a sensed obstacle occupies is given with the Minkowski sum $\oplus$ as $C_i = \{Loc_i\} \oplus J_i$. The region $C = \bigcup_{i=1}^{\infty} C_i$ is filled with obstructing objects in the environment. We call C' as the open region that is not obstructed by objects. The point Poisson process has an intensity (rate) of λ and each sensor senses an object with probability s_p; thus, the Poisson process for sensed objects grows has the rate $\lambda_s = s_p\lambda$.

A sensor on a vehicle at location Loc_i with a radial (disk shaped) sensing support region S_i can view any unobstructed location in $Loc_i \oplus S_i$. We define the sensing field as independently marked point Poisson process represented by $\bar{\delta}$. We model the sensing field this way mainly because we capture the environment that could potentially comprise of obstructing objects, such as bikes, buildings, and pedestrians. Specifically, we associate each obstructing object with shape J_i to its field as $Q_i = (J_i, S_i)$, resulting in the marked point Poisson process $\bar{\delta} = \{(Loc_i, Q_i), i \in \mathbb{N}^+\}$. Thus, the environment and the sensing fields are modelled by the $\bar{\delta}$ process which associates Q_i, i.e., the J and S values, to each obstruction i. This way we aim to model all the objects in the environment.

In a given environment and sensing field $\bar{\delta}$ and a subset of collaborating sensors, we need to determine the sensing redundancy, which can be accomplished by knowing the sensing field. Thus, based on the sensing field, we define a coverage area set $\Omega_i(\bar{\delta})$ as follows. A sensor can view any location ζ in $Loc_i \oplus S_i^0$, if the location is not obstructed. We consider a sensing disc centered at 0 with a fixed radius, i.e., $S_i^0 = f(0, \text{Radius})$. We denote $l_{0,\zeta}$ for the closed line segment between the center 0 and the location $\zeta \in \mathbb{R}^2$ if there are other objects present that block the line of sight. Formally, $l_{Loc_i,\zeta} \bigcap E^{-i} \subseteq \{\zeta\}$. Thus,

$$\Omega_i(\bar{\delta}) = \{l_{Loc_i,\zeta} \bigcap E^{-i} \subseteq \{\zeta\}\}, \tag{5.5}$$

where $E^{-i} = \bigcup_{k:k\neq i} E_k$ denotes the environment excluding E_i. Throughout, we assume that there is no self-blocking, i.e., that the line of sight channels are not blocked by the sensor at location ζ. Equation (5.5) means that the coverage area set Ω represents the surrounding environment without any obstructions, i.e., the area of the surrounding environment over which the sensor has an unobstructed (direct line of sight) view.

With the superscript 0 denoting the area that we are targeting, the expectation of the non-overlapping region can be evaluated as

$$E[|\Omega|] = E[|S^0 \bigcap J^0|] + E\left[\int_{S^0 \setminus J^0} e^{-\lambda E[\left|l_{0,\zeta} \oplus J'\right|]} d\zeta\right], \tag{5.6}$$

where $J' = \{\zeta | -\zeta \in J\}$. The value of $E[|\Omega|]$ can be evaluated following standard procedures Chiu *et al.* (2013). This formulation also shows how the coverage area of a given sensor decreases with the increase in the density of obstructing objects, i.e., the quantity inside the integral decreases exponentially with increasing λ.

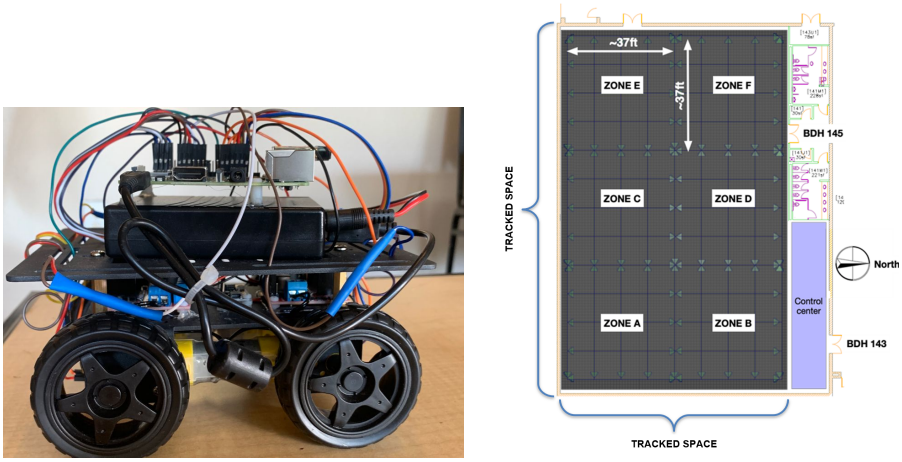


Figure 5.2: Raspberry Pi Vehicle and Asu Drone Studio.

5.5 Evaluation

We consider six 37 x 37 ft (11.2 x 11.2 m) zones in the Arizona State University Drone Studio which includes tracked space and a control center, see Fig. 5.2. Each zone has a 5×5 grid of segmented lanes. We utilized two RSUs to cover all six zones. We use the sigmoid as activation function for each layer in the central AE and the local AEs. We use denoising AEs and follow the corruption process in Vincent *et al.* (2008) with the noise value of 0.1. The central AE has two layers and each local AEs has a single layer.

5.5.1 Data Model

The dataset consists of the data from five Raspberry Pi vehicles, see Fig. 5.2, specifically from the on-board speed-sensors (*LM393*), IMU sensors, passive-GPS markers (based on Opti-Track recommendation), and camera (Raspberry Pi V2), see Venkatraman (2021). The data was transmitted by externally mounted 801.11p modules on the vehicles to two respective software-defined base stations collocated with SDN controllers. Essentially, the base stations can be extrapolated as two RSUs

covering the six cells. We observe the downlink traffic (DT) and the uplink traffic (UT) based on the location coordinates of the mobile vehicles covering the area of six 37×37 ft (11.2×11.2 m) zones. The dataset $\mathcal{M} = \{m_{(u,v,t)}\}$ records the average traffic load in each base station for every one-second time slot for a total of $T = 2$ hours, whereby, $m_{u,v,t}$ includes the set of uplink/downlink traffic samples for the time slots $t \in T$ for zone (u, v). We normalize the data within the interval [0,1] with the tanh estimator method Scheirer *et al.* (2010).

Each vehicle moves for $T = 2$ hours following the lanes in the rectangular lane grid with uniformly random turn choices at the intersections. Unless noted otherwise, the rate λ at which obstacles appear in the environment is set to 0.05 times the number of vehicles. Randomly picked real-life objects, such as a chair or parked vehicles, were uniformly randomly placed as obstacles to replicate a real-world scenario that includes pedestrians and rogue obstacle movements and placements. The evaluation employed a hybrid strategy consisting mainly of measurements with the vehicles in the Drone Studio while the sensor coverage calculations and the sensor redundancy (see Fig. 5.5) were obtained from accompanying discrete event simulations. The 95% confidence intervals of all performance metrics were less than 4% of the corresponding sample means. The confidence intervals are omitted from the the plots to avoid visual clutter.

We first explore how this data is relevant and is representative for typical to real-world datasets. As observed in several studies, e.g., Chinchali *et al.* (2018), the cellular data, i.e., viewing $m_{(u,v,t)}$ as a collection of random process samples, has significant correlations in both the spatial and temporal dimensions. Due to space constraints, we only present a characterization of the spatial correlations in Table 5.3. Table 5.3 clearly indicates non-zero spatial correlations in the dataset $\mathcal{M}$ that are highly dependent on the vehicle location.

5.5.2 Results and Discussion

We compare the prediction errors of two common state-of-the-art training models, namely the Autoregressive Integrated Moving Average (ARIMA) and Support Vector Regression (SVR) models, against the proposed VeNet training model in Table 5.4. The neural network comparison in Table 5.1 indicates that the majority of the related approaches utilize versions of averaging, such as federated averaging, which average weights in gradient descent, and thus follow the same underlying averaging principle as ARIMA. All compared models operate in the SDN based VeNet architecture. Table 5.4 indicates that VeNet achieves substantially smaller prediction errors than the ARIMA and SVR training models; typically, VeNet reduces the MSE and LL down to three quarters or less of the ARIMA and SVR models. Our VeNet model achieves these relatively small prediction errors by considering the spatio-temporal dependencies (correlations) of the sample data.

Now, we elaborate further on the architectural benefits of the multiple local AEs for capturing local information for better representation. As we observe from Figure 5.3, as the number of local AEs (for the 6-zone network) increases, the performance measured in terms of relative (percentage improvement of the prediction accuracy, i.e., reduced prediction error) improves up to 3 local AEs and then flattens out for four and more local AEs. The results clearly indicate that finer-grained learning of the local (spatial) data characteristics enables better prediction. We show the signalling bitrate defined as the average (over time slots) of the bitrate of the signalling (control) traffic per vehicle in Figure 5.4. Importantly, we observe from Fig. 5.4 that the improved VeNet prediction performance from Fig. 5.3 is achieved with a substantially reduced signalling traffic bitrate compared to ARIMA and SVR. The reduced VeNet signalling bitrate is mainly due to the learning support of the local AEs.

We consider sensing redundancy to be the number in percent of vehicles which can view (sense) a given zone location as a function of the number of vehicles in a platoon. Generally, the higher the redundancy, the higher the reliability, i.e., the lower the probability of a collision. We observe from Fig. 5.5 that the sensing redundancy increases with the number of vehicles in a platoon. We clarify that this sensing redundancy is not achieved through direct vehicle-to-vehicle sharing of sensing information. Instead, VeNet achieves the collaborative sensing through the aggregation of the sensing data at the RSU and the processing of the sensed data by the hybrid stacked AE. The lower redundancy of FedCo compared to VeNet in Fig. 5.5 is due to the poor adaptation of FedCo to local changes. FedCo is suitable for small regions with low velocities as the learning considers multiple cost metrics that are averaged in every iteration. However, IoV environments typically extend over several cells and have high mobility. To address these challenges, VeNet performs "mini-batch" learning for every local data set in the local AE model, whereby the central server exchanges the local model if a vehicle moves to another zone.

Figure 5.6 shows the energy consumed by a Raspberry Pi vehicle. We use a USB voltmeter and ampere meter to measure the voltage and current with respect to the amount of data transmitted up to the maximum speed of the Raspberry Pi vehicle of 15 m/s; faster speeds were evaluated with simulations that were configured and validated with the measurements conducted for lower speeds. Figure 5.6(a) considers different values of the transmission parameter α_n Xiao *et al.* (2022), which is inversely proportional to the round-trip communication time between the vehicle and RSU. Fig. 5.6(a) indicates that as the transmission parameter value α_n increases (i.e., the communication round-trip time decreases), the energy consumption is reduced. Also, we observe from Fig. 5.6(a) that with an increase in training data size, the energy consumption increases. Importantly, Figs. 5.6(b) and (c) indicate that VeNet consis-

tently lowers the energy consumption compared to ARIMA and SVR; whereby the energy consumption reduction of VeNet compared to SVR is particularly pronounced. The high SVR energy consumption is due to some computationally demanding learning steps, such as hyper-plane optimization, margin optimum valuations, and kernel function transformations.

Figure 5.7 shows the delay, which is defined as the time taken to execute the learning algorithms, including Alg. 1, with the training data, and to complete the learning process. We observe from Fig. 5.7 that the delay generally increases with the number of vehicles. This is mainly because more vehicles lead to more local characteristics that have to be learnt. We also observe from Fig. 5.7 that VeNet achieves much shorter delays than ARIMA and SVR. This is mainly because ARIMA and SVR do not consider the localized spatial characteristics of the data form the zones. In contrast, VeNet benefits both from considering the localized spatial data characteristics and the central controller that acts as a mediator when a vehicle moves from one zone to another. This ensures that the resources (computation of data) and local model accuracy are not deteriorated as vehicles transition among RSUs. The current local AE learning progress related to a vehicle is passed to the new RSU and added to the corresponding local AE model at the new RSU. This reduces the delay for VeNet compared to the common state-of-the-art ARIMA and SVR models. We note that all of the learning occurs remotely on the SDN-VeNet controller that is collocated with the RSU, and is fed back into the vehicle. FedCo incurs longer delays than VeNet in Figure 5.7 mainly because the federated averaging in FedCo is only performed with a global purview limiting the adaptability to local changes.

5.6 Conclusions and Future Work

We designed the novel VeNet training algorithm that improves the data representation by considering the local (spatial) data characteristics with a stacked hybrid AE consisting of multiple local AEs and one central AE. We evaluated VeNet with experiments with Raspberry Pi based vehicles. We found that VeNet substantially improves the prediction, i.e., reduces the mean square error and log loss compared with the state-of-the-art ARIMA and SVR models (reduction down to three quarters of the ARIMA and SVR errors and losses), while requiring less signalling traffic bitrate. We demonstrated that VeNet achieves effective collaborative sensing and increased the sensing coverage in a platooning scenario. Also, VeNet substantially reduces the energy consumption in the vehicles and reduces the learning delay.

A future research direction is to combine VeNet with 5G based traffic control, e.g., traffic light control Busch *et al.* (2020); Yang *et al.* (2022); Zhang *et al.* (2021), for overall improved Smart City and Industry 4.0 concepts Chen *et al.* (2021a). It would also be interesting to investigate such traffic control approaches in conjunction with the supporting mobile edge computing (MEC) techniques that can facilitate the rapid transfer Doan *et al.* (2021); Liu *et al.* (2016); Doan *et al.* (2022); Thyagaturu *et al.* (2022a) of the control and learning states to MEC nodes close to the vehicle location as vehicles move. A related important future research direction is to examine fault tolerance of the VeNet system and any extensions, whereby fault tolerance to disconnections of the wireless communication links as well as outages of the MEC computing and prediction failures need to be considered in a rigorous manner so as to safeguard the vehicles and their passengers and cargo. We also note that this study excluded privacy concerns about the shared datasets; ensuring adequate data privacy and learning performance of the IoV data representations is an important direction

for future research.

Table 5.1: Summary of Comparison of Venet with State-of-the-art Approaches. ✓ Means Principle Is Employed, X Means Principle Is Not Employed.

Study	SDN	DL	Neural Netw. struct.	Pros	Cons
Rasouli and Tsotsos (2020)	✓	X	X	Software assistance for prediction	Overheads due to excessive signalling
Aljeri and Boukerche (2021)	✓	X	X	Distr. controllers for managing on-road vehicles	Sub-optimal bandwidth allocation
Samarakoon *et al.* (2020)	✓	X	X	Network queue optimization	No on-road support for urban traffic planning
Wisitpongphan *et al.* (2007)	✓	X	X	Message passing between vehicles is optimized	No on-road support for pattern predict. of vehicles
Liu *et al.* (2018)	✓	X	X	Theoretical optimality is observed	No practical application for implementation
Ning *et al.* (2021)	✓	X	X	Offloading assistance of computation	No netw. resource optim. for signal. overheads
Xiao *et al.* (2022)	X	✓	Centr. fed. averaging	Max-Min optim. of computation and storage units in vehicles; no SDN support	Excessive energy consumption of on-board units
Lv *et al.* (2015)	X	✓	One stacked AE	Traf. flow predict. with spatio-temporal correl.	Complicated to implement.
Zhao *et al.* (2020)	X	✓	Centr. GRU/LSTM mod.	Similar to Lv *et al.* (2015)	Inherits the limitations of Lv *et al.* (2015)
Pokhrel and Choi (2020)	X	✓	Centr. fed. block-chain mod. (runs averaging)	Training model is based on a block chain consensus mechanism	Searching all the block-chain components adds complexity and signalling
Ye *et al.* (2020)	X	✓	Centr. server + approach similar to fed. averaging	Applies 2-D contract theory for commun. between vehicular clients and the central server	Long training time and requires optimization models for improving accuracy
Du *et al.* (2020)	X	✓	Centr. server + fed. averaging	Self-driven vehicles use distributed machine learning communication	Long training time and requires optimization models for improving accuracy.
Liu *et al.* (2021)	X	✓	Centr server + local running any CNN	Security framework offloads training model to the edge devices, i.e., RSUs and vehicles	Expensive to implement
Liu *et al.* (2022)	X	✓	Centr. server (fed. avg.) + local customized training	Handles uplink congestion via selective vehicle participation	Requires high signalling to coordinate participation
Chen *et al.* (2021b)	X	✓	Clust. + LSTM + one stacked AE	Uses single AE; post-processing improves accuracy	Heavy pre-processing to cluster data and post-processing
Balasubramanian *et al.* (2021a)	✓	✓	Mult. servers (fed. avg.) + LSTM	Multiple distr. servers to preserve data privacy, with one central coordinating controller at BS for mobility management	Expensive due to distr. controllers, each training own local data and sending updates to centr. controller for global coord., adding to latency
VeNet	✓	✓	Mult. loc. AEs + centr. AE; LSTM	Uplink + downl. ctl. and data transm. channels are isolated and learn the data transm. param.	Long train. time if data size increases in some scenarios, e.g., heavy traffic urban planning

Table 5.2: Summary of Key Notations.

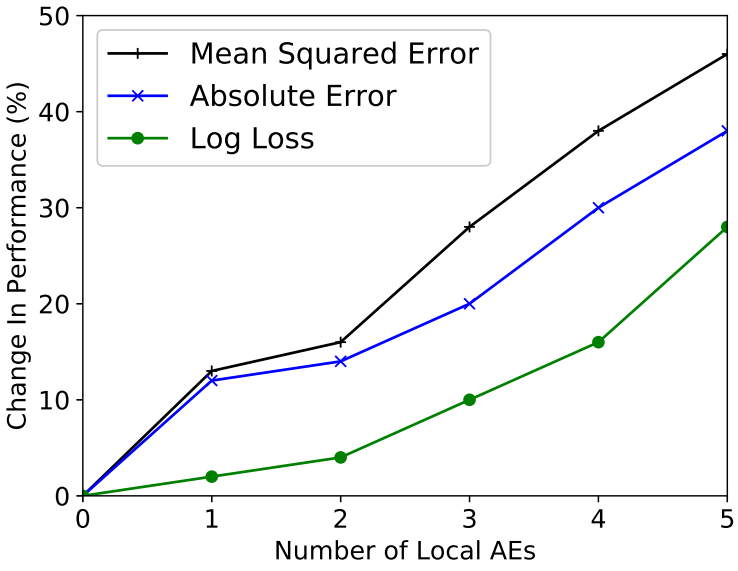
Notat.	Meaning
u, v	Two-dimensional coordinates of a zone
$\hat{m}_{u,v}$	Vector of tanh estimator normalized (to range $[0, 1]$) time series traffic and location data of zone u, v
$\mathcal{M}$	Set of down/uplink traf. loads and sensing data for zones u, v, time slots t, $t \geq 0$
Z^{ℓ}	Input of layer ℓ derived from $m_{u,v}$ data
R^{ℓ}	$= E(Z^{\ell})$ Encoding for layer ℓ
$D(R^{\ell})$	Decoding function
W_l^{ℓ}, W_c^{ℓ}	Weight matrix of local, central encoder, layer ℓ
b_l^{ℓ}, b_c^{ℓ}	Bias for local, central layer ℓ
$\bar{x}$	Decoder weights and biases have bar; Encoder weights and biases have no bar
$\hat{x}$	Trained encoder weights and biases
$\hat{\bar{x}}$	Trained decoder weights and biases
$\rho(\cdot)$	Reconstruction loss
Ω_i	Coverage area of sensor i

Table 5.3: Spatial Autocorrelations in the Dataset M in Between the Six Zones.

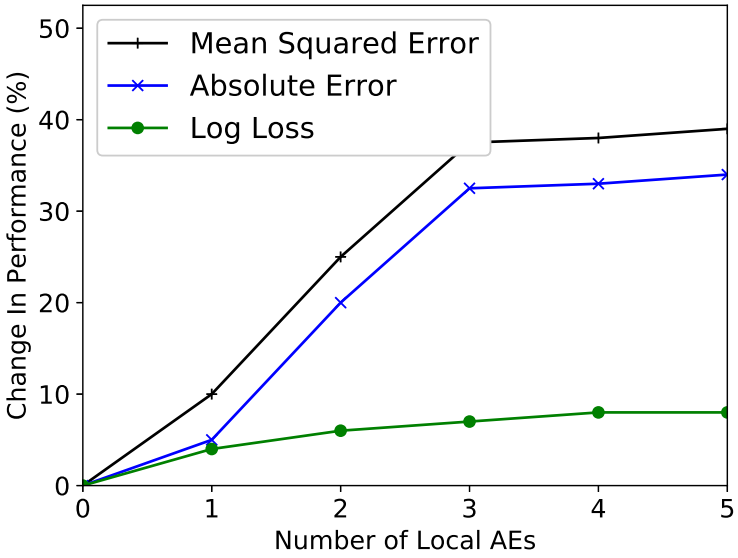
	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6
Zone 1	1	0.162	0.432	0.130	0.040	0.321
Zone 2	0.392	1	0.338	0.129	0.084	0.310
Zone 3	0.340	0.541	1	0.159	0.162	0.690
Zone 4	0.432	0.439	0.458	1	0.104	0.130
Zone 5	0.320	0.471	0.492	0.508	1	0.163
Zone 6	0.282	0.491	0.550	0.431	0.535	1

Table 5.4: Mean Squared Errors (Mse) and Log Loss (Ll) in Zone 2 for Downlink Traffic (Dt) and Uplink Traffic (Ut).

Training Model	MSE DT	MSE UT	LL DT	LL UT
ARIMA	0.045	0.049	0.80	0.86
SVR	0.060	0.065	0.70	0.74
VeNet (4 local AEs)	0.030	0.036	0.52	0.55



(a) Uplink



(b) Downlink

Figure 5.3: Improvement in Venet Uplink and Downlink Prediction Performance As a Fct. Of Number of Local Aes, a Higher Positive "change in Per- Formance" Corresponds to a Smaller Error or Loss; Fixed Number of 5 Vehicles

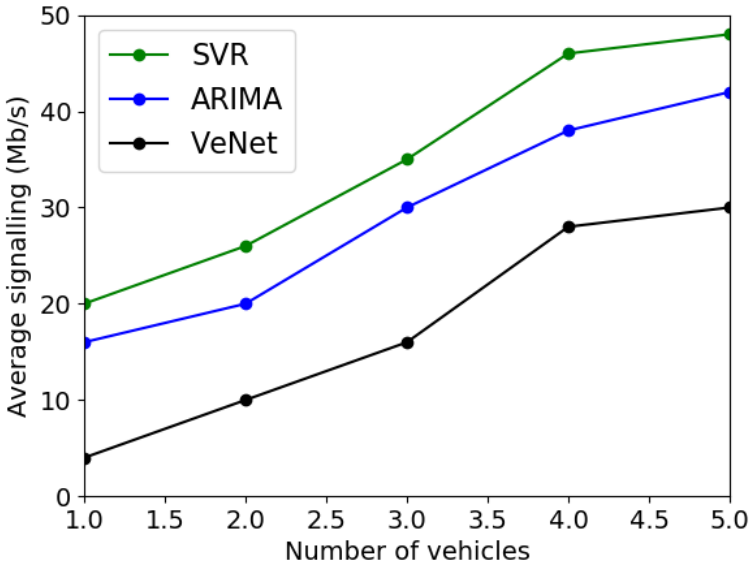


Figure 5.4: Signalling Traffic Bitrate as a Fct. Of Number of Vehicles; fixed transmission param. $\alpha_n = 4$, 4 local AEs in VeNet.

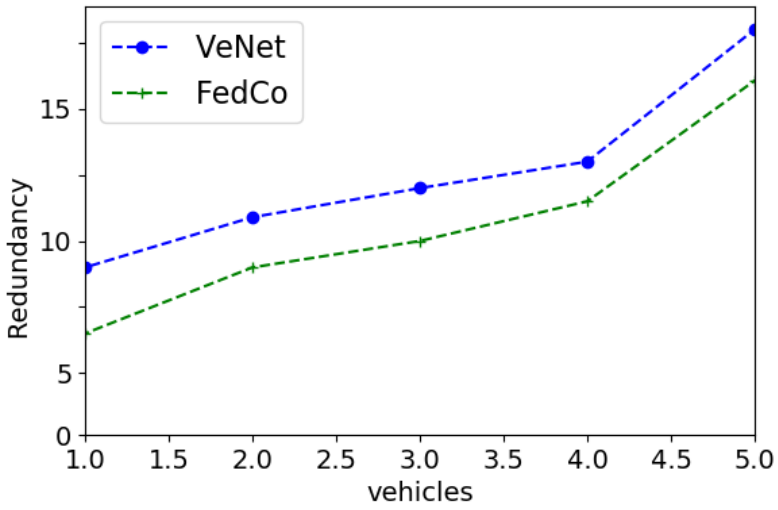
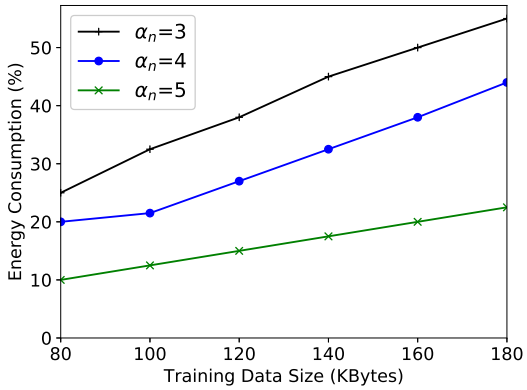
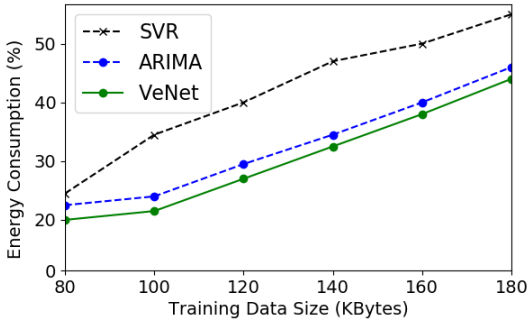


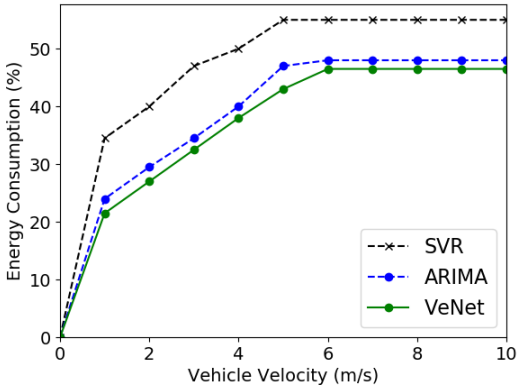
Figure 5.5: Venet Sensing Redundancy (Number of Vehicles in Percent That Can Sense a Location as a Fct. Of the Number of Vehicles in a Platoon; fixed obstacle rate $\lambda = 0.05$, sensing prob. $s_p = 1$, 4 local AEs in VeNet.



(a) VeNet; Train. data size, veloc. = 10 m/s, transm. param. $\alpha_n = 4$,



(b) VeNet vs. ARIMA, SVR; veloc. = 10 m/s, $\alpha_n = 4$



(c) VeNet vs. ARIMA, SVR; $\alpha_n = 4$, Train. Data = 180 KB

Figure 5.6: Energy Consumption on a Raspberry Pi Vehicle; 4 local AEs in VeNet.

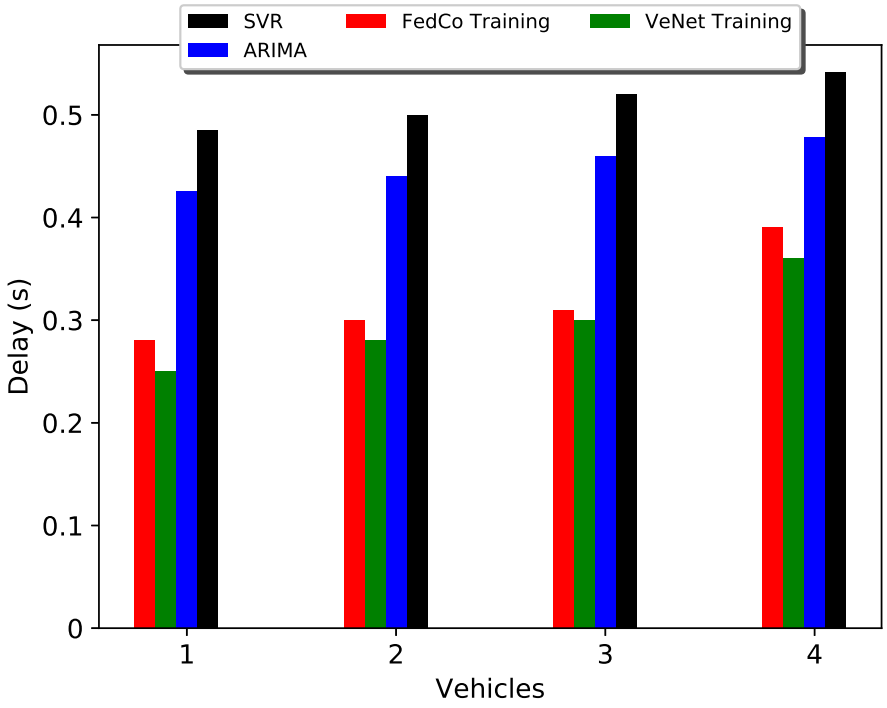


Figure 5.7: Delay in Seconds as a Fct. Of Number of Vehicles.

Chapter 6

CONCLUSION

In this thesis we make a step towards building intelligent network control plane for next generation applications. We observed various network metrics such as latency, bandwidth etc. necessary to ensure a good QoS provisioning in next generation networks. In Chapter 1 we presented a hot take on Time sensitive networks (TSN) and SDN control plane design for such networks. In Chapter 2, we addressed the recovery mechanisms in TSN when failure events arise and explore in depth how SDN control plane can make independent decisions to have back-up paths ready for re-routing traffic.

We discussed FedCo in Chapter 3 that captures a key ingredient of the control plane namely mobility handover. We observe that NGN applications rely heavily on seamless handover at the edge. We show the control plane of this architecture seamlessly offloads data by moving the end-point of the tunnel to the visiting location S-GW with fundamental SDN principles. In Chapter 4, we extend the mobility characteristics to Vehicular Networks and perform deep neural networked learning tasks for vehicular networks. Aptly called as VeNet, this model proposes a new paradigm of Internet of Vehicles architecture for producing a seamless automotive system. We use raspberry pi vehicles to perform some key experiments to show its performance benefits.

In doing so, we have shown performance benefits in each Chapter engendered by the development of intelligent network control plane. The prediction abilities given to network control plane allows network processes such as network resource reconfiguration and fault recovery to handled with very high precision. This makes

it very essential for the next generation network applications.

REFERENCES

Al Ridhawi, I., M. Aloqaily, B. Kantarci, Y. Jararweh and H. Mouftah, "A continuous diversified vehicular cloud service availability framework for smart cities", Computer Networks **145**, 207–218 (2018).

Aljeri, N. and A. Boukerche, "EDiPSo: An efficient scalable topology discovery protocol for software-defined vehicular network", Computer Networks **200**, 108342 (2021).

Atallah, A. A., G. B. Hamad and O. A. Mohamed, "Routing and scheduling of time-triggered traffic in time-sensitive networks", IEEE Transactions on Industrial Informatics **16**, 7, 4525–4534 (2019).

Baccarelli, E., P. G. V. Naranjo, M. Scarpiniti, M. Shojafar and J. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study", IEEE Access **5**, 9882–9910 (2017).

Balasubramanian, V., M. Aloqaily and M. Reisslein, "An SDN architecture for time sensitive industrial IoT", Computer Networks p. Art. no. 107739 (2020).

Balasubramanian, V., M. Aloqaily and M. Reisslein, "FedCo: A federated learning controller for content management in multi-party edge systems", in "Proc. IEEE ICCCN", pp. 1–9 (2021a).

Balasubramanian, V., M. Aloqaily, M. Reisslein and A. Scaglione, "Intelligent resource management at the edge for ubiquitous IoT: An SDN-based federated learning approach", IEEE Network **35**, 5, 114–121 (2021b).

Balasubramanian, V. and A. Karmouch, "An infrastructure as a service for mobile ad-hoc cloud", in "Proc. IEEE Comp. Commun. Workshop and Conf. (CCWC)", pp. 1–7 (2017).

Balasubramanian, V., S. Otoum and M. Reisslein, "VeNet: Hybrid stacked autoencoder learning for cooperative edge intelligence in IoV", IEEE Transactions on Intelligent Transportation Systems (2022).

Balasubramanian, V., M. Wang, M. Reisslein and C. Xu, "Edge-boost: Enhancing multimedia delivery with mobile edge caching in 5G-D2D networks", in "Proc. IEEE ICME", pp. 1684–1689 (2019).

Barakabitze, A. A., A. Ahmad, R. Mijumbi and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges", Computer Networks **167**, 106984 (2020).

Belliardi, R., et al., "Use Cases IEC/IEEE 60802, V1.3", Available from http://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf; Last accessed Nov. 9, 2020 (2018).

Bello, L. L. and W. Steiner, "A perspective on IEEE Time-Sensitive Networking for industrial communication and automation systems", Proc. IEEE **107**, 6, 1094–1120 (2019).

Bengio, Y., P. Lamblin, D. Popovici and H. Larochelle, "Greedy layer-wise training of deep networks", in "Proc. 19th Int. Conf. on Neural Inform. Proc. Systems", p. 153–160 (2006).

Boehm, M., J. Ohms, M. Kumar, O. Gebauer and D. Wermser, "Time-sensitive software-defined networking: A unified control-plane for TSN and SDN", in "Proc. VDE ITG-Symp. Mobile Commun.-Techno. and Appl.", pp. 1–6 (2019).

Boyd, S., S. P. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge university press, 2004).

Bréhon-Grataloup, L., R. Kacimi and A.-L. Beylot, "Mobile edge computing for V2X architectures and applications: A survey", Computer Networks **206**, 108797 (2022).

Brik, B., P. A. Frangoudis and A. Ksentini, "Service-oriented MEC applications placement in a federated edge cloud architecture", in "Proc. IEEE ICC", pp. 1–6 (2020).

Busch, J. V., V. Latzko, M. Reisslein and F. H. Fitzek, "Optimised traffic light management through reinforcement learning: Traffic state agnostic agent vs. holistic agent with current V2I traffic state knowledge", IEEE Open J. of Intelligent Transp. Sys. **1**, 201–216 (2020).

Castillo, E. F., O. M. C. Rendon, A. Ordonez and L. Zambenedetti Granville, "IPro: An approach for intelligent SDN monitoring", Computer Networks **170**, Art. No. 107108 (2020).

Chang, Z., X. Zhou, Z. Wang, H. Li and X. Zhang, "Edge-assisted adaptive video streaming with deep learning in mobile edge networks", in "Proc. IEEE Wireless Comm. Netw. Conf. (WCNC)", pp. 1–6 (2019).

Chen, C., L. Liu, S. Wan, X. Hui and Q. Pei, "Data dissemination for Industry 4.0 applications in Internet of Vehicles based on short-term traffic prediction", ACM TOIT **22**, 1, 1–18 (2021a).

Chen, C., Z. Liu, S. Wan, J. Luan and Q. Pei, "Traffic flow prediction based on deep learning in internet of vehicles", IEEE Transactions on Intelligent Transportation Systems **22**, 6, 3776–3789 (2021b).

Chen, M., Z. Yang, W. Saad, C. Yin, H. V. Poor and S. Cui, "A joint learning and communications framework for federated learning over wireless networks", IEEE Transactions on Wireless Communications **20**, 1, 269–283 (2021).

Chinchali, S., P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone and S. Katti, "Cellular network traffic scheduling with deep reinforcement learning", in "Proc. AAAI Conf. Artif. Intel.", pp. 766–774 (2018).

Chiu, S. N., D. Stoyan, W. S. Kendall and J. Mecke, *Stochastic Geometry and its Applications* (John Wiley & Sons, Hoboken, NJ, 2013).

Craciunas, S. S. and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems", Real-Time Systems **52**, 2, 161–200 (2016).

Deng, S., H. Zhao, W. Fang, J. Yin, S. Dustdar and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence", IEEE Internet of Things Journal **7**, 8, 7457–7469 (2020).

Detti, A., C. Pisa, S. Salsano and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmSDN)", in "Proc. IEEE WiMob", pp. 89–95 (2013).

Doan, T. V., G. Nguyen, M. Reisslein and F. H. P. Fitzek, "SAP: Subchain-aware NFV service placement in mobile edge cloud", IEEE TNSM (2022).

Doan, T. V., G. T. Nguyen, M. Reisslein and F. H. P. Fitzek, "FAST: Flexible and low-latency state transfer in mobile edge computing", IEEE Access **9**, 115315–115334 (2021).

Du, Z., C. Wu, T. Yoshinaga, K.-L. A. Yau, Y. Ji and J. Li, "Federated learning for vehicular internet of things: Recent advances and open issues", IEEE Open J. of the Computer Soc. **1**, 45–61 (2020).

Falk, J., F. Dürr and K. Rothermel, "Time-triggered traffic planning for data networks with conflict graphs", in "Proc. IEEE Real-Time and Embedded Techn. and Appl. Symp. (RTAS)", pp. 124–136 (2020).

Falk, J., D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++", in "Proc. IEEE Int. Conf. on Networked Systems (NetSys)", pp. 1–8 (2019).

Farzaneh, M. H. and A. Knoll, "Time-sensitive networking (TSN): An experimental setup", in "Proc. IEEE Vehicular Net. Conf. (VNC)", pp. 23–26 (2017).

Farzaneh, M. H., S. Shafaei and A. Knoll, "Formally verifiable modeling of in-vehicle time-sensitive networks (TSN) based on logic programming", in "Proc. IEEE Vehicular Net. Conf. (VNC)", pp. 1–4 (2016).

Feng, Z., Q. Deng, M. Cai and J. Li, "Efficient reservation-based fault-tolerant scheduling for ieee 802.1qbv time-sensitive networking", Journal of Systems Architecture **123**, 102381 (2022).

Ferragut, A. and F. Paganini, "Resource allocation over multirate wireless networks: A network utility maximization perspective", Computer Networks **55**, 11, 2658–2674 (2011).

Ferrer, A. J., J. M. Marquès and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing", ACM Comp. Sur. **51**, 6, 1–36 (2019).

Gavriluţ, V. and P. Pop, "Traffic-type assignment for TSN-based mixed-criticality cyber-physical systems", ACM Trans. Cyber-physical Sys. **4**, 2, 1–27 (2020).

Gerhard, T., T. Kobzan, I. Blöcher and M. Hendel, "Software-defined flow reservation: Configuring IEEE 802.1Q time-sensitive networks by the use of software-defined networking", in "Proc. IEEE Int. Conf. on Emerging Techn and Factory Automation (ETFA)", pp. 216–223 (2019).

Guck, J. W., M. Reisslein and W. Kellerer, "Function split between delay-constrained routing and resource allocation for centrally managed QoS in industrial networks", IEEE Transactions on Industrial Informatics **12**, 6, 2050–2061 (2016).

Gutiérrez, M., W. Steiner, R. Dobrin and S. Punnekkat, "Synchronization quality of IEEE 802.1AS in large-scale industrial automation networks", in "Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)", pp. 273–282 (2017).

Habak, K., E. W. Zegura, M. Ammar and K. A. Harras, "Workload management for dynamic mobile device clusters in edge femtoclouds", in "Proc. ACM/IEEE Symp. Edge Comp.", pp. 6:1–6:14 (2017).

Karakoc, N., A. Scaglione, A. Nedic and M. Reisslein, "Multi-layer decomposition of network utility maximization problems", IEEE/ACM Trans. on Netw. **28**, 5, 2077–2091 (2020).

Kehrer, S., O. Kleineberg and D. Heffernan, "A comparison of fault-tolerance concepts for ieee 802.1 time sensitive networks (tsn)", in "Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)", pp. 1–8 (2014).

Kellerer, W., P. Kalmbach, A. Blenk, A. Basta, M. Reisslein and S. Schmid, "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges", Proc. IEEE **107**, 4, 711–731 (2019).

Kim, S.-W., B. Qin, Z. J. Chong, X. Shen, W. Liu, M. H. Ang, E. Frazzoli and D. Rus, "Multivehicle cooperative driving using cooperative perception: Design and experimental validation", IEEE Transactions on Intelligent Transportation Systems **16**, 2, 663–680 (2015).

King, D., A. Farrel, E. N. King, R. Casellas, L. Velasco, R. Nejabati and A. Lord, "The dichotomy of distributed and centralized control: METRO-HAUL, when control planes collide for 5G networks", Optical Switching and Networking **33**, 49–55 (2019).

Krolikowski, J., S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang and X. Geng, "Joint routing and scheduling for large-scale deterministic ip networks", arXiv preprint arXiv:2004.02717 (2020).

Le Boudec, J.-Y., "A theory of traffic regulators for deterministic networks with application to interleaved regulators", IEEE/ACM Transactions on Networking **26**, 6, 2721–2733 (2018).

Leng, X., K. Hou, Y. Chen, K. Bu, L. Song and Y. Li, "A lightweight policy enforcement system for resource protection and management in the SDN-based cloud", Computer Networks **161**, 68–81 (2019).

Li, E., L. Zeng, Z. Zhou and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing", IEEE Trans. Wireless Commun. **19**, 1, 447–457 (2020).

Li, X., L. Cheng, C. Sun, K.-Y. Lam, X. Wang and F. Li, "Federated-learning-empowered collaborative data sharing for vehicular edge networks", IEEE Network **35**, 3, 116–124 (2021).

Li, X., X. Wang, K. Li and V. C. Leung, "CaaS: Caching as a service for 5G networks", IEEE Access **5**, 5982–5993 (2017a).

Li, Y., M. C. Gursoy and S. Velipasalar, "A delay-aware caching algorithm for wireless D2D caching networks", arXiv:1704.01984 (2017b).

Liang, L., H. Ye and G. Y. Li, "Toward intelligent vehicular networks: A machine learning framework", IEEE Internet of Things Journal **6**, 1, 124–135 (2019).

Liang, L., H. Ye, G. Yu and G. Y. Li, "Deep-learning-based wireless resource allocation with application to vehicular networks", Proceedings of the IEEE **108**, 2, 341–356 (2020).

Linguaglossa, L., S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel and G. Bianchi, "Survey of performance acceleration techniques for network function virtualization", Proceedings of the IEEE **107**, 4, 746–764 (2019).

Liu, H., S. Zhang, P. Zhang, X. Zhou, X. Shao, G. Pu and Y. Zhang, "Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing", IEEE Transactions on Vehicular Technology **70**, 6, 6073–6084 (2021).

Liu, L., H. Xu, Z. Niu, P. Wang and D. Han, "U-HAUL: Efficient state migration in NFV", in "Proc. ACM SIGOPS Asia-Pacific Workshop on Systems", pp. 1–8 (2016).

Liu, S., J. Yu, X. Deng and S. Wan, "FedCPF: An efficient-communication federated learning approach for vehicular edge computing in 6G communication networks", IEEE Transactions on Intelligent Transportation Systems **23**, 2, 1616–1629 (2022).

Liu, T., Y. Zhu, R. Jiang and Q. Zhao, "Distributed social welfare maximization in urban vehicular participatory sensing systems", IEEE Transactions on Mobile Computing **17**, 6, 1314–1325 (2018).

Long, C., Y. Cao, T. Jiang and Q. Zhang, "Edge computing framework for cooperative video processing in multimedia IoT systems", IEEE Transactions on Multimedia **20**, 5, 1126–1139 (2018).

Lv, Y., Y. Duan, W. Kang, Z. Li and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach", IEEE Transactions on Intelligent Transportation Systems **16**, 2, 865–873 (2015).

McKeown, N., T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling innovation in campus networks", ACM SIGCOMM Comp. Commun. Rev. **38**, 2, 69–74 (2008).

Mehrabi, M., D. You, V. Latzko, H. Salah, M. Reisslein and F. H. Fitzek, "Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey", IEEE Access **7**, 166079–166108 (2019).

Nasrallah, A., V. Balasubramanian, A. Thyagaturu, M. Reisslein and H. ElBakoury, "Reconfiguration algorithms for high precision communications in time sensitive networks", in "Proc. IEEE Globecom Workshops (GC Wkshps)", pp. 1–6 (2019).

Nasrallah, A., A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein and H. Elbakoury, "Performance comparison of IEEE 802.1 TSN time aware shaper (TAS) and asynchronous traffic shaper (ATS)", IEEE Access **7**, 44165–44181 (2019).

Nasrallah, A., A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein and H. ElBakoury, "Ultra-Low Latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research", IEEE Commun. Surv. & Tut. **21**, 1, 88–145 (2019).

Nayak, N. G., F. Dürr and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks", IEEE Transactions on Industrial Informatics **14**, 5, 2066–2075 (2017).

Neely, M. J., "Stochastic network optimization with application to communication and queueing systems", Synthesis Lectures on Communication Networks **3**, 1, 1–211 (2010).

Niemiec, G. S., L. M. Batista, A. E. Schaeffer-Filho and G. L. Nazar, "A survey on FPGA support for the feasible execution of virtualized network functions", IEEE Commun. Surv. & Tut. **22**, 1, 504–525 (2019).

Ning, Z., K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu and R. Y. K. Kwok, "Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution", IEEE Trans. Intelligent Transportation Systems **22**, 4, 2212–2225 (2021).

Otoum, S., I. Al Ridhawi and H. T. Mouftah, "Blockchain-supported federated learning for trustworthy vehicular networks", in "Proc. IEEE Globecom", pp. 1–6 (2020).

Otoum, S., B. Kantarci and H. Mouftah, "Adaptively supervised and intrusion-aware data aggregation for wireless sensor clusters in critical infrastructures", in "Proc. IEEE Int. Conf. on Commun. (ICC)", pp. 1–6 (2018).

Park, G. S., W. Kim, S. H. Jeong and H. Song, "Smart base station-assisted partial-flow device-to-device offloading system for video streaming services", IEEE Transactions on Mobile Computing **16**, 9, 2639–2655 (2017).

Park, J., S. Samarakoon, M. Bennis and M. Debbah, "Wireless network intelligence at the edge", Proceedings of the IEEE **107**, 11, 2204–2239 (2019).

Peressini, A. L., F. E. Sullivan and J. J. Uhl Jr, *The Mathematics of Nonlinear Programming* (Springer-Verlag, New York, 1988).

Pinheiro, A. J., E. B. Gondim and D. R. Campelo, "An efficient architecture for dynamic middlebox policy enforcement in SDN networks", Computer Networks **122**, 153–162 (2017).

Pokhrel, S. R. and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges", IEEE Transactions on Communications **68**, 8, 4734–4746 (2020).

Posner, J., L. Tseng, M. Aloqaily and Y. Jararweh, "Federated learning in vehicular networks: Opportunities and solutions", IEEE Network **35**, 2, 152–159 (2021).

Raagaard, M. L., P. Pop, M. Gutiérrez and W. Steiner, "Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing", in "Proc. IEEE Fog World Congr. (FWC)", pp. 1–6 (2017).

Ramanan, K., M. I. Reiman *et al.*, "The heavy traffic limit of an unbalanced generalized processor sharing model", The Annals of Applied Probability **18**, 1, 22–58 (2008).

Raptis, T. P., A. Passarella and M. Conti, "Data management in Industry 4.0: State of the art and open challenges", IEEE Access **7**, 97052–97093 (2019).

Rasouli, A. and J. K. Tsotsos, "Autonomous vehicles that interact with pedestrians: A survey of theory and practice", IEEE Transactions on Intelligent Transportation Systems **21**, 3, 900–918 (2020).

Redzovic, H., A. Smiljanic and M. Vesovic, "Implementation and performance comparison of high-capacity software routers", Computer Networks **183**, 107585 (2020).

Ridhawi, I., S. Otoum, M. Aloqaily, Y. Jararweh and T. Baker, "Providing secure and reliable communication for next generation networks in smart cities", Sustainable Cities and Society **56**, 102080.1–102080.14 (2020).

Roughgarden, T., "Algorithmic game theory", Communications of the ACM **53**, 7, 78–86 (2010).

Sacco, A., F. Esposito and G. Marchetto, "A federated learning approach to routing in challenged sdn-enabled edge networks", in "2020 6th IEEE Conference on Network Softwarization (NetSoft)", pp. 150–154 (2020).

Said, S. B. H., Q. H. Truong and M. Boc, "SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN)", ACM SIGBED Review **16**, 1, 27–32 (2019).

Samarakoon, S., M. Bennis, W. Saad and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications", IEEE Trans. Commun. **68**, 2, 1146–1159 (2020).

Sanjab, A., W. Saad and T. Başar, "Prospect theory for enhanced cyber-physical security of drone delivery systems: A network interdiction game", in "Proc. IEEE ICC", pp. 1–6 (2017).

Scheirer, W., A. Rocha, R. Micheals and T. Boult, "Robust fusion: Extreme value theory for recognition score normalization", in "Proc. European Conf. on Computer Vision", pp. 481–495 (2010).

Schweissguth, E., D. Timmermann, H. Parzyjegla, P. Danielis and G. Mühl, "ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks", in "Proc. IEEE Int. Conf. on Embedded and Real-Time Comp. Sys. and Appl. (RTCSA)", pp. 1–11 (2020).

Shantharama, P., A. S. Thyagaturu and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies", IEEE Access **8**, 132021–132085 (2020).

Shantharama et al., "LayBack: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing", IEEE Access **6**, 57545–57561 (2018).

Shen, L., X. Yang and B. Ramamurthy, "Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks", IEEE/ACM Transactions on Networking **13**, 4, 918–931 (2005).

Silva, L., P. Pedreiras, P. Fonseca and L. Almeida, "On the adequacy of SDN and TSN for industry 4.0", in "Proc. IEEE Int. Symp. on Real-Time Distributed Computing", pp. 43–51 (2019).

Stanton, K. B., "Distributing deterministic, accurate time for tightly coordinated network and software applications: IEEE 802.1 AS, the TSN profile of PTP", IEEE Communications Standards Magazine **2**, 2, 34–40 (2018).

Steiner, W., "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks", in "Proc. 31st IEEE Real-Time Systems Symp.", pp. 375–384 (2010).

Tapolcai, J., L. Rónyai, B. Vass and L. Gyimóthi, "List of shared risk link groups representing regional failures with limited size", in "Proc. IEEE INFOCOM", pp. 1–9 (2017).

Thyagaturu, A. S., G. Nguyen, B. P. Rimal and M. Reisslein, "Ubi-Flex-Cloud: Ubiquitous flexible cloud computing: status quo and research imperatives", Applied Computing and Informatics , ahead-of-print (2022a).

Thyagaturu, A. S., P. Shantharama, A. Nasrallah and M. Reisslein, "Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies", IEEE Access (2022b).

Venkatraman, "Venet: Vehicular iot", in "https://github.com/Venkatraman9214/VehicularIoT", (2021).

Vincent, P., H. Larochelle, Y. Bengio and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders", in "Proc. 25th Int. Conf. on Machine Learning", pp. 1096–1103 (2008).

Vu, T. T., D. T. Ngo, N. H. Tran, H. Q. Ngo, M. N. Dao and R. H. Middleton, "Cell-free massive MIMO for wireless federated learning", IEEE Trans. Wirel. Commun. **19**, 10, 6377–6392 (2020).

Wang, G., Y. Zhao, J. Huang and Y. Wu, "An effective approach to controller placement in software defined wide area networks", IEEE TNSM **15**, 1, 344–355 (2018).

Wang, S., T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning", in "Proc. IEEE INFOCOM", pp. 63–71 (2018).

Wang, S., T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He and K. Chan, "Adaptive federated learning in resource constrained edge computing systems", IEEE Journal on Selected Areas in Communications **37**, 6, 1205–1221 (2019).

Wang, X., Y. Han, V. C. M. Leung, D. Niyato, X. Yan and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey", IEEE Commun. Surv. & Tut. **22**, 2, 869–904 (2020).

Wang, Y., X. Wang, H. Li, Y. Dong, Q. Liu and X. Shi, "A multi-service differentiation traffic management strategy in SDN cloud data center", Computer Networks **171**, 107143 (2020).

Wisitpongphan, N., F. Bai, P. Mudalige, V. Sadekar and O. Tonguz, "Routing in sparse vehicular ad hoc wireless networks", IEEE J. Sel. Areas in Commun. **25**, 8, 1538–1556 (2007).

Wolsey, L. A. and G. L. Nemhauser, *Integer and Combinatorial Optimization*, vol. 55 (John Wiley & Sons, 1999).

Wu, D., L. Zhou, Y. Cai and Y. Qian, "Collaborative caching and matching for D2D content sharing", IEEE Wirel. Commun. **25**, 3, 43–49 (2018).

X. Kong, et al., "Deep reinforcement learning based energy efficient edge computing for internet of vehicles", IEEE Trans. on Industrial Informatics, in print pp. 1–1 (2022).

Xiang, Z., F. Gabriel, E. Urbano, G. T. Nguyen, M. Reisslein and F. H. P. Fitzek, "Reducing latency in virtual machines: Enabling tactile internet for human-machine co-working", IEEE Journal on Selected Areas in Communications **37**, 5, 1098–1116 (2019).

Xiang, Z., M. Howeler, D. You, M. Reisslein and F. H. P. Fitzek, "X-MAN: A non-intrusive power manager for energy-adaptive cloud-native network functions", IEEE Transactions on Network and Service Management **19**, 2, 1017–1035 (2022).

Xiao, H., J. Zhao, Q. Pei, J. Feng, L. Liu and W. Shi, "Vehicle selection and resource optimization for federated learning in vehicular edge computing", IEEE Trans. Int. Transp. Sys., in print pp. 1–15 (2022).

Xu, C., M. Wang, X. Chen, L. Zhong and L. A. Grieco, "Optimal information centric caching in 5G device-to-device communications", IEEE Trans. Mob. Comp. **17**, 9, 2114–2126 (2018).

Y. Li, et al., "Content-aware playout and packet scheduling for video streaming over wireless links", IEEE Trans. Multimedia **10**, 5, 885–895 (2008).

Yang, K., T. Jiang, Y. Shi and Z. Ding, "Federated learning via over-the-air computation", IEEE Transactions on Wireless Communications **19**, 3, 2022–2035 (2020).

Yang, X., Y. Xu, L. Kuang, Z. Wang, H. Gao and X. Wang, "An information fusion approach to intelligent traffic signal control using the joint methods of multiagent reinforcement learning and artificial intelligence of things", IEEE T-ITS, in print pp. 1–11 (2022).

Yang, Z. and K. L. Yeung, "Flow monitoring scheme design in SDN", Computer Networks **167**, 107007 (2020).

Ye, D., R. Yu, M. Pan and Z. Han, "Federated learning in vehicular edge computing: A selective model aggregation approach", IEEE Access **8**, 23920–23935 (2020).

Yu, Q. and M. Gu, "Adaptive group routing and scheduling in multicast time-sensitive networks", IEEE Access **8**, 37855–37865 (2020).

Yu, Y., X. Si, C. Hu and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures", Neural Computation **31**, 7, 1235–1270 (2019).

Yu, Z., J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang and N. Georgalas, "Federated learning based proactive content caching in edge computing", in "Proc. IEEE GLOBECOMx", pp. 1–6 (2018).

Zhang, J., L. Chen, T. Wang and X. Wang, "Analysis of TSN for industrial automation based on network calculus", in "Proc. IEEE Int. Con. on Emerging Technologies and Factory Automation", pp. 240–247 (2019).

Zhang, K., S. Leng, Y. He, S. Maharjan and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing", IEEE Wireless Commun. **25**, 3, 80–87 (2018).

Zhang, R., A. Ishikawa, W. Wang, B. Striner and O. K. Tonguz, "Using reinforcement learning with partial vehicle detection for intelligent traffic signal control", IEEE Transactions on Intelligent Transportation Systems **22**, 1, 404–415 (2021).

Zhao, L., P. Pop and S. S. Craciunas, "Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus", IEEE Access **6**, 41803–41815 (2018).

Zhao, L., Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction", IEEE Trans. Intelligent Transp. Sys. **21**, 9, 3848–3858 (2020).

Zhao, Z., C. Feng, H. H. Yang and X. Luo, "Federated-learning-enabled intelligent fog radio access networks: Fundamental theory, key techniques, and future trends", IEEE Wireless Communications **27**, 2, 22–28 (2020).

Álvarez, I., D. Čavka, J. Proenza and M. Barranco, "Simulation of the proactive transmission of replicated frames mechanism over TSN", in "Proc. IEEE ETFA", pp. 1375–1378 (2019).