Architectures and Algorithms for Transport-layer Multi-connectivity in Next Generation Wireless Networks

Author:

Kariem Fahmi

Supervisor:

Prof. Douglas Leith

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Doctor of Philosophy

University of Dublin, Trinity College

March 2021

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

I, the undersigned, agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent / do not consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Kariem Fahmi

June 12, 2021

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Kariem Fahmi

June 12, 2021

Acknowledgments

I was very fortunate during my PhD study to have been guided and supported by many great people to whom I owe great debt of gratitude.

First and foremost, I would like to thank my supervisor, Prof. Douglas Leith, for his invaluable guidance, brilliant insights, and for being an impeccable scientific role-model. He has made this 4-year journey a truly transformative experience.

I would also like to thank my advisors in Bell Labs Ireland, Dr. Stepan Kucera and Dr. Holger Claussen, for our many fruitful discussions and for allowing the work in this thesis to be consistently informed by the important problems faced in industry.

Finally, I would like to thank my family and friends, specially my dear friend Andreea, for always being there for me and helping me get through those many tough times. Most of all I would like to thank my dear mom for her continuous encouragement and relentless support in every way humanly possible.

Kariem Fahmi

University of Dublin, Trinity College
March 2021

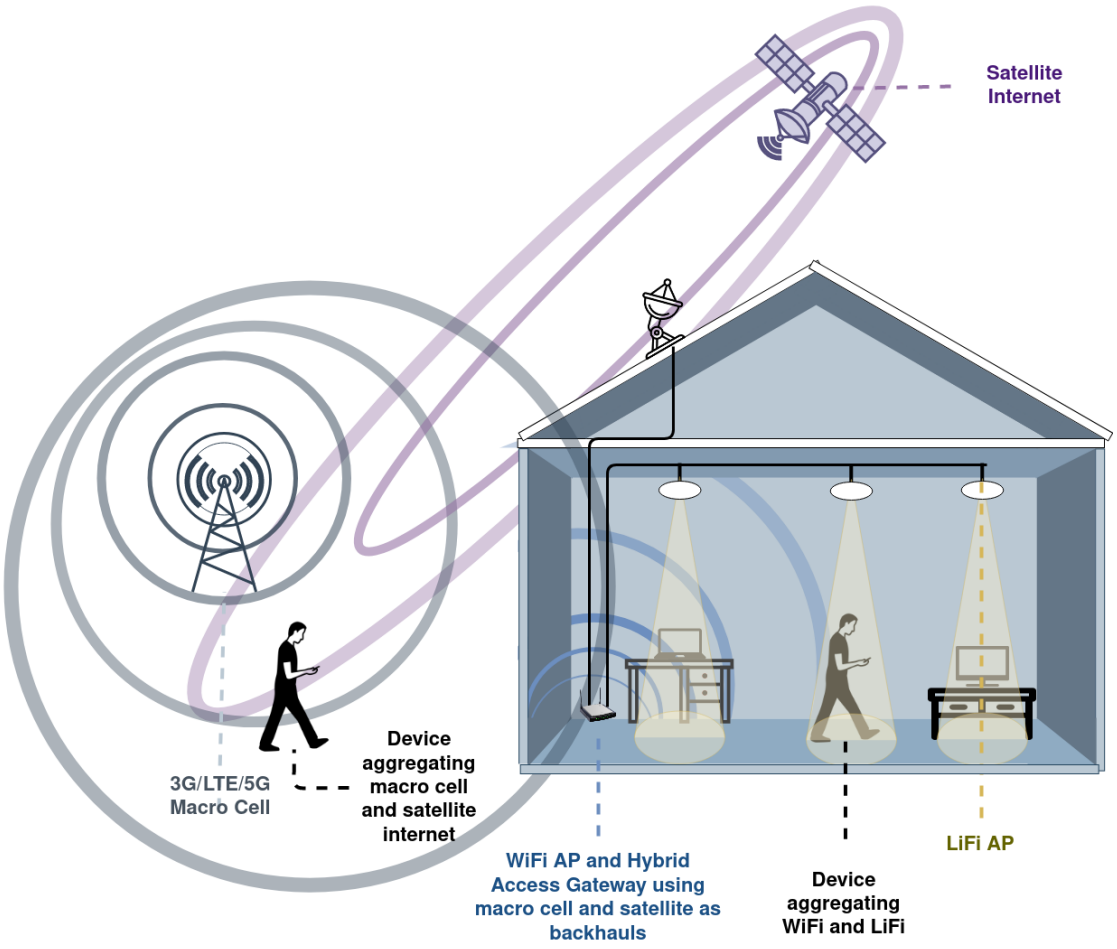
Contents

List of Tables

List of Figures

Chapter 1

Introduction



(a) Wireless multi-connectivity outdoors and indoors

Figure 1.1: Figure shows multi-connectivity being used outdoors to aggregate a macrocell and satellite internet , and indoors to aggregate the same two wireless links as backhauls by a hybrid access router and disseminate their capacity over WiFi and LiFi to a user with a multi-connectivity capable smart device

.

1.1 Motivation For multi-connectivity (MC)

The emerging interest in next generation network applications such as virtual/augmented reality and ultra-high definition video services for inter-personal communications (virtual reality telepresence, 360 video conferenc-ing), inter-vehicular communications (multi-camera multi-angle "X-ray" vision), augmented vision (graphical and data overlay in manufacturing, maintenance, logistics environments) and machine control (remote surgery, drone control) along with the rapid deployment of billions of Internet of Things (IoT) devices have created unprecedented demand for low latency, high bandwidth and high reliability wireless connectivity.

Next generation high-speed radio access technologies using mm-wave carriers like 5G New Radio[99], new light-based wireless communication technology (LiFi)[27], readily accessible satellite Internet [91], along with the evolving LTE [81] and Wi-Fi [100] networks will significantly enhance the ability of wireless communication to support this new connected reality. They will also create a heterogeneous wireless environment where devices need to draw on different types of wireless access technologies simultaneously – by using multi-connectivity (MC) – if they are to achieve the best possible quality of service (QoS).

MC is the technique of connecting two hosts through more than one network, often over different physical links. While MC can aggregate multiple wired networks to improve capacity, latency and reliability, it is significantly more useful in the context of wireless connectivity. Wireless links routinely, but unpredictably, degrade for multiple reasons: (i) physical layer - time-varying capacity due to multi-path propagation and shadowing, poor or no connectivity at the coverage edge, cell selection and handovers, (ii) medium access layer - CSMA/CA-based contention in IEEE 802.11 networks, multi-user multiplexing, shared/private queuing, (iii) transport layer - periodical data losses followed by protracted end-to-end re-transmissions, self-inflicted excessive queuing delays of up to units of seconds due to blind capacity probing of Transmission Control Protocol (TCP). However, aggregating multiple wireless links can significantly reduce or even eliminate these problems while maintaining the freedom provided by wireless connectivity. MC reduces the probability of outages as multiple independent links are unlikely to undergo handovers simultaneously, increases average capacity to the combined capacity of all links and improves latency through transmission of redundant packets either using simple replication or in the form of Forward Error Correction (FEC) to minimize latency/losses.

Fig 1.1 shows an example of wireless MC being used outdoors and indoors. In the outdoors environment, the smart device, with MC capability, is aggregating 3G/LTE/5G

from a macro cell together with satellite internet. In the indoors setup, the hybrid access router is using MC to obtain connectivity through the satellite internet, which it is getting through a roof-mounted receiver, and the macro cell. It is then disseminating their capacity via WiFi and LiFi to the user.

1.2 Application areas of multi-connectivity

MC has already started to be used in a number of real world applications, ranging from personal use on smart devices to factory automation. We highlight three application areas below.

Personal

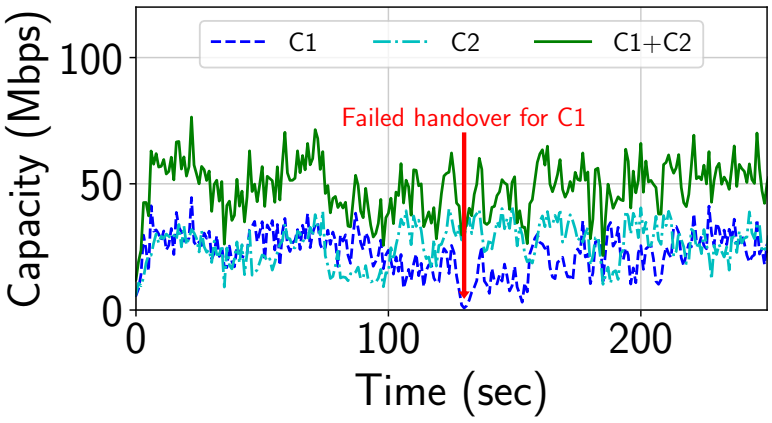
In the area of personal smart devices, including smartphones and tablets, the availability of both an LTE and a WiFi interface on most devices, combined with the ubiquity of WiFi hotspots and LTE coverage, creates an environment where most users can readily take advantage of MC. Two notable commercial entities have leveraged this fact. First, the smartphone manufacturer Apple deployed MPTCP [39], a transport-layer multi-connectivity (TLMC) protocol, in their smart phone operating system iOS and their laptop/desktop operating system MacOS to improve the reliability of their flagship services, including Siri, Apple Maps and Apple Music. Second, in the operator space, the mobile network operator (MNO) Korea Telecom (KT) also used MPTCP [59] to enable MPTCP-capable phones (i.e. phones running a kernel with MPTCP) on their mobile network to aggregate KT LTE together with WiFi from their KT WiFi hotspots transparently, to reach gigabit capacity, in what they referred to as gigabit LTE.

Mobile network operators (MNO)

MNO have utilized MC in two key areas: hybrid access for home connectivity and WiFi offload. Hybrid access combines DSL and 3G/LTE/5G in a controlled way to deliver fibre-like experience while minimizing expensive mobile network usage, particularly targeting rural areas without fibre coverage. For example, the company Tessares [62] provides this service to multiple ISPs around the world using a transparent MPTCP proxy placed inside customer premise equipment (CPE) that is equipped with both a DSL backhaul and a cellular backhaul. The proxy distributes the traffic across both links with the aim of achieving advertised DSL rates for all customers regardless of the capacity of the DSL connection. MNOs also utilize MC in the form of WiFi-offloading to relieve pressure off

the mobile network (MN). In the past, WiFi-offloading was limited to having a device seamlessly authenticate, using SIM credentials, and offload traffic to an MNO controlled WiFi network instead of the mobile network whenever possible. However, the introduction of new 3GPP standards will enable the MNO to perform more sophisticated functions such as load-balancing and splitting traffic. In particular, the Access Traffic Steering, Switching & Splitting (ATSSS) [89] feature in 5G release 16 allows the MNO to utilize a MC technology, such as MPTCP, to stripe traffic across the mobile network and their own WiFi networks whenever possible. This evolves WiFi offloading into full-blown MC and allows it to be used not only for cost reduction but also for QoS fulfillment. According to Cisco [9], WiFi offload taffic, that is traffic that was seamlessly offloaded to an MNO controlled WiFi network, account for 59% of 5G traffic by 2022, allowing most of the MNO traffic to benefit from MC.

While in the past, WiFi-offloading was limited to having a device switch to a WiFi network instead of the MN whenever possible, more recently with the introduction of the 3GPP hotspot 2.0 standard, which allows customers to login to operator-controlled WiFi hotspots with their SIM credentials, and the introduction of the Access Traffic Steering, Switching & Splitting (ATSSS) [89] in 5G release 16 , which evolves WiFi offload function from simply switching to more sophisticated traffic splitting and load-balancing using MPTCP, WiFi offloading will evolve into full-blown MC and will be used not only for cost-reduction but also for QoS fulfillment. According to Cisco [9], WiFi offload will account for 71% of 5G traffic by 2023, allowing most of MN traffic to benefit from MC.



(a) Link capacity over time

Figure 1.2: Figure shows the individual and combined capacity of two LTE links from two different carriers, C1 and C2, mounted on a train traveling from Germany to Austria at speeds between 100-150 km/h. It also highlights a failed handover on link C1, which led to an outage.

Vehicle to ground

The benefits of MC for wireless links are amplified under high mobility, which has led to multiple commercial entities offering wireless MC solutions aimed at moving vehicles , including trains, buses, large ships, UAV/drones and various kinds of robots [80][29][85][69][1][78]. The main motivation for this is that wireless connections degrade under high mobility due to fast fading and frequent handovers. Fast fading is caused by the large Doppler spread during high mobility and results in channel estimation errors which increase loss and reduce rate. Additionally, the high mobility causes the link to experience frequent handovers which result in brief, but frequently recurring, outages and loss of buffered data inside the old base station. Meanwhile, remotely controlled vehicles, such UAV/drones, require reliable latency-sensitive communication to receive control signals and transmit real-time video – both necessary for remote control – and transportation vehicles, such as trains, require both high capacity communication to provide on-board internet for large numbers of passengers and reliable latency-sensitive communication for mission-critical traffic such as closed-circuit TV (CCTV) and communication-based train control (CBTC). MC allows vehicle communication to leverage spectral diversity (e.g. different cellular carriers) and spatial diversity (e.g. placing wireless clients at the head and tail of a train) increasing the average capacity and improving reliability. A real world example from our own measurements: Fig 1.2 shows the shows the individual and combined capacity of two LTE links from two different carriers mounted on a train traveling from Germany to Austria at speeds between 100-150 km/h. It can be seen that MC would have prevented a complete outage at t=120 seconds by allowing C2 to compensate for C1 experiencing a failed handover.

1.3 Multi-connectivity technology state of the art: MPTCP

Currently, the most widely adopted MC technologies is MPTCP. It is an IETF standardized extension [19] of the TCP protocol that enables it to communicate over multiple links. It is a fairly mature protocol that has been in development since 2012. MPTCP is implemented inside the operating system kernel and has versions for Linux [63], FreeBSD [61], Apple iOS and MacOS [39]. Notable commercial entities such as Korea Telecom (KT) and Apple have adopted MPTCP in their products, and it has been the subject of over 600 research papers [79].

Transport-Layer Multi-Connectivity (TLMC) through MPTCP has a number of key

advantages:

- Backward compatibility with TCP - Applications that use TCP are not required to change their code to be able to use MPTCP. By extending an existing transport layer protocol, MPTCP is transparently used by legacy TCP applications, as if it was an updated version of TCP. Additionally, MPTCP will fall back to TCP in the event that the remote-host does not support it.

- Transparency to middle boxes - MPTCP disguises itself as TCP to middle boxes by replicating the header format, while using extra TCP options to carry its own signaling. This allows to avert interference from the majority of middle boxes, although some will still remove any unfamiliar TCP options, forcing it to fall back to TCP.

- Technology independence and scalability - Transport-layer solutions, like MPTCP, are technology agnostic that are generally independent from the needs and evolution of the underlying technology standards (eg, as defined by the 3GPP and the IEEE).

- Accurate end-to-end performance control - 90% of Internet traffic uses TCP [33] and thus relies on the transport-layer for rate control and reliability. Consequently, the transport-layer maintains indicators of end-to-end performance such as the delay and capacity. Control of these mechanisms and information on these indicators is a critical pre-requisite for a MC solution to maximize throughput and minimize latency. MPTCP, by virtue of existing at the transport-layer, fulfills this pre-requisite. However, this is not the case for other layers of the network stack and replicating these functionalities at other layers would risk cross-layer interference and would involve high complexity. An analogical example is the problem of so-called TCP-meltdown [11] that occurs when stacking multiple layers of reliability such as when tunneling TCP over TCP.

1.4 Contributions and outline of the thesis

In this thesis, we make four contributions to TLMC

- First, we address the challenges in deploying and using MPTCP as a higher-layer steering function in the 5G Access Steering Splitting Switching (ATSSS) function [89]. While the design choices of MPTCP have given it broad support for existing applications and made great progress in creating a deployable MC solution, they did

not address all the obstacles towards adoption. MPTCP's implementation inside the kernel limits its ability to be deployed in devices with proprietary kernels, such as many smartphones and laptops; the OEM must take steps towards integrating MPTCP into their kernel. Even after MPTCP is integrated into a device, updates are slow and customizations are impossible since the OEM controls the kernel. One consequence of this limitation is that it prevents MNO from being able to use MC to flexibly manage the wireless resources available to their users in order to better satisfy QoS requirements. Thus, we propose a novel new user-space approach to deploying TLMC that is more readily deployable and extensible compared to a kernel implementation, requires no changes to the user device, and is backward compatible with all existing applications and device. The proposal is based on leveraging the built-in VPN framework available on all major mobile operating systems to deploy new transport-layer behavior, such as TLMC, on the path between user devices and a network-edge proxy. As a proof of concept, we create an Android implementation to evaluate its efficacy, efficiency and flexibility. We repeatedly leverage this framework in the later chapters of the thesis to deploy and evaluate our TLMC solutions.

- Second, we address the problem of scheduling packet transmissions amongst multiple wireless paths with uncertain, time-varying delay. The MPTCP multi-path scheduler, minRtt, employs a strategy that attempts to minimize per-packet delay, and assumes links are homogeneous and stable. However, wireless links have time-varying delay and capacity, and thus are not stable nor homogeneous. We also make the observation that the requirement for multi-path scheduling is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the per packet delay which is important. This has fundamental implications for multipath scheduler design. We introduce SOS (Stochastic Object-aware Scheduler), the first multipath scheduler that considers application layer object sizes and their relationship to link uncertainty. We demonstrate that SOS reduces the 95% percentile object delivery delay by 50-100% over production WiFi and LTE links compared to state-of-the art schedulers. We extend SOS to utilize FEC and to show that it is possible to achieve significant improvement in the 95% percentile object delay without sacrificing mean object delay using minimal redundancy. Finally, we show how to interface SOS with cwnd-based/ack-clocked congestion control (as usually used in TCP) and show up to 150% improvement in the 95% percentile and mean object delay vs MPTCP/minRTT.

- Third, we analyze the challenges faced by MPTCP when used to aggregate multiple WAN/Internet connections in Multi-WAN Routers (MWR). We observe that the two architectural variants, proxying and tunneling, used to deploy MPTCP in MWR suffer from key performance problems. First, the proxy variant creates one MPTCP connection for each TCP connection, which results in a large number of parallel uncoordinated MPTCP connections, which we explain leads to underutilization of the available capacity, suboptimal multi-path scheduling, and increased loss rate. Second, the tunnel variant, which relies on encapsulating TCP over MPTCP, stacks two reliability layers and leads to a large number of spurious retransmissions, an issue known as TCP meltdown. Instead, we propose a new multi-path solution more suited to MWR, called BOOST. This solution eliminates the problems with both the proxy and tunnel approaches by multiplexing TCP connections over a single persistent multi-path connection. BOOST also takes a novel approach to multi-path scheduling that combines multi-path load balancing and scheduling. In particular, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity. Evaluations show that BOOST provides better throughput, lower losses, and retransmissions while requiring less memory compared to both MPTCP variants

- Fourth, we consider the task of transporting mission-critical Train to Ground (T2G) traffic, such as Closed-Circuit TV (CCTV) and Communication-Based Train Control (CBTC), which have strict QoS requirements for high availability and low packet loss, in underground trains that are equipped with multiple WiFi backhauls. MPTCP is a reliable protocol, and thus it is not always well suited for real-time traffic. However, we review the state of the art in this application area, and we observe that the simple replication approach taken in other studies comes at the cost of many duplicate packet transmissions and so wasted network capacity. Instead we take the approach of predicting when a handover is about to occur and only replicating packets shortly before it happens, significantly reducing the amount of redundancy needed and freeing up network resources for more productive use. We propose a novel neural network-based predictor, based only on features extracted from the periodic RSSI scans, that predicts when a WiFi client, with a proprietary and closed-source handover algorithm, will trigger a handover. We train and evaluate our predictor on a large data set collected in a production environment, and

show it predicts handover with high accuracy. We use this predictor as a basis for an adaptive replication scheme and combine it with a multi-path load balancer that load-balances mission-critical traffic flows across available links in a round-robin fashion. We compare three versions of this scheme: no replication (predictor always returns false), smart replication (predictor works normally), and replication (predictor always returns true). Evaluations show that smart replication significantly reduces losses, compared to no replication, while using a small fraction of the redundancy, compared to the replication scheme.

1.4.1 Outline of the thesis

- In Chapter 2, we present a brief overview of key concepts in MPTCP and MC that are relevant to the problems addressed in later chapters and our proposed solutions.

- In Chapter 3, *Deploying Next Generation Transport Protocols in Smart Device Using the VPN Framework*, we propose a new way of deploying TLMC in user-space in an over-the-top way that allows it be immediately deployed to smart devices through a simple app download and gives it much more flexibility compared to a kernel implementation.

- In Chapter 4, *Low Delay Scheduling of Objects Over Multiple Wireless Paths*, we address the problem of optimally scheduling application-layer objects over links with time-varying delay

- In Chapter 5, *BOOST: Transport-Layer Multi-Connectivity Solution for Multi-WAN Routers*, we outline the problems involved in utilizing MPTCP as a TLMC solution in Multi-WAN Routers (MWR) that handle traffic from a large number of users simultaneously, and propose a new TLMC solution, called BOOST, that is more suitable for MWR.

- In Chapter 6, *Smart replication based on handover prediction for seamless and efficient real time communication in underground railway train*, efficiently mitigating delay caused handovers for real-time services in underground T2G communications

- In Chapter 7, we provide a conclusion and future work.

1.5 Publications

1.5.1 Research Papers

Published

- **Fahmi, K.**, Leith, D. J., Kucera, S., & Claussen, H. (2020). SOS: Stochastic Object-aware Scheduler for low delay communication over multiple wireless paths. ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 1–6.

- Kucera, S., **Fahmi, K.**, & Claussen, H. (2019). Latency As a Service: Enabling Reliable Data Delivery over Multiple Unreliable Wireless Links. 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), 1–5.

- Basaras, P., Kucera, S., **Fahmi, K.**, Claussen, H., & Iosifidis, G. (2020). Demo: Seamless Mobile Video Streaming in Multicast Multi-RAT Communications. IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 1280–1281.

Submitted

- **Fahmi, K.** & Leith, D. J. (2021). BOOST: Multi-Path Backhaul Solution for Train-To-Ground Communication. *Submitted to the 2021 International Federation for Information Processing (IFIP) Networking Conference.*

- **Fahmi, K.** & Leith, D. J. (2021). Smart replication based on handover prediction for seamless and efficient real time communication in underground trains. *Submitted to the 2021 International Conference on Computer Communications and Networks (ICCCN)*

- **Fahmi, K.** & Leith, D. J. (2021). Low Delay Scheduling of Objects Over Multiple Wireless Paths *Submitted to the Elsevier Journal of Computer Communications (COMCOM).*

1.5.2 Patents

- Fahmi, K., & Kucera, S. (2020). Optimising Traffic Flows (Patent No. EP3588895A1).

- Fahmi, K., & Kucera, S. (2018). A Method of Distributing Load and a Network Apparatus (Patent No. EP3364630A1).

- Fahmi, K., & Kucera, S. (2018). A Method of Distributing a Sub-Flow Associated with a Session and a Network Apparatus (Patent No. EP3364624A1).
- Fahmi, K., & Kucera, S. (2018). Multiple Path Transmission of Data (Patent No. EP3297322A1).
- Fahmi, K., & Kucera, S. (2018). Network Node Communication (Patent No. EP3355653A1).
- Buddhikot, M., Fahmi, K., & Kucera, S. (2019). Multi-Path Data Communications (Patent No. WO2019063868A1).
- Fahmi, K., Kucera, S., Segel, J., & Sridhar, K. (2018). Methods and Network Elements for Multi-Connectivity Control (Patent No. WO2018156681A1).

Chapter 2

Background

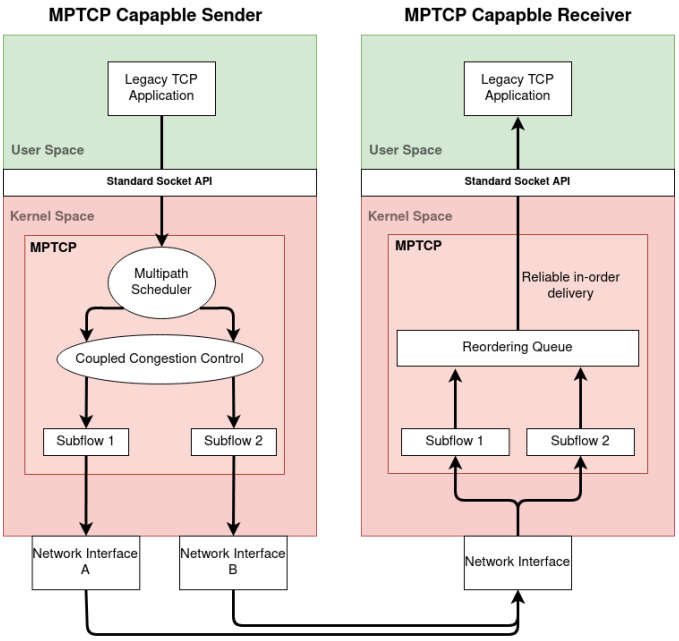
In Section 2.1, we provide background on MPTCP design and key mechanisms that are referenced in later chapters. In Section 2.2 we describe the proxy architecture commonly used to deploy MC protocols. Background that is specific to a particular contribution is presented in the respective chapter of the thesis.

2.1 MPTCP

In this section, we describe specific MPTCP mechanisms to help illuminate the problems and solutions discussed in later chapters. We limit the discussion only to mechanisms relevant to this thesis and skip ones that are outside the scope.

2.1.1 Overview

MPTCP is a kernel space implementation of a MC transport-layer protocol [60]. It is backward compatible with TCP, meaning that existing TCP applications can use it. As shown in Fig. 2.1, the legacy TCP application uses the standard socket API to communicate with the remote host, but inside the kernel the TCP protocol is substituted with MPTCP. The three main components of MPTCP are the multi-path scheduler, the coupled congestion control (CCC) and the subflows [79]. Data from the application is distributed amongst the available subflows by the scheduler and the congestion window (CWND) of each subflow is decided using the CCC. The congestion window represents the amount of data that ca be in-flight at once on a subflow. In order to minimize interference from middle-boxes, MPTCP subflows appear as if they are independent TCP flows.



(a) MPTCP overview

Figure 2.1: Figure shows overview of overview of the MPTCP software architecture illustrating relevant component in both sender and receiver

.

2.1.2 Connection Establishment

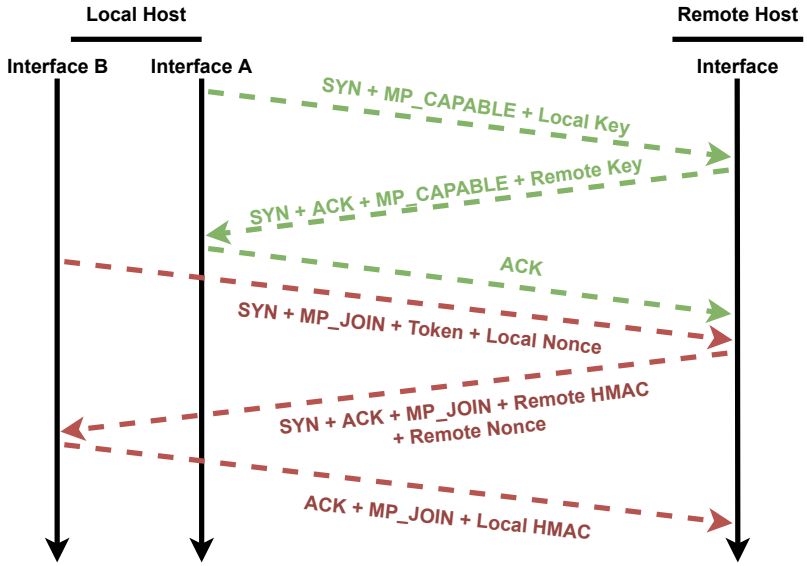
MPTCP starts a new connection by performing the same three-way handshake as TCP to establish its initial subflow. A subflow in MPTCP refers to a specific path between the local host and the remote host where data can be exchanged. The initial subflow is always established over the default route configured in the operating system's routing table. During the handshake, MPTCP will transmit two key values as TCP options in the SYN packet: the MP_CAPABLE option and a random key – the local key. The MP_CAPABLE option serves to identify whether the other side of the connection supports MPTCP. If it does, it will respond with the MP_CAPABLE option in the SYN-ACK packet and it will also attach its own key – the remote key. If the remote host doesn't support MPTCP, the connection will fall back to standard TCP.

2.1.3 Subflow Establishment

Once the initial subflow is established and both sides support MPTCP, the local host will attempt to establish secondary subflows with the remote host by performing 3-way TCP handshakes over other available routes. For each available network interface, MPTCP will use that interface to send a SYN packet to the remote host IP address with the MP_JOIN MPTCP option, a hashed token generated from the two keys received during the inital handshake, and a random value – the local nonce. If the remote host is reachable and

it receives the SYN-JOIN packet, it will use the attached token to identify the original connection and map the subflow to it. It will then respond with a SYN-ACK packet with the option MP_JOIN-ACK, a random value – the remote nonce – and the remote Hash-based Message Authentication Code (HMAC). The remote HMAC is created using the two original keys and local nonce. The purpose is to authenticate the subflow. Finally, the local host will respond with an ACK with the MP_JOIN option and a local HMAC.

Fig 2.2 shows the exchange of messages during the MPTCP handshake and subflow addition.



(a)

Figure 2.2: Exchange of messages during the MPTCP handshake and subflow addition

.

2.1.4 Multi-path Scheduling

Once at least two subflows are established, MPTCP will start splitting application traffic across available subflows, which is referred to as multi-path scheduling. The objective of a multi-path scheduler is to ensure that data arrives in-order and with the minimum delay possible. In protocols that guarantee ordered delivery, such as TCP and MPTCP, data received out of order will be held in a re-ordering buffer until it can be delivered in order, increasing the packet's delay. This effect is referred to as head of line (HoL) blocking. While HoL blocking is not unique to MC – it can be induced by packet loss in single path communication – it is much more common in MC due to multi-path scheduling. Besides increasing delay, HoL blocking will reduce the receive window announced by the receiver to the sender due to buffer space being taken up by blocked packets. The receive window represents the amount of free buffer space for receiving data in an MPTCP receiver. A

low receive window may potentially reduce the sending rate below link capacity, wasting available capacity. If the receive window is reduced to zero as a result of HoL blocking, MPTCP will re-inject the packets that are causing the HoL blocking on other links and penalize the slow link that caused the block by halving its CWND. This strategy is referred to as penalization and re-injection.

The default MPTCP scheduler is MinRTT. It works by adding data to the link with the lowest smoothened round trip time (SRTT) until it has completely filled its congestion window (CWND) before moving on to the the link with next lowest SRTT and repeating the process.

2.1.5 Congestion Control

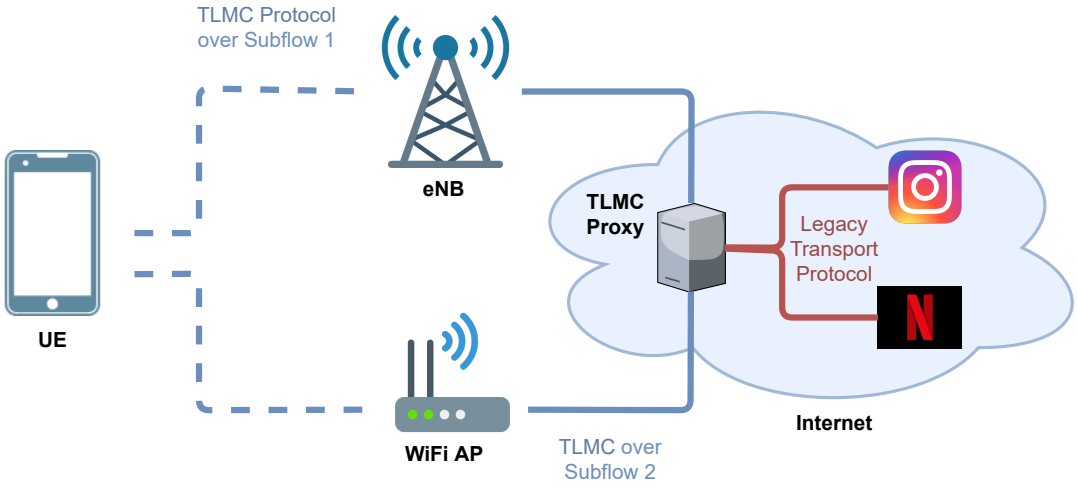
MPTCP uses what is referred to as coupled congestion control (CCC). In CCC, a single CC process manages the send rate for all subflows. The main motivation behind CCC is fairness. If two subflows in one MPTCP connection share a bottleneck, then having independent CC will result in the MPTCP connection having twice the share of the bottleneck as a single TCP connection, which is unfair. CCC algorithms attempt to infer if the bottleneck is shared and decide the rate accordingly. However, MPTCP can also be configured to have independent CC processes for each flow, such as RENO or CUBIC.

2.1.6 Data Sequencing

MPTCP utilizes both a connection-level sequence number and ack number, referred to as Data Sequence Signal (DSS) and Data Acknowledgment (DACK) respectively, and a subflow-level sequence number and ack number, which are the standard TCP sequence (SEQ) and Acknowledgment (ACK) numbers. The DSS is injected into each packet using MPTCP DSS option. The DSS is necessary to make sure data received on all subflows can be delivered to the application in-order. Subflows maintain their own independent SEQ and ACK states, just like a normal TCP connection. They use these states for acknowledging data, in order to advance the subflow's send window, and re-transmissions for lost data. However, data sent on one subflow may be re-injected in a different subflow, with an appropriate SEQ number, to resolve HoL blocking through the penalization and re-injection mechanism described previously.

2.2 Proxy-based MC network architecture

Proxy-based communication is a common approach of deploying TLMC protocols, like MPTCP, due to the lack of support from content servers. Transport protocols are end-to-end, and thus support for the protocol is needed on both sides of a connection. However, while a user can modify his own device to add support for a MC protocol, he cannot add support in the content server. One solution to this is to deploy a proxy located close to the network edge (e.g. within a cloudlet) in order to implement new transport-layer behavior, such as TLMC protocols, over the path between the proxy and the client, which will span the wireless hop. Standard legacy protocols are then used to communicate between the proxy and the end servers. Fig 2.3 shows an example of this network architecture.



(a)

Figure 2.3: Proxy-based MC Network Architecture

.

Chapter 3

Deploying next generation transport protocols in smart device using the VPN framework

3.1 Introduction

As discussed briefly in the introduction, 5G release 16 has introduced the Access Traffic Steering, Switching & Splitting (ATSSS) [89] feature which allows an MNO to create rules for managing simultaneous user access to 3GPP (e.g. 5G) and non-3GPP (e.g. WiFi) wireless access technologies that are connected to the same core, termed 5G converged core. Using ATSSS, an operator can provision policies to prioritize one access technology to reduce cost, load balance users to avoid congestion, boost a user's throughput through aggregation or map specific services to different access technologies for improved QoS in key applications.

ATSSS is only a policy engine and needs an underlying technology, termed "steering function" by the 3GPP [89], to enforce the actual policy. Steering functions can either be lower-layer steering functions, called the ATSSS functions, or higher-layer steering functions, such as MPTCP [89]. Fig 3.1 shows the architecture for an MPTCP-based ATSSS deployment. Compared to the lower-layer steering functionality, MPTCP has a key advantage in terms of performance due to being a transport layer MC solution. In particular, MPTCP is able to dynamically split traffic across links as their capacity changes. Conversely, the 3GPP lower layer splitting functions simply splits traffic across access technologies according to some pre-defined ratio communicated by the 5G core policy control function (PCF) regardless of the actual link capacity. Furthermore, performing

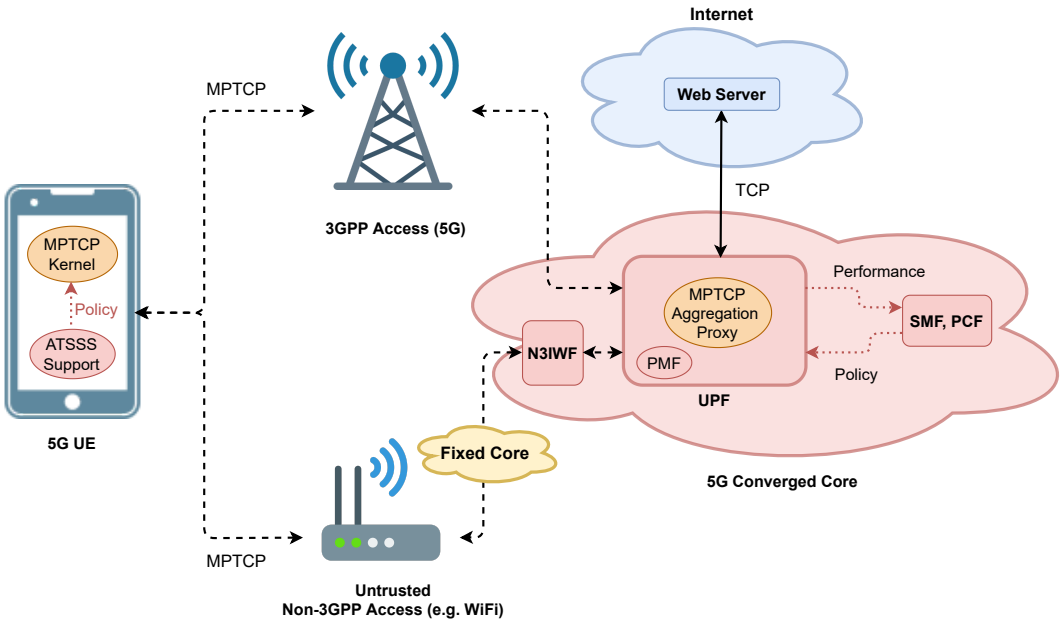


Figure 3.1: Placement options for client-side and network-side proxies.

traffic splitting on lower layers when transporting TCP traffic can cause link capacity to appear highly fluctuating to the TCP congestion control, potentially causing it to reduce its sending rate. MPTCP does not have this problem as it is a replacement for TCP.

For ATSSS to use the MPTCP steering function, the user device needs to be running an MPTCP kernel, which currently still does not have widespread adoption and is not yet merged with either the upstream Androind kernel or Linux kernel. While Apple successfully integrated MPTCP into their iOS and MacOS operating systems, this is much harder to do in the highly fragmented Android operating system, which is currently used by 85% of smart phones [36]. For MPTCP to gain mainstream adoption in Android devices, first it will need to be merged with the upstream Linux and Android kernels, and then hundreds of original equipment manufacturers (OEM) would need to take steps to integrate it into their proprietary kernels. However, in addition to this process being extremely slow, currently OEMs keep devices running on the kernel they were designed for, even after they've been updated to newer Android versions, because those kernels contain all the appropriate devices drivers and various device specific tweaks and optimizations. So while newer Android devices may one day have MPTCP in the kernel, existing Android devices will likely never support it.

Another drawback for the MPTCP kernel implementation is that it cannot be easily modified to add new features which might be standardized in the future. The reason is that the kernel is responsible for all of the essential system functionality: task scheduling, memory access, reading/writing to disk, etc. Any changes to the kernel that compromise

its stability will result in the entire system being unusable, and as such kernel changes take a significant amount of time and testing before they're deployed. Historically, the risk and effort associated with kernel updates have been one of the driving forces behind the ossification of transport protocols [71] in the Internet, and was one of the reasons Apple decided to move their network stack to a user-space implementation [40].

In this chapter, we propose a new user-space approach for deploying transport layer MC that is more readily deployable and extensible compared to a kernel implementation, requires no changes to the user device, and is backward compatible with all existing applications and device. The approach is based on two key observation. The first is that the VPN framework, which is available on all major mobile operating systems, permits a user-space application (i.e. a VPN client) to take charge of managing all network traffic to and from a user device, and thus enables it to implement new transmission logic along the path between the user device and the VPN server without any need to modify the operating system or the kernel. The second is that using UDP allows the VPN client/server to re-implement all the traditional transport layer functionalities such as reliability, rate control plus add new functionalities such as MC entirely in user-space.

Using this approach, the upper-layer steering function of ATSSS is provided using a VPN application without needing any support from the OEM or needing to wait for kernel updates. This approach further allows the operator to deploy updates quickly over-the-air, which would be impossible to do using a kernel-based transport layer implementation.

A unique advantage of the VPN approach compared to other proxying approaches such as a SOCKS proxy is that it doesn't need to be configured per application and is compatible with all protocols above layer 1 of the OSI model. For example, it is possible to create a SOCKS proxy server as an application on an Android device and use it to transmit application traffic using new transport layer logic in a similar way to the VPN approach. However, in order to use a SOCKS proxy in Android, a user needs to individually configure each of the applications of interest (e.g. web browser) to route traffic through that proxy. The fact that not all applications are proxy-aware and that this requires significant user effort and expertise makes this approach less deployable and useful than the VPN approach. Additionally, a SOCKS proxy traditionally transmits traffic over TCP and hence developers of UDP based applications such as VOIP or video chat will not include support for SOCKS proxies in their applications. While on Linux devices the TPROXY [94] option of the IPTables tool allows a proxy to transparently re-route all traffic through it, this capability is not available on un-modified Android and

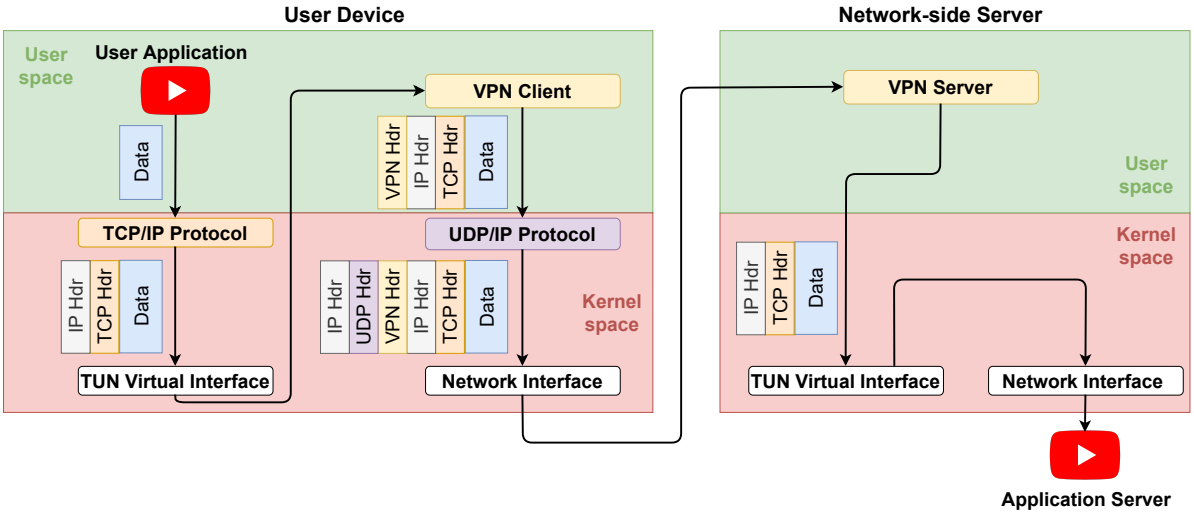


Figure 3.2: Generic VPN software architecture during uplink transmissionz

iOS devices.

Our contributions in this chapter are as follows

- We propose a new over-the-top approach to deploying next generation transport layer protocols in unmodified iOS and Android devices using the built-in VPN framework, available in all mainstream mobile operating systems, and we show how this can be used to enforce fine-grained transport layer policies for implementing advanced ATSSS in 5G networks.

- We implement a proof of concept in Android and show that its performance is equal to or better than the kernel TCP stack, and the memory and CPU overhead is minor. We further implement and evaluate an example use case of the solution to showcase its flexibility.

3.2 Overview of VPN framework in mobile operating systems

Together Android and iOS are used on approximately 99% of smart phones [36] and both offer a VPN API for developers to create a custom VPN client [98][66] and support always-on VPN. In always-on mode, the VPN tunnel is activated when the device starts and is used by all network traffic in the device. While this mode is meant to enable corporate devices to maintain secure connections all the time, an operator is able to leverage it for the purpose of enabling persistent utilization of new transport layer behavior as will be explained subsequently.

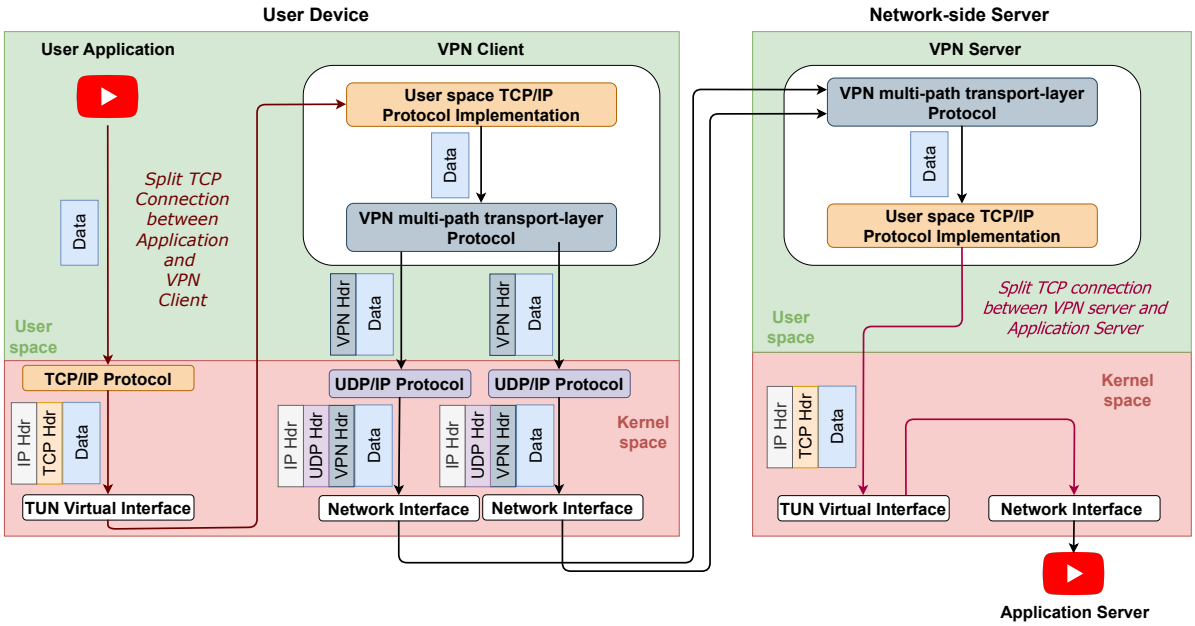


Figure 3.3: Multipath VPN software architecture during uplink transmissionz

When a VPN client starts, the operating system creates a virtual network interface and modifies the routing table such that all traffic is routed through that virtual interface. The VPN client is given control over that virtual interface in the form of a file descriptor and all network packets sent by applications will be written to a virtual file pointed to by this file descriptor. Depending on the type of interface created by the VPN client, packets are written as either layer-3 packets (i.e. IP packets in the OSI model) in TUN type interface or layer-2 (i.e. link-layer packets / ethernet packets in the OSI model) in TAP type interfaces. In the case of an uplink packet, as soon as it is written to the virtual interface, the VPN client reads it using the file descriptor, adds VPN protocol headers (e.g. IPSec) and transmits it towards a VPN server using a UDP socket. VPN clients/servers rarely use TCP since tunneling TCP on top of TCP can result in significantly degraded performance due to the so-called TCP Meltdown [12]. Once the VPN server receives the packet, it strips away the VPN headers and writes the original layer-3 or layer-2 packets to its own TUN/TAP interface so that the operating system handles routing them to their correct destination. Fig 3.2 shows the behavior of a generic VPN during transmission of an uplink packet.

3.3 MNO-managed VPN transport layer

The proposed solution leverages the VPN framework to implement new transport layer behavior, such as MC, entirely in user-space in a way that can be managed by the MNO to enable the higher layer steering function of ATSS. To enable this new behavior, a

user would simply need to install and enable a VPN client on their unmodified smart device. In addition to this approach being more readily deployable compared to MPTCP, the flexibility of the user-space implementation allows an operator to implement custom transport layer features that would be difficult to replicate inside the kernel. In this section we discuss the proposed solution in more detail.

3.3.1 Architecture

The main idea of the architecture is to use a VPN client and a VPN server as proxies to implement new transport layer behavior over the path between the user device and a network-side server, which spans the wireless hop. More concretely, on the user-device side, the VPN client will read application traffic from the virtual interface, remove their transport layer and IP headers, extract the payload and forward the payload with custom transport layer protocol headers, disguised as application headers, using multiple UDP sockets bound to different network interfaces. The custom headers can mirror the headers of any single-path or multi-path protocol such as TCP, MPTCP, SCTP [92] or even QUIC [101]. The choice of UDP is due to widespread support for the protocol and its simplicity. It does not implement any of the transport layer functions we are interested in such as reliability, rate control or in-order delivery. In Android, binding a socket to an interface is done using the multi-network API introduced in Android 5 [3]. In iOS, this binding is done by setting the socket options [56] using the IP_BOUND_IF option. On the network-side, the VPN server aggregates incoming data from all UDP flows and delivers, removes the custom headers and forward the data to the application server.

TCP applications

In order for the VPN to transparently introduce new transport layer behavior for TCP applications, it needs to first split the TCP connection, similar to a proxy, and then to use the new transport layer logic for transmitting the application data between the VPN server and client, as shown in fig 3.3. To do this on un-modified smart phones, the VPN client needs to re-implement the TCP and IP protocol logic in user space in order to allow it to transparently terminate the TCP connection from the application. While simply tunneling TCP packets would alleviate the need to re-implement the TCP and IP protocols, stacking two layers of reliability (i.e. TCP and the VPN's transport layer protocol) leads to significant performance degradation due to the so-called TCP Meltdown [12]. Also, since data arrives as packets inside the virtual interface, it is not possible to use

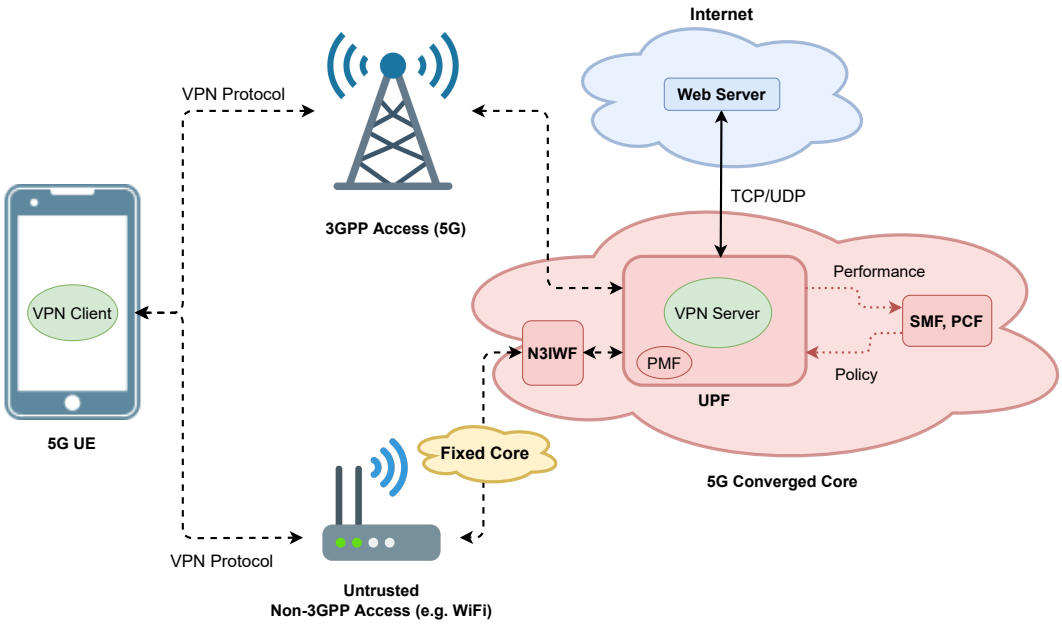


Figure 3.4: Core deployment architecture

a TCP socket to terminate the connection. Similarly, the VPN server needs to terminate the TCP connection from the application server.

UDP applications

UDP applications are much simpler to handle since UDP is a connection-less protocol. The VPN/Client server simply extracts the data from the UDP packet, transmits it between the VPN client and server over the multi-path protocol, and forwards it to the application or the application server over UDP. The option to tunnel the UDP packets is also feasible, but incurs the overhead of extra headers. Since UDP is not reliable, the VPN has to modify the multi-path protocol for UDP to disable reliability, and possibly congestion control.

3.3.2 VPN server deployment

Similar to the MPTCP architecture shown in 3.1, the proposed solution requires a network side server to host the VPN server, which aggregates the traffic from multiple links and translates it back to the original protocol. There are two main deployment options for this approach: core deployment or overlay deployment.

Core deployment In this option, the VPN server is placed inside the 5G converged core as shown in Figure 3.4. This allows tighter, i.e. less complex and more efficient, integration with the 5G core functionalities. In particular it allows the existing Policy

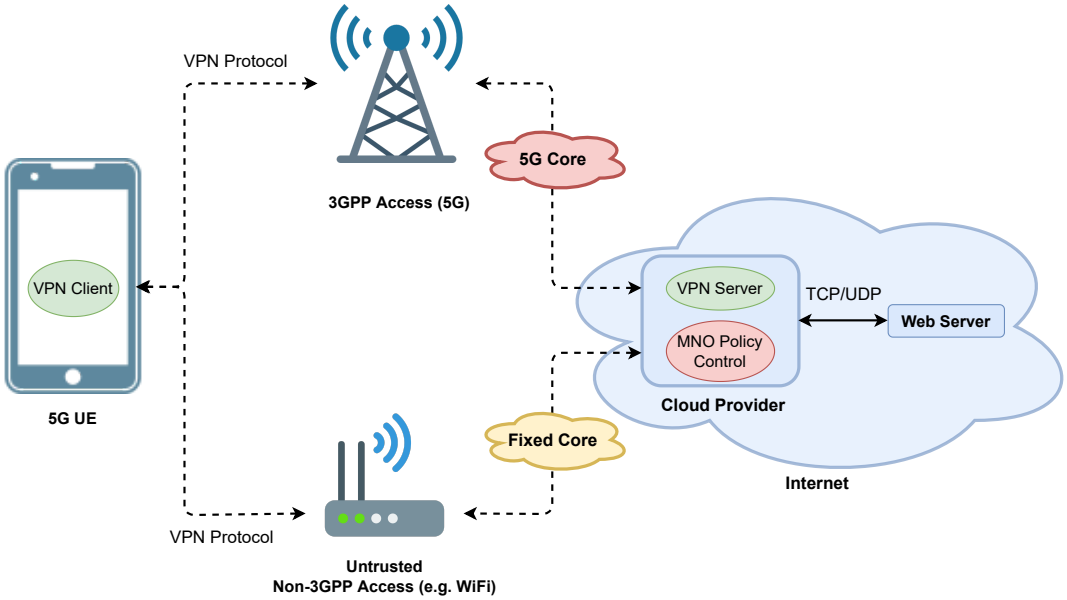


Figure 3.5: Overlay deployment architecture

Control Function (PCF), Session Management Function (SMF) and Performance Monitoring Function (PMF) to remain un-modified as the VPN server would be implemented to respond appropriately to the pre-defined signaling of these functions. The un-trusted non-3GPP fixed access is able to access the VPN server in the core using the Non-3GPP Interworking Function (N3IWF). The only drawback of this approach is that it requires a 5G converged core deployment by the MNO, which at the time of this writing is not yet widely implemented.

Overlay In this option, the hosting of the VPN server is outsourced to 3rd party cloud providers (e.g. Amazon Web Services), as shown in Fig 3.5. This is easier to deploy quickly as a 5G converged core is not needed, but it would require a different approach to policy control as the 5G functions would be unable to control the VPN server placed outside the core. One solution is to port the network policy and performance functions (i.e. SMF, PCF and PMF) into a process that runs in the cloud provider and is able to instruct the VPN server.

3.3.3 MNO management

The management of the proposed solution is based on policy signaling between the MNO policy functions and the VPN server, which is then relayed to the VPN client. To enable accurate policy selection, the VPN client will communicate the mapping between network flows and user applications to the VPN server which will be relayed to the MNO policy functions for policy selection. The MNO is then able to provision fine-grained policy to

control transport layer behavior of all network flows to improve QoS.

Mapping network flows to applications

For an MNO, mapping network flows to application is essential for enforcing per-application QoS requirements but difficult to do accurately. Traditionally, MNOs relied on TCP/UDP port numbers to map network flows to applications by using a comprehensive list of such mappings [86]. However, with the popularity of HTTP based REST services, many different applications utilize the same HTTP ports (i.e. 80 or 443) while having different QoS requirements. For example, video streaming and file downloads both use HTTP. The IP addresses of the application servers can also be used to identify specific applications, but one application's servers may have frequently changing IP addresses due to the use of Content Distribution Networks (CDN). Deep Packet Inspection is another tool that is used for traffic classification but is significantly impacted by the widespread use of encryption protocols such as TLS [105].

Using the VPN client in Android, the MNO is able to map network flows to application traffic with full accuracy. Each packet the client reads from the virtual interface contains either a TCP or UDP source port number. To map a packet to a user application, the client reads all the entries in the /proc/net/tcp and /proc/net/udp directories in Android virtual filesystem and maps the source port of the packet an application User Identifier (UID). This identifier is then used to retrieve the unique package/application name associated with this UID using the getNameForUid function of the PackageManager class [70]. Once the application name is identified, the VPN client will use the policy signaling protocol to transmit the application name along with the destination port and IP to the VPN server in order for the MNO policy function to select the appropriate network policy for handling this flow.

In iOS, the previously discussed approach is not possible due to security restrictions that create a sandbox around any iOS application preventing it from accessing any information from the operating system except those permitted by the iOS APIs. An alternative is to aggregate IP address to application name mapping collected from Android users, who are likely using similar applications, and use it to construct a database that is then used to map the packets received in the iOS VPN client to application names for MNO policy enforcement.

Policy Signaling

As discussed in Section 3.3.2, the MNO can either re-use the existing policy functions of the 5G core in the core deployment option or port them into a standalone process for use in third party cloud provider in the overlay option. In both options, the MNO policy control function will communicate with the VPN server through a pre-defined API to obtain performance metrics, such as the latency, loss and throughput, and to assign policies to each flow, using IP 5-tuple (i.e. source IP and port, destination IP and port, and protocol). The VPN server can enforce policies pertaining to down-link transmission itself but has to signal uplink transmission policy to the VPN client. This signaling will happen using a separate channel between the VPN server and client, and is independent of the VPN transport layer protocol. An advantage of this approach is that there's no need for ATSSS support in the 5G UE.

3.3.4 Installing the VPN app on user devices

We propose three different approaches for installing the VPN client in user devices

Pre-installation An MNO can pre-install the VPN client in smartphones that they sell to consumers. Most operators already ship a number of their own applications that offer extra services on their network with their smartphones. For example, T-Mobile in the US ships T-Mobile Visual Voicemail, T-MobileTV and T-Mobile Scam Shield with their Samsung Galaxy S8 smartphone [76]. This is the most attractive option since it can be done unilaterally, allowing them to bypass OEMs and consumers. According to consumer surveys [10], 55% of consumers buy smartphones directly from the MNO, allowing this option to target most users.

User installation An MNO can offer incentives for users to install the VPN client. For example, by promising users who install and enable the client with higher network speeds and improved QoS for key applications by using MC.

OEM installation MNOs can collaborate with the OEMs to ship the VPN client as part of their Android distribution. An example of this is Korea Telecom (KT) collaborating with Samsung to enable MPTCP on the Samsung Galaxy S6 and the Samsung Galaxy S6 Edge [38] for their Gigabit LTE project that combined KT WiFi hotspots with KT LTE.

3.3.5 Use cases for MNO control of transport layer behavior

In addition to enabling the basic standardized functions of the higher layer steering function defined by the 3GPP, such as allowing applications to split traffic or assigning different links to different applications, a key advantage of the VPN-based transport layer protocol is that it is readily customizable and flexible which allows an MNO to implement new capabilities that would be difficult to replicate in MPTCP's kernel implementation. Below we present 4 examples:

Priority-aware multi-path scheduling with throughput targets By aggregating WiFi and 5G on the uplink/downlink while prioritizing WiFi , the proposed solution enables a user device to achieve advertised data rates whenever possible while minimizing the utilization of the more valuable 5G capacity. Since the VPN transport layer protocol can observe the obtainable share of capacity on each link over the path between the user device and the VPN server, through monitoring of acknowledgment, and is also in charge of deciding the multi-path scheduling, it is able to easily top up the available WiFi capacity using the 5G link in order to achieve the desires throughput.

Forward error correction For latency sensitive real-time applications, such as VOIP or video chat, forward error correction (FEC) in the form of simple data repetition or more advanced coding concepts such as random linear coding, Reed-Solomon coding or rateless coding allows retrieving lost or excessively delayed data packets. The VPN transport layer protcol can transparently add FEC to the data flow of these applications in communication between the VPN client and the VPN server, while forwarding clean data to the end points to avoid any unexpected behavior. More complicated FEC schemes, such as Reed-Solomon coding, are more difficult to implement in the kernel than in user space due to the lack of access to third party libraries.

Cross-layer congestion control Usage of cross-layer congestion control that leverages information from the wireless link can improve the utilization of the wireless channel. Multiple studies into the performance of TCP in wireless networks have concluded that the dynamics of TCP congestion control result in significant performance degradation and under-utilization of the wireless resources [67][32]. One study found that TCP utilizes less than 50% of the available bandwidth in LTE networks [32]. The main culprit detected in these studies is the rapid fluctuation of the capacity in wireless links. This problem is exacerbated with the introduction of millimeter-wave links in 5G [46][74] due to the intense

susceptibility of higher frequencies to blockage caused by obstacles and misalignment. A solution to this is the adoption of cross-layer congestion control [55][106][4] which leverages information from the link layer and the NodeB, such as CQI and queue length, to predict the capacity of the wireless link, which is the typical communication bottleneck. The VPN transport layer protocol can be used for deploying such an approach since it is under the control of the MNO. Communication of the aforementioned indicators can be added as part of the signaling protocol used between the VPN client and the VPN server.

Asymmetric routing Offloading uplink traffic from WiFi to 5G can significantly improve WiFi downlink capacity by removing MAC layer contention, especially when servicing a large number of users [51]. Data rates over the WiFi interface degrade quickly as the number of users increases due to uplink contention based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) [97][57][5]. The VPN transport layer protocol in the VPN client can offload WiFi acknowledgments to 5G using asymmetric routing to improve the WiFi capacity, for example, in situations with a large number of WiFi users.

3.4 Example implementation and evaluation

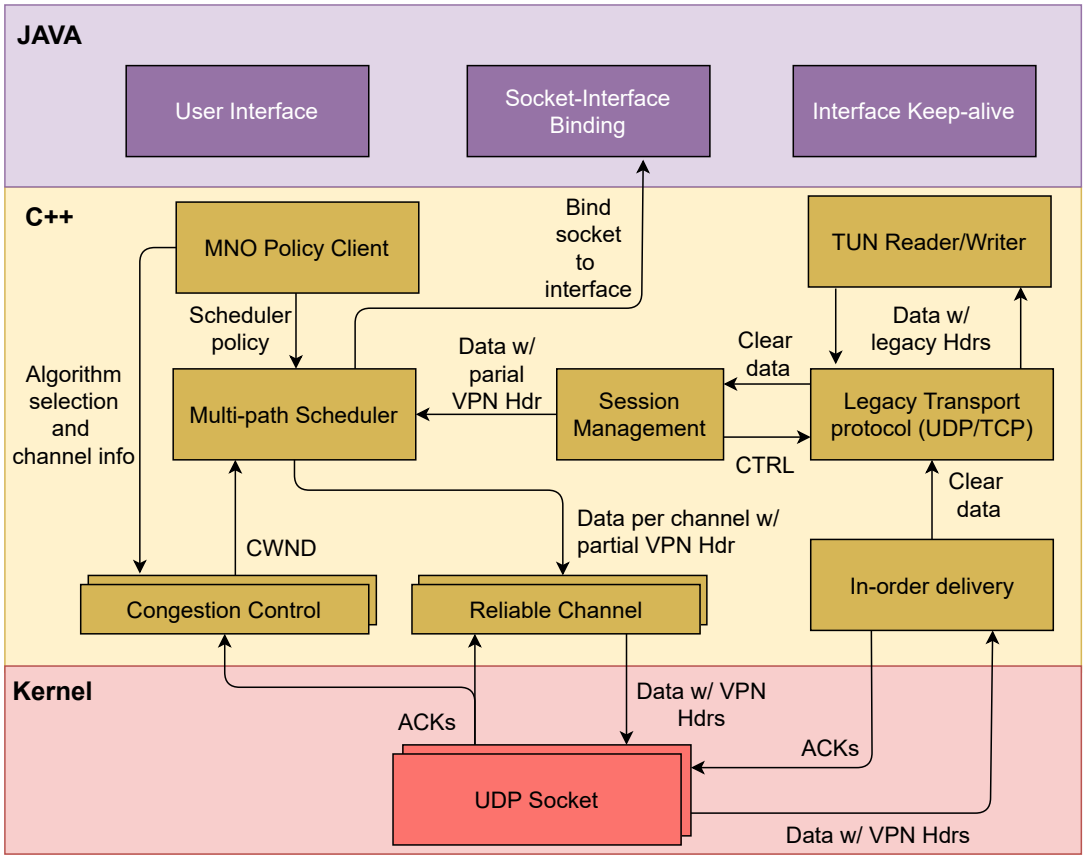


Figure 3.6: Android implementation architecture

To verify the viability of the proposed solution in terms of performance, since an overhead is expected due to the user-space implementation, we created an implementation for the Android operating system and evaluated its efficiency and system overhead compared to the kernel. We further implemented the *Priority-aware multi-path scheduling with throughput targets* example use case discussed in Section 3.3.5 to demonstrate the flexibility of the solution.

3.4.1 Android Implementation

The implementation is done mainly in C++ as a native Android application, with certain functions that require the Android SDK being in Java. As soon as the Java application is started by the user or the operating system, it loads the native library that contains the C++ code and executes the main function which initiates the TUN reader and TUN writer threads. The implementation runs a simple MPTCP-like protocol for evaluation purposes. The key modules as shown in fig 3.6.

TUN Reader/Writer Two separate threads are assigned to read and write layer 2 packets from and to the TUN interface. Since the memory access involved in reading and writing can easily become a bottleneck, we configured the TUN interface to utilize the maximum allowed MTU size, which is 64,000 bytes. By doing this, the VPN client is able to write 42 times the data length that it would be able to read with a standard MTU size (i.e. 64000/1500 = 42). Furthermore, when terminating TCP connections, the TCP protocol implementation in the VPN client is configured to respond to the TCP MSS discovery of the user's application TCP connection with the same large value in order to inform it that it may send 64000 byte packet when it can, improving the reading efficiency.

Transport Protocol Interface An implementation of both TCP and UDP is used to allow interfacing with the existing transport protocols of the user device. Its main job is to allow the VPN client to behave as a proxy for both TCP and UDP connections. In the case of UDP, this is fairly simple as it will simply remove UDP headers from packets read in the tunnel and add headers when writing them back to the tunnel. In the case of TCP, as discussed previously in Section 3.3.1, this module terminates the TCP connection in the tunnel. This means it has to mimic all aspects of the TCP protocol, including handshake and termination sequences, and maintain the necessary state, such as sequence and ack numbers.

Session Management This module creates and manages the different sessions that handle the individual TCP and UDP connections going through the VPN. In particular, it appends a unique flow ID to each packet to indicate the parent connection. This ID is generated using a monotonically increasing 4 byte unsigned integer counter that is incremented with each new IP 5-tuple (i.e. source port, destination port, source IP, destination IP and protocol). On the first packet encountered for a connection, it will set a 1-bit SYN flag to indicate to the VPN server that this is a new connection, and it will append the destination address of this connection. The corresponding module in the VPN server will instruct the transport protocol module to initiate a new connection with that destination address. If that connections fails, the module will transmit a packet with 1-bit RST flag set, which will indicate to the VPN client that the connection fails and that it should terminate the connection on its side. If the connection succeeds, it will transmit a packet with 1-bit SYN flag and a 1-bit ACK flag set to indicate success. On connection termination, a similar process takes place, but with the 1-bit FIN flag set. During data exchange, the session management will construct a partially filled VPN header that is forwarded to the multi-path scheduler. In this header, only the session-level sequence number (DSS) flag is filled in. This DSS value is incremented by the amount of bytes transmitted by the session thus far.

Multi-path Scheduler This handles splitting, steering or load-balancing the data across different links, according to the policy and the selected scheduling algorithm communicated by the MNO policy client, and the available link capacity. In our implementation, we implemented the *Priority-aware multi-path scheduling with throughput targets* described in Section 3.3.5 and a simple round-robin scheduler.

Reliable Channel This module handles reliable transmission by tracking the reception of packets through received acknowledgments (ACKs) and performing re-transmissions based on quick re-transmissions, as defined in New Reno [75], and return-trip timeouts [75]. Each packet that passes through this module has a sequence numbers (SEQ) which reflect the number of bytes transmitted on a channel so far to allow the receiver to perform in-order delivery and send the appropriate ACKs. This module is disabled when handling UDP sessions.

In-order delivery This transmits ACKs for received data and handles re-ordering of received data so it is delivered reliably and in-order. ACK numbers transmitted represent

the highest SEQ number received in-order. When receiving out of order SEQ numbers, duplicate ACKs are transmitted. This module is disabled when handling UDP sessions.

MNO Policy Client This retrieves policies from the 5G policy function and communicates it to the relevant modules. Upon the VPN client starting, this module will establish a TCP connection to the MNO policy server running inside the VPN server and keep this connection alive during the life time of the VPN client.

Interface Keep-alive & Socket-interfae Binding These are Java modules that handle keeping both network interfaces (5G and WiFi) active, while allowing the C++ code to bind sockets to each interface when needed. First, un-used interfaces are turned off by Android to save energy, and hence to keep them alive, a thread needs to be constantly requesting the network from the Android ConnectivityManager API, which is the function of the Interface Keep-alive module. Second, binding a socket to a network interface requires root privileges if done in native code (i.e. C or C++) through socket options, but can be done by any Java app by utilizing the ConnectivityManager API. Hence, we created a Java module that contains a functions for creating a socket, binding it to a network and returning the file descriptor of the socket through Java reflection. We exposed this module to the C++ code through the Java Native Interface (JNI). While keeping the interfaces alive constantly might lower battery life, its impact is minor since most battery life in smartphones is used by the display.

3.4.2 Efficiency and system overhead evaluation

To better understand the efficiency and system overhead of the proposed solution, we evaluated the CPU usage and the peak rate possible with and without the VPN client.

Setup

We installed our Android VPN client on a Nexus 5 Hammerhead smartphone with a quadcore processor running Android 5.0. We picked an older phone and operating system in order to show that the VPN overhead is not too high for older hardware and that the approach is compatible with older versions of Android, allowing it to be deployable on the majority of existing devices. We setup a local VPN server connected via 1Gbit/s ethernet cable to an 802.11ac WiFi AP on 5Ghz using an 80 MHz channel. The smartphone was connected to the AP via its WiFi interface only. We used a single link since we are only interested in evaluating the efficiency of the proposed solution here. We ran an iPerf3

uplink TCP test with no rate limits from the smartphone, both with and without the VPN client in order to compare the efficiency and overhead. The TCP congestion algorithm in Android was configured to Reno to match the VPN's congestion control. The choice of Reno was due to it being simpler to implement, while still allowing a fair comparison with the kernel.

3.4.3 Results

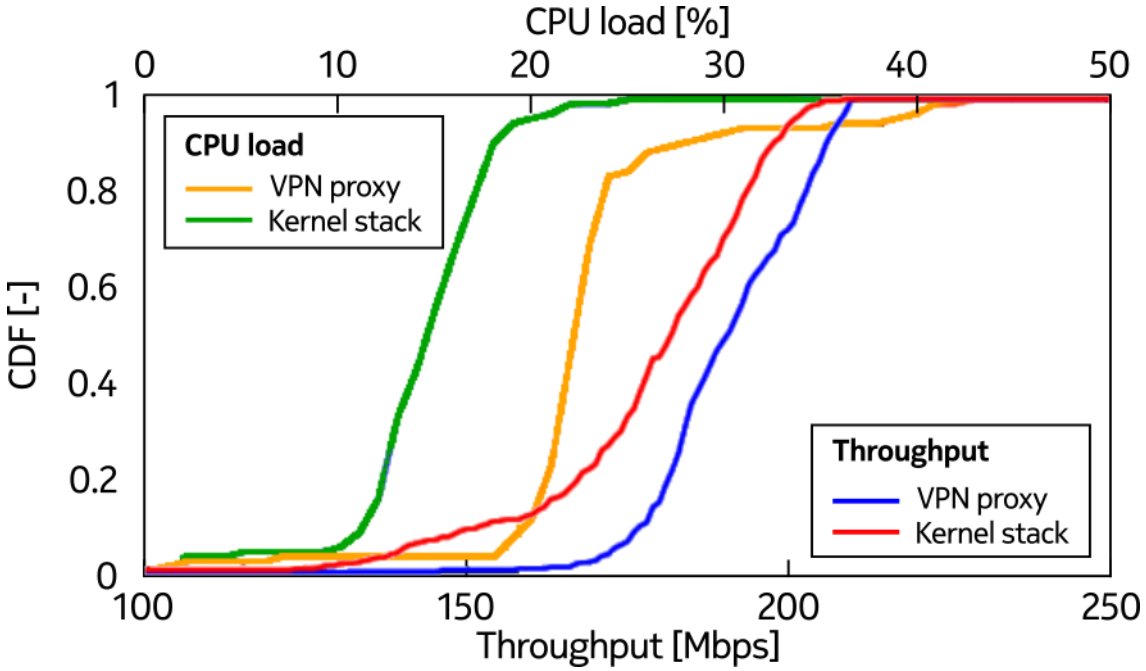


Figure 3.7: Proxy performance (network limited): Probability distribution function of achievable throughput and the associated CPU usage by Nexus 5 smartphone by aggregating LTE and Wi-Fi network.

Fig 3.7 shows the CDF for the throughput and the CPU load during the test for both the VPN scenario and the kernel scenario. The VPN was was able to achieve around 190 Mbit/s mean throughput while the kernel achieved somewhat less at around 175 Mbit/s. In terms of CPU, the VPN consumed roughly an additional 20% of CPU time on average (i.e. one extra core out of 4). Since the capacity of the network was around 800 Mbit/s, confirmed with a test using a laptop with a much more powerful single core speed instead of the smartphone, it is clear that in both test scenarios the bottleneck was the CPU performance of a single core . In terms of memory usage, there was negligible memory overhead from the VPN, around 2% memory used.

We attribute the higher VPN throughput to two optimizations: the larger MTU size on the TUN interface which allows the iPerf to transmit more data with fewer context switches, and the fact the VPN transmits multiple UDP packets with a single system call to sendmmsg. Together these optimizations allows the iPerf thread to transmit more data

using the same amount of CPU time than it would with a normal MTU size and for the VPN client to match this efficiency.

The higher CPU usage is explained by observing that in the kernel scenario, only iPerf was running and it was fully utilizing one CPU out of the 4 cores available in the smartphone. In the VPN scenario, both iPerf and the VPN client were fully utilizing separate cores, and hence the CPU usage roughly doubled. We can thus assume that the CPU usage overhead associated with packet processing of the VPN client will be equal to the user application it is serving, since it is transmitting the same amount of traffic. Considering that when applications are transmitting low throughput they have very low CPU usage, the added overhead is minor.

3.4.4 Evaluation of Priority-aware multi-path scheduling with throughput targets

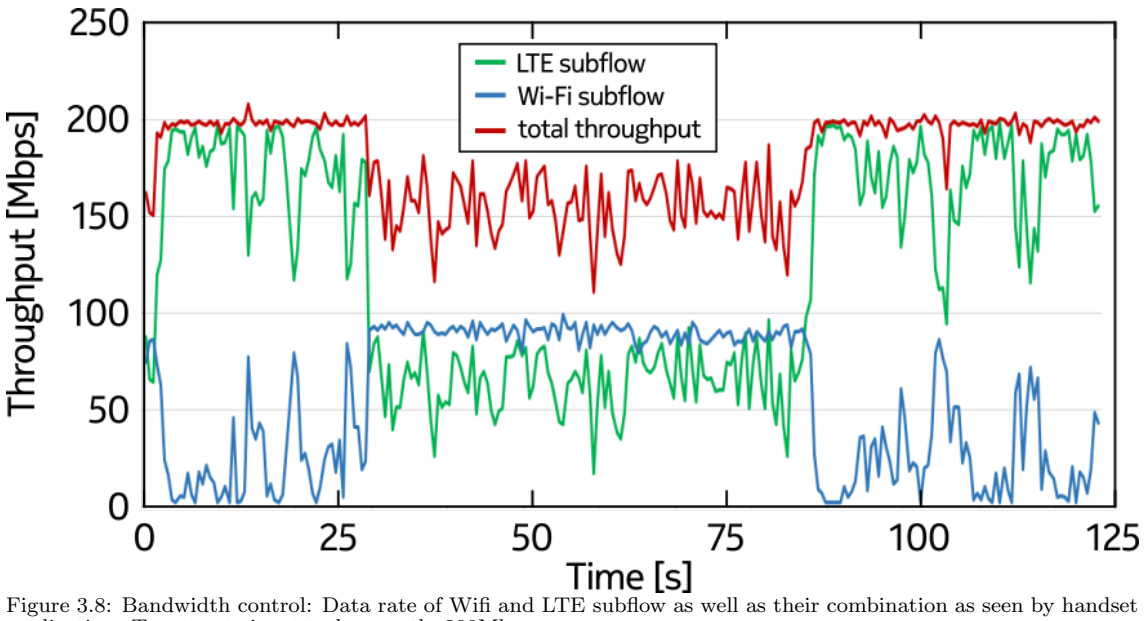


Figure 3.8: Bandwidth control: Data rate of Wifi and LTE subflow as well as their combination as seen by handset application. Target rate is set to be exactly 200Mbps.

To demonstrate the flexibility of the approach, we implemented a *Priority-aware multi-path scheduling with throughput targets*. This scheduler attempts to achieve a target throughput while prioritizing certain links. In this example, the scheduler in the VPN server was configured to maintain 200 mbit/s while prioritizing using WiFi over the LTE.

Setup

In this experiment, the Android VPN client was installed on a Samsung galaxy LTE tablet running Android 5.0 with a dual-core processor. A VPN server was setup in an Amazon

Web Services (AWS) virtual machine so it can be accessed through LTE. A 802.11ac WiFi AP on 5Ghz using an 80 MHz channel was setup with a 1 Gbit/s internet backhaul. The tablet was connected to the VPN server through both WiFi and LTE. An iPerf3 downlink TCP test with no rate limits was started from the tablet.

Results

Fig. 3.8 shows the evolution of the Wifi and LTE capacity in the experiment. Midway through the expirement, constant-bitrate Wi-Fi cross-traffic was introduced causing Wi-Fi throughput drop, as well as the aggregated throughput as seen by the user application. As observed, when the LTE link is able to compensate for the WiFi drops, the sum rate is very stable despite the link volatility and minor cross-traffic presence - any single-path impairments are efficiently masked from the application. When the LTE is not able to compensate, it utilizes the available capacity fully.

3.5 Related Work

The challenges facing the deployment of both MPTCP and new transport protocols in general have been discussed in multiple studies [71][24][30][73][25][58][65], albeit not in the context of ATSSS or MNO control.

3.5.1 Deploying new transport protocols

In [71], the authors discuss the ossification of network protocols and points out that one reasons is due to the effort and the number of stakeholders require to modify the operating system kernel code. They propose a new transport layer framework that, amongst other design features, would allow users to install, use and upgrade new transport protocols without special privileges from the operating system to enable the speed of the evolution of the protocols to be independent of the speed that operating systems are updated, but they do not provide a more specific technical proposal. In [30], the authors argue that the blame for the slow evolution of the TCP protocol (with extensions taking many years to become widely used) should be placed on end systems, and not on middle boxes. They revisit the case of user space protocol stacks in order to ease the deployment of new protocols, extensions or performance optimizations. They introduce MultiStack, which is a kernel framework that allows the creation of user space libraries that implement transport protocols with enough support from the kernel to be highly efficient. In [73], authors argues that the difficulty in upgrading transport protocols is that most upgrades

can only be used when both ends support them, and that has lead to most upgrades being quick fixes that try to be useful even if only one side supports them, such as New Reno Fast Retransmission. Meanwhile more useful upgrades such as SACK are slower to be adopted. as they need support on both ends. The authors propose a new kernel framework for creating a self-spreading transport protocol (STP). It works by allowing code for a transport protocol to be transmitted from one end point of a connection to the other and then used for communication between the two end points. In both [25] and [58], authors propose a modified kernel TCP stack that allows user-space code to retrieve and modify the TCP state variables to enable user space control on the transport protocol. One example use case proposed is to migrate a connection from one host to another by retrieving the sequence and acknowledgment number from the TCP connection on the old host and setting them on the TCP connection in the new host so that the connection can be migrated seamlessly.

3.5.2 Deploying MPTCP

Authors in [24] discuss the challenges facing MPTCP merging with the mainstream kernel and in particular the mixing of the MPTCP and TCP code, which they claim substantially increases the complexity of the Linux kernel TCP/IP implementation and slows down integration. As a solution they suggest completely isolating the TCP and MPTCP code and present various tweaks to the MPTCP codebase to accomplish this. In [65], the authors used the Linux Kernel Library (LKL), which is an architectural variant of the Linux kernel that compiles it as a linkable library, to compile the MPTCP kernel as a library and have applications load it to make use of MPTCP. In order to get unmodified applications to do this, the LD_PRELOAD parameter is used when starting any application to dynamically re-route any calls to the socket API through the MPTCP LKL. Both LD_PRELOAD and the LK library require special privileges and hence the device must be rooted.

All the works reviewed either require modifying the existing kernel or rooting the device in order to enable a new transport layer protocol.

3.6 Conclusion

In this chapter we proposed a novel approach for deploying new transport layer behavior in unmodified smartphone using the VPN framework in a way that can be managed by the MNO for enforcing fine-grained policy control of the transport layer. We show how various architectural options and strategies for implementing the solution. We implemented a

proof of concept in the Android operating system and demonstrated that it has good efficiency and performance compared to the legacy kernel transport protocols. Finally, we constructed an example use case where the the mulit-path protocol on the device is instructed to maintain a target throughput while prioritizing the WiFi interface and evaluated over real networks.

Chapter 4

Low Delay Scheduling of Objects Over Multiple Wireless Paths

4.1 Introduction

In this chapter we consider the task of scheduling packet transmissions amongst multiple paths with uncertain, time-varying packet delay. We make the observation that the requirement is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the packet delay which is important. We define objects as the atomic unit of data that the application needs to receive fully before any action is taken. Depending on the application, an object can fit inside a single packet (e.g. 10ms of speech VOIP) or it can span hundreds of packets (e.g. UHD video). This observation fundamentally changes both scheduler design and the scope for making use of links with fluctuating packet delay. Firstly, in-order delivery of packets is no longer important but rather it is the time when all of the packets forming an object are received which is the key quantity of interest. Secondly, when the requirement is to transmit individual packets subject to a delay deadline then it is difficult to make use of a path with highly fluctuating packet delay since many packets will miss the deadline. However, when the requirement is to transmit an object consisting of many packets then statistical multiplexing means that it is indeed possible to use a path with fluctuating packet delay while ensuring low object delivery delay with high probability, as we show in more detail later.

Our main contributions are summarized as follows:

1. We propose SOS (Stochastic Object-aware Scheduler), a multi-path scheduler that takes in consideration the size of the application layer object and the uncertainty of the

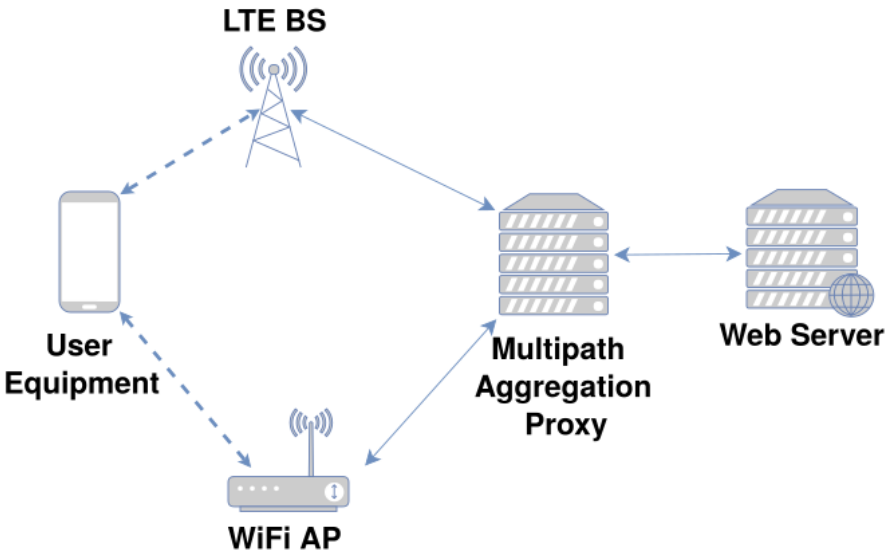


Figure 4.1: Schematic of a cloudlet-based edge transport architecture.

path packet delay in order to deliver objects with a reliable bound on delay, making it suitable for latency-sensitive applications. We offer a low complexity implementation that can execute in $O(\log n)$ time for two links, where n is the number of packets in an object. We demonstrate that SOS reduces the 95% percentile object delivery delay by 50-100% over production WiFi and LTE links compared to state-of-the art schedulers.

2. We extend SOS to utilize FEC (Forward Error Correction) in the form of linear block codes that match the object sizes. By introducing redundancy in this way we show that the scheduler is able to opportunistically exploit periods when the fluctuating path packet delay is low so as to further reduce object delivery delay, albeit at the cost of slightly reduced throughput capacity.

3. We also extend SOS to interface with cwnd-based congestion control, using Reno as an example, in order to optimize scheduling when an object partition is split across multiple cwnds. We implement this in a user-space network stack, evaluate its performance against MPTCP/minRTT under different link conditions and using different object sizes and show up to a 150% improvement in the 95% percentile and mean object delivery delay is achieved.

4.2 Preliminaries

4.2.1 Application Layer Objects

We begin by noting that the number of packets used to transmit an object is a direct proxy for the object size. The underlying mechanism is that an object of size N bytes, such as a

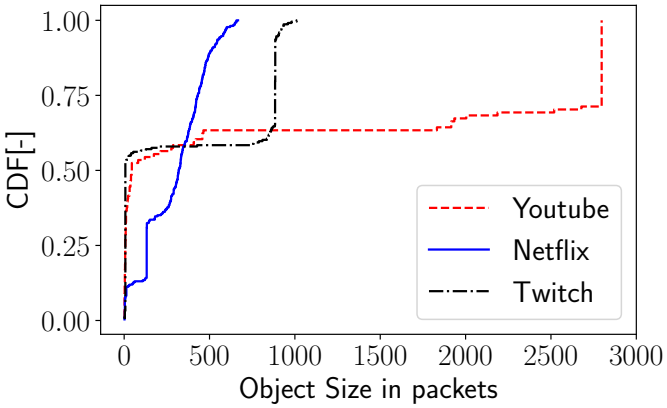


Figure 4.2: HTTP object sizes for videos offered by 3 popular streaming websites (Youtube, Netflix and Twitch).

video frame, is typically sent by an application using a TCP socket. The networking layer within the sender operating system then segments the object and transmits it over the network as $\lfloor N/MSS \rfloor + 1$ packets, where MSS is the network path maximum segment size. Some applications use UDP, in which case the segmentation is carried out by the application itself, but the focus in the present chapter is on objects sent using TCP with its associated reliability and in-order delivery guarantees.

Applications employ different sizes of objects/frames depending on the type of the application. For example, Figure 4.2 plots the measured distribution of HTTP object sizes (measured in packets) for three popular streaming web sites•. Observe that YouTube video object sizes can be as large as 2500 packets. Similarly, TLS (Transport Layer Security protocol) employs its own framing on top of TCP/UDP which can span up to 13 packets [14] in v1.2, with data unable to be decrypted until a frame is received completely. Compression algorithms such as gzip behave in a similar way.

These observations are pertinent because they mean that the relevant delay for applications is the time it takes to deliver an application object, rather than the time taken to deliver individual packets. Hence, for example, application latency may not be improved by in-order packet delivery (since what matters is that all of the packets forming an object are received, not their order of arrival) and so schedulers that minimize usage of variable-delay links that cause head of line blocking might be inadvertently hurting application latency by reducing aggregate capacity.

•Data was collected using HTTP Archive (HAR) files obtained from the Chrome web browser in March, 2018. The object sizes were inferred from the HTTP response content-length field. Youtube video measured is "Ghost Town 8k". NetFlix movie measured is "The Sinner" and a random Twitch stream active at the time was selected.

4.2.2 Packet Delays

For a stream of packets indexed $k = 1, 2, \ldots$ we let *packet delay* $P + \Delta_k$ denote the time between when a packet is transmitted and when it arrives. Here P is a constant associated with *propagation delay* across the path from sender to receiver and *jitter* Δ_k is a fluctuating delay associated with effects such as link layer retransmissions, queueing etc. The *inter-packet delay* is the time $T_k := P + \Delta_k - (P + \Delta_{k-1}) = \Delta_k - \Delta_{k-1}$ between the arrival of packets $k-1$ and k, where we define $T_1 = \Delta_k$. We assume there is no reordering within a path (although there may well be reordering across paths) and so the inter-arrival delay $T_k \geq 0$.

For an object transmitted across a path we also define the *object delay* to be the time taken from transmission of the first packet from the object through to reception at the destination of all of the packets forming the object.

4.2.3 Variability of LTE and WiFi Packet Delays

Figure 4.3 plots example measurements of packet delay on production WiFi and LTE paths, taken from [72]. These measurements are for UDP packets in order to eliminate the influence of TCP congestion control on the variability of the packet delay, and are measured between a UE with WiFi and LTE interfaces and a server, both located in the same city. It can be seen that a range of packet delay behaviours are observed, with the packet delay behaviour of WiFi and LTE often differing significantly. For example, in Figure 4.3(a) it can be seen that the WiFi link is on average faster than the LTE link (has lower mean packet delay) but also has higher variability/jitter. Conversely in Figs. 4.3(c) and 4.3(d) the LTE path has much higher packet delay variability than WiFi. The magnitude of the packet delay fluctuations is also quite variable. For example in Figure 4.3(b) the packet delay fluctuations are around 10-20ms whereas in Figure 4.3(c) the LTE packet delay fluctuations can be as high as 600-800ms.

4.2.4 Multipath Schedulers: State of the Art

Perhaps the most well known multipath scheduler is the MPTCP default scheduler, min-RTT [79]. This scheduler attempts to send packets out on the link with the lowest RTT (Round Trip Time) first until its cwnd (Congestion Window) is full and then moves to the next lowest RTT link, filling its cwnd. Once all cwnd are full, it will send packets on a link as soon as there is space in its cwnd. Additionally, the scheduler will re-inject packets that are causing HoL blocking on one link into another link and penalize the link

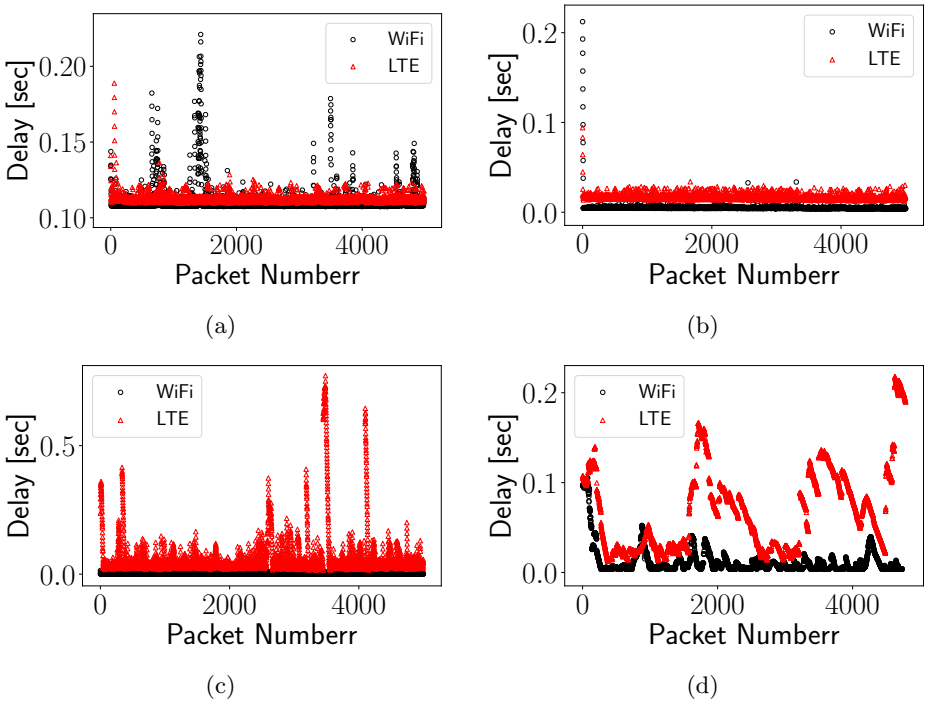


Figure 4.3: Packet delay measured while transmitting 5000 UDP packets to a server from 4 different locations in Dublin, Ireland.

that caused the HoL blocking by halving its cwnd [79].

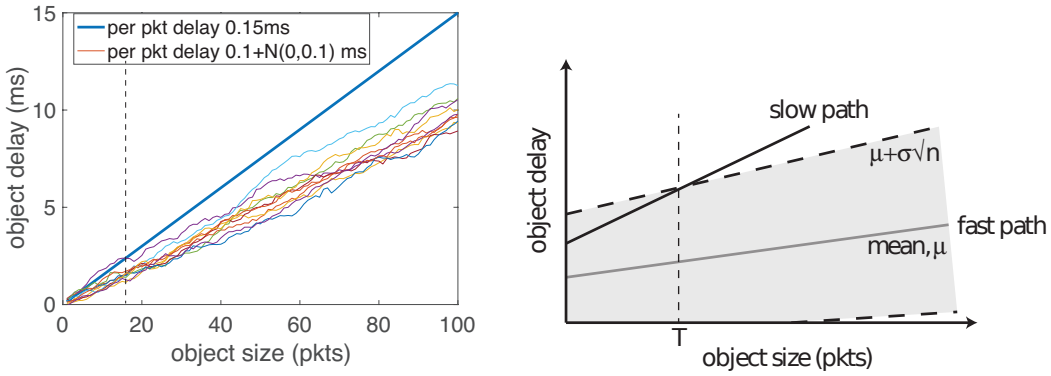
In [17][83][104] and [8] various approaches are proposed for reducing out of order delivery in minRTT. In [17], a Blocking Estimation (BLEST) scheduler is introduced. BLEST addresses the issue of HoL blocking when one path is slower than the other, in which it is possible for the faster link to deliver multiple cwnds worth of packets in the time taken for the slower link to deliver one cwnd worth of packets which, using minRTT, leads to out of order delivery. BLEST estimates the amount of packets that can be delivered on the faster path while the slower path delivers one cwnd of packers. It also accounts for growth of the cwnd on the faster subflow during the transmission. In [83], a Delay Aware Packet scheduler (DAP) is introduced. DAP creates a schedule based on the Lowest Common Multiple (LCM) of forward delays on all paths. For example, if the forward delay on the slower link is 5 times that of the faster link, then DAP will transmit packets 1 to 5 on the faster link and packet 6 on the slow link. One issue with DAP is that once a schedule is created it will not be modified until completed, which causes it to be less reactive than other approaches. In [104], the Out-of-order Transmission for In-order Arrival Scheduler (OTIAS) is introduced. OTIAS will queue packets to paths regardless of whether they have free space in the cwnd. It schedules on a per-packet basis as soon as the packet arrives and creates a queue on each path. In [50], an Earliest Completion First (ECF) (also commonly referred to as an Earliest Deadline First/EDF) scheduler is

introduced, which takes into consideration the RTT of each path and decides whether it is faster for a packet to wait for the faster path to have space open in its cwnd or to send it on the slower path instead. It also attempts to address path delay variance in a simple manner by calculating the standard deviation of the RTT on each path and adding it to the value of the RTT on each path. One issue with this approach however is that it will not schedule packets out of order and as such if it is better for the current packet to wait for the faster path, other packets waiting in the send buffer will also have to wait.

In [103] and [68] the impact of packet losses is considered, in addition to delay. In [103], a scheme is introduced to avoid scheduling packets on congested paths, aimed at reducing losses. The scheme calculates an estimate for the path capacity based on the value of the cwnd half way between the last loss event and the current SSThreshold. Once the number of in-flight packets exceeds this estimate, the estimate is updated to the value of in-flight packets. Furthermore, a threshold is established to identify when the path is congested based on the ratio between the number of in-flight packets and the capacity estimate. A packet is then scheduled on a path if it will not cause it to exceed the congestion threshold. If both flows are below the threshold, the path with the lowest RTT is selected. Essentially, this scheme trades packet delay for reducing the probability of packet loss. In [68] , a Fine-grained Forward Prediction based Dynamic Packet Scheduler (F^2P-DPS) is introduced, which attempts to calculate how many cwnds worth of packets can be transmitted on the faster path while the slower path delivers its cwnd, taking into consideration losses. It calculates the expected amount of losses on each link and whether recovery will be done via RTO or Fast Retransmission.

In [82], an MPTCP scheduler is proposed specifically for aggregating 802.11ad and 802.11ac. It transmits data across the two interfaces based on a ratio that aims to maximize total throughput. A 50/50 ratio is initially assigned. Probing commences, evaluating neighboring ratios (e.g. 51/50 and 49/51) in order to observe if one of the probed ratios leads to an aggregate throughput improvement. If a probed ratio does result in an improvement, it is selected and probing repeats using the new neighboring ratios. This algorithm is predicted to converge to to the optimal ratio eventually.

In [28] the authors propose MP-DASH, an application-aware MPTCP scheduler for DASH videos that aims to minimize the usage of the cellular connection in a device that has both WiFi and cellular connectivity, while maintaining continuous playback. MP-DASH is able to augment any MPTCP scheduler with the intelligence of how to control the cellular interface. It simply toggles the cellular interface on/off depending on whether



(a) (b)

Figure 4.4: Illustrating time taken to transmit an object over a slower path with fixed delay and over a faster path with fluctuating delay. Plot (a) shows some example paths when the inter-packet delay on the faster path is normally distributed with standard deviation $\sigma = 0.1$ms. The shaded region in plot (b) indicates schematically the mean plus one standard deviation in transmission time for the faster path. The vertical dashed line on both plots indicates the object size above which the delay on the faster path is, with high probability, lower than that of the slower path.

the estimated WiFi throughput is sufficient to maintain the buffer occupancy level, which would ensure playback would never stall. The scheduler does not attempt to throttle cellular throughput when cellular connectivity is enabled, but rather uses it at its full capacity.

Perhaps the closest work to the present chapter in the sense that it takes explicit account of delay variability when making scheduling decisions is [22], which introduces Stochastic Earliest Deadline Path First (SEDPF). SEDPF models inter-packet delay as i.i.d. Gaussian random variables. Using the Gaussian assumption, it calculates the distribution of the total inter-packet delay of in-flight packets on each link with the new packet added. It then evaluates the maximum of multiple sets of Gaussian random variables, using an approximate closed form, each representing the delay of a packet being sent on a particular path. It then selects the set that had the lowest maximum, as that implies the lowest delay.

None of the above schedulers exploit knowledge of application layer objects and their impact on scheduling decisions, which is our focus in the present chapter. With the exception of [22], existing schedulers also deal only indirectly with the time-varying and uncertain nature of path delay, namely by utilizing a smoothened average RTT and applying this to all packets sharing the same cwnd[§] when making scheduling decisions.

4.3 Scheduling Objects

4.3.1 QoS Over Paths With Time-Varying Delay

Consider two paths, one of which is slower but has consistent packet delay and the other which is faster but has highly fluctuating packet delay. When the requirement is to transmit individual packets subject to a packet delay deadline then it is difficult to make efficient use of the link with fluctuating packet delay since many packets will miss the deadline. However, when the requirement is to transmit an object consisting of multiple packets subject to an overall deadline then the situation changes fundamentally.

The reason why is illustrated in Figure 4.4. Figure 4.4(a) plots sample paths of the object delay (the time taken for all of the packets forming an object to arrive) on a fluctuating faster path vs the object size in packets. It can be seen that the variance of the object delay increases with the object size, as expected since the object delay $P + \sum_{i=1}^{n} T_i$ is the sum of the inter-packet delays T_i and when these are i.i.d this has standard deviation $\sqrt{n}\sigma$ where σ is the standard deviation of the inter-packet delays T_i. However, the object delay on the slower path scales with n, see Figure 4.4(b). Hence, for a sufficiently large object size the object delay will, with high probability, be lower on the faster path despite its highly fluctuating delay.

This basic observation has significant implications. In particular, it means that we can still make efficient use of wireless links with highly fluctuating inter-packet delay (e.g. mmWave, LiFi) while providing controlled quality of service, provided that when scheduling packets across paths we move from consideration of packets individually to consideration of objects (collections of packets subject to an overall delivery deadline).

4.3.2 Object Scheduler Design

In light of the above observation we would like to design a multipath packet scheduler that is cognisant of (i) objects within the packet stream and (ii) the impact of fluctuations in inter-packet delay on object transmission time.

Index the available paths by $j = 1, 2, \ldots, m$ and let $T_k^{(j)}$ and $P^{(j)}$ denote the inter-packet delay and propagation delay, respectively, experienced by packet $k = 1, 2, \ldots$ transmitted on path j. Given an object consisting of n packets to be transmitted across the paths let $n^{(j)}$ denote the number of packets sent over path j, with $\sum_{j=1}^{m} n^{(j)} = n$, and

[§]The difference in delay between the head of the cwnd and the tail can, for example, be large when network buffers start to fill.

$\vec{n} = [n^{(1)}, \ldots, n^{(m)}]^T$. The object delay is then given by

$$D(\vec{n}) := \max_{j \in \{1,\ldots,m\}} T^{(j)}(n^{(j)}) + P^{(j)} \tag{4.1}$$

where $T^{(j)}(n^{(j)}) = \sum_{k=1}^{n^{(j)}} T_k^{(j)}$. The difficulty in designing a scheduler is, of course, that the inter-packet delays $T_k^{(j)}$ are variable and unknown, and so also the aggregate delays $T^{(j)}(n^{(j)})$. While we might attempt to calculate the probability distribution of object delay $D(\vec{n})$ for every partition $\vec{n}$ this quickly becomes computationally expensive for larger object sizes (so infeasible for real-time packet scheduling) plus in any case we usually lack full details of the distribution of the inter-packet delay on each path. We therefore adopt the following approximate approach and select parameters $w^{(j)} \geq 0$ such that

$$P(T^{(j)}(n^{(j)}) \geq T_U^{(j)}(n^{(j)})) \leq \epsilon^{(j)} := \epsilon/m \tag{4.2}$$

where $\mu^{(j)} = E[T_k^{(j)}] \geq 0$, $T_U^{(j)}(n^{(j)}) = n^{(j)}\mu^{(j)} + \sqrt{n^{(j)}}w^{(j)}$ and ϵ is a design parameter which, unless otherwise stated, in the rest of the chapter we select to be 0.05, providing reliability at the 95% percentile. In particular, modelling the inter-packet delays $T_k^{(j)}$ as being i.i.d then using the Chernoff-Hoeffding bound we can solve for $w^{(j)}$ using

$$w^{(j)} = \sqrt{\frac{-\ln(\epsilon^{(j)})(a^{(j)} - b^{(j)})^2}{2}} \tag{4.3}$$

where $a^{(j)}$ and $b^{(j)}$ are upper and lower bounds on the inter-packet delay, $a^{(j)} \leq T_k^{(j)} \leq b^{(j)}$. It follows from (5.4.3) that

$$P(D(\vec{n}) \geq D_U(\vec{n})) \leq 1 - (1 - \epsilon/m)^m \leq \epsilon \tag{4.4}$$

where $D_U(\vec{n}) = \max_{j \in \{1,\ldots,m\}} T_U^{(j)}(n^{(j)}) + P^{(j)})$. We now select a partition $\vec{n}$ that solves the following optimization problem,

$$\min_{\vec{n} \in \mathbb{N}^m} \quad D_U(\vec{n}) \quad s.t. \quad \sum_{j=1}^{m} n^{(j)} = n \tag{4.5}$$

Solving (4.5)

When we relax the optimization (4.5) so that $\vec{n} \in {\mathbb{R}_+}^m$ i.e. a fractional number of packets can be sent on each path, then for objects consisting of a sufficiently large number of packets the solution to the optimisation ensures equal 95th percentile delay on all paths

and so is of the Wardrop form. That is, we have:

Lemma 1 (Wardrop Optimum)**.** *Suppose that for each path $j = 1, \ldots, m$ the mean inter-packet delay $\mu^{(j)}$ and inter-packet delay variability parameter $w^{(j)}$ are not both zero i.e. the inter-packet delay is not identically zero on any path. Then the solution $\vec{n}^*$ to relaxed optimization $\min_{\vec{n} \in \mathbb{R}_+{}^m} \quad D_U(\vec{n}) \quad s.t. \quad \sum_{j=1}^m n^{(j)} = n$, satisfies*

$$|T_U^{(j)}((n^*)^{(j)}) - T_U^{(i)}((n^*)^{(i)}) + P^{(j)} - P^{(i)}| = 0 \tag{4.6}$$

for all $i, j \in \{1, \ldots, m\}$ provided n sufficiently large that all elements of $\vec{n}^*$ are non-zero i.e. all paths are used to transmit packets.

Proof. We proceed by contradiction. Suppose $\vec{n}^*$ is optimal and $D_U(\vec{n}^*) = T_U^{(i)}((n^*)^{(i)}) + P^{(i)} + \delta$, $\delta > 0$, for some i. Let $L = \{j \in \{1, \ldots, m\} : D_U(\vec{n}^*) = T_U^{(l)}((n^*)^{(l)}) + P^{(j)}\}$ i.e. the set of paths with maximal delay. We can always decrease $(\vec{n}^*)^{(l)}$ by $\epsilon > 0$ for $l \in L$ and increase $(n^*)^{(i)}$ by $\epsilon|L|$ so that the constraint $\sum_{j=1}^m (n^*)^{(j)} = n$ remains satisfied. Select ϵ such that $((n^*)^{(i)} + \epsilon|L|)\mu^{(i)} + w^{(i)}\sqrt{(n^*)^{(i)} + \epsilon|L|} + P^{(i)} \leq \max_{l \in L}((n^*)^{(l)} - \epsilon)\mu^{(l)} + w^{(l)}\sqrt{(n^*)^{(l)} - \epsilon} + P^{(l)}$. This is always possible since $(n^*)^{(i)}\mu^{(i)} + w^{(i)}\sqrt{(n^*)^{(i)}} + P^{(i)} + \delta = (n^*)^{(l)}\mu^{(l)} + w^{(l)}\sqrt{(n^*)^{(l)}} + P^{(j)}$, by assumption, and $(n^*)^{(l)} > 0$. Since one of $\mu^{(j)}$ and/or $w^{(j)}$ is greater than zero this change decrease the maximum path delay, yielding the desired contradiction. □

When we now restrict ourselves to sending an integer number of packets on each path essentially the same reasoning tells us that the solution to optimization (4.5) ensures approximately equal 95th percentile delay on all paths, with inequalities in delay arising due to quantization. That is, we can solve optimization (4.5) by first solving the continuous-valued optimization in Lemma 1 and then searching over the integer vectors obtained by taking the ceil or floor of each element of the continuous solution to find the integer vector minimizing D_U. Pseudo-code is given in Algorithm 1. This search is over 2^m combinations and so is fast for realistic values of m e.g. $m \leq 4$. Alternatively, when $m = 2$ then the solution to (4.5) can also be found by bisection search with time complexity of $O(\log n)$.

Estimating $E[T_k^{(j)}]$, $a^{(j)}$ and $b^{(j)}$

In our tests the mean $E[T_k^{(j)}]$, lower bound $a^{(j)}$, and upper bound $b^{(j)}$ of the inter-packet delay on path j are estimated as, respectively, the average, minimum and 95% percentile of the last 5000 packets, or all previous packets if fewer than 5000 were sent (5000 is used as it is a reasonable upper bound on object size). The inter-packet delay is calculated by

observing the time between arrival of acknowledgments at the sender. This way we allow the scheduler to be implemented without any modifications to the receiver. Note also that with this approach the link parameters are updated on every ACK, allowing the scheduler to react quickly to changes.

Receive Buffer Management

Packets are buffered at the receiver until they can be delivered in-order to the application so as to preserve TCP byte-stream semantics. This means that the receiver can become blocked when the receive buffer capacity is reached due to out-of-order delivery or outage on one or more paths. To reduce the occurrence of such blocking we size the receive window as $\sum_{j=0}^{m} \frac{D_U(\vec{n})}{\mu^{(j)}}$ by setting the socket buffer to the appropriate size for each connection. In addition, in the rare cases where blocking does occur we re-transmit the packets which fail to be delivered. Receiver buffer management can be implemented through adapting socket buffer sizes in a user-space network stack that implements SOS.[47]

Interaction with Congestion Control

In order to focus on the performance of the scheduler in isolation, in the present section and in Section 4.4 we assume that the number of packets in an object is smaller than the size of the congestion window, cwnd. Hence, object transmission is not delayed by action of the congestion controller. This assumption is relaxed in Section 4.5,.

Acquiring Object Sizes

Object information can be acquired for known protocols such as HTTP using decryption at the transport layer. If decryption is not possible, the scheduler can also consider TLS block sizes instead, since decryption cannot occur before the entire block is received, causing a TLS block to behave as a single object. The scheduler can also be placed inside the browser (similarly to QUIC) so that it has access to the object sizes before encryption occurs. For applications that use message based socket APIs [**message-based-sockets**], the object sizes would be made available to the operating system and to the multi-path scheduler.

4.3.3 Performance Evaluation

In this section we evaluate the performance of the proposed object scheduler, which we refer to as SOS (Stochastic Object-aware Scheduler) across a range of link specs and object

sizes. To provide a baseline we compare this with the performance of the Earliest Deadline First (EDF) and Stochastic Earliest Deadline Path First (SEDPF) packet schedulers, EDF being optimal when path delays are fixed and known and SEDPF being a state of the art multipath packet scheduler introduced in [22] that takes explicit account of path delay fluctuations.

Evaluation Setup

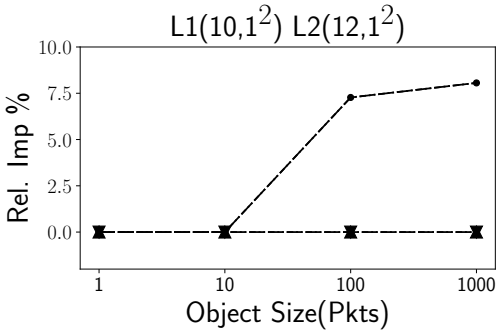
We implemented all of the schedulers within a python script and simulated the transmission of objects over two links using both synthetic packet delay data and delay data measured from real links in three different locations. Each object is transmitted and received before the next object is transmitted. As already noted, the CWND is larger than the object size. There are no losses.

Synthetic Packet Delay Data

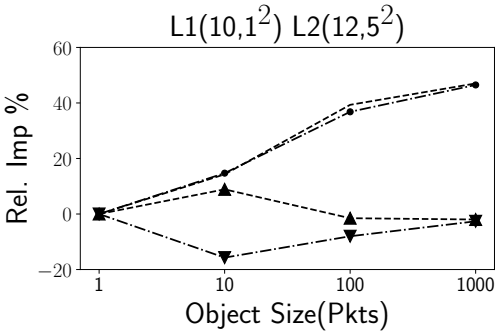
We begin by evaluating performance across two paths where the inter-packet delays are synthetically generated and follow a Gamma distribution (we note that [7, 44] observe that network delay tends to follow a Gamma distribution with a heavy tail). In the plots L1(1,σ^2) indicates that path one has mean inter-packet delay of 1ms and that the standard deviation σ is marked on the x-axis of the plot. Similarly, L2 indicates the properties of path two. The paths specs are selected to illustrate the performance both when having very stable links and highly variable links. Additionally, we alternate between the faster link being more variable and the slower link more variable. For each scheduler the mean and standard deviation of the inter-packet delay are provided to each scheduler so as to eliminate the estimation of the channel as a variable in the results.

Figure 4.5 and Figure 4.6 compares the performance of the SOS against SEDPF and EDF schedulers. In figures 4.5(b)-(f) path one has inter-packet delay with mean 10ms and standard deviation 1ms while path two has mean 12ms and a different standard deviation in each figure as is displayed above the figure. Figures 4.7(b)-(f) path one has mean 10ms and a different standard deviation in each figure as is displayed, while path two has inter-packet delay with mean 12ms and 1ms standard deviation.The object size is varied from 1 to 1000 packets as visible on the x-axis. Each scheduler transmits 10000 objects of each size with each link configuration, and the object delay is measured. The mean and the 95th percentile of object delay are then calculated for each object size and link configuration, and the relative improvement by SOS is shown

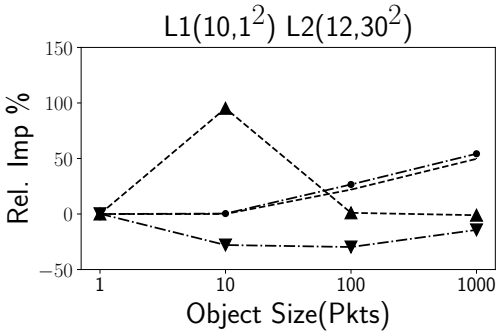
(a) Legend



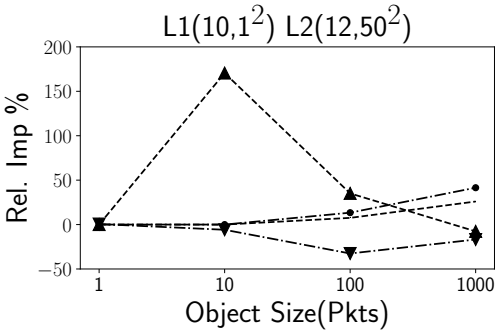
(b)



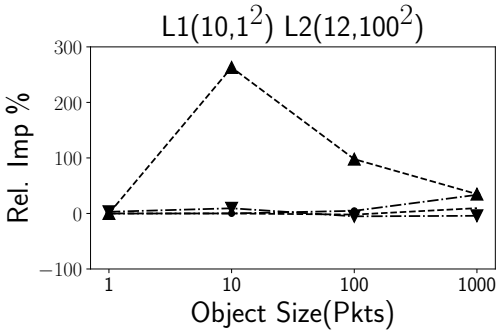
(c)



(d)



(e)



(f)

Figure 4.5: Comparing SOS against SEDPF and EDF. Plot shows relative improvement in object delay by SOS at both 95th percentile and mean over synthetic links generated using Gamma distributed inter-packet delays.

In Figures 4.5(b)-(f) and 4.6(b)-(f) and for objects consisting of a single packet, SOS and SEDPF schedulers have essentially identical performance. In this case both schedulers opt not to use the more variable link. However, it can be seen that as the object size increases, SOS begins to achieve increasingly large improvements in both the mean and 95th percentile object delay over SEDPF. What is happening is that SOS utilizes the variable link faster than SEDPF as it is aware that the larger object size will ensure less variability.

In Figures 4.5(b)-(f) and 4.6(b)-(f) SOS is equal to or better than EDF with respect to 95% percentile in almost every case. It can be seen that even for objects consisting of a single packet SOS achieves substantial improvements (of around $\times 3.5$) over EDF in the 95th percentile object delay in certain cases. The improvement is a result of SOS transmitting less packets over the variable link compared to EDF, resulting in a better 95th percentile object delay. EDF performed consistently better on mean object delay than SOS, but not by a large margin, as a result of sending data over the faster more variable link. However, this difference shrinks as the object sizes get larger.

Note that the 95th percentile is roughly an upper bound on the object delay and so is often the quantity of most interest for latency sensitive applications (video chat, gaming, tactile internet) where objects are required to be delivered within a specified deadline. Figure 4.6(b)-(d) also show large improvements in the 95th percentile object delay, although it can be seen that these come at the cost of reduced mean object delay performance. That is, SOS trades off mean object delay performance for improved worst-case performance, as expected.

In summary, by being cognizant both of path packet delay fluctuations and that it is the time to deliver objects rather than packets which is the quantity of interest, SOS offers significant gains in object delay performance over state of the art packet schedulers.

Experimental Measurements

In addition to the above evaluation using synthetic packet delays we also collected packet delay data in three different locations in Dublin, Ireland, namely a Coffeeshop (Insomnia), the Nokia Bell Labs site and the Trinity College Dublin campus. The trace contains inter-packet delay measured simultaneously over two parallel TCP connections (one for WiFi and one for LTE) using the default CUBIC congestion control in Linux, between a Linux laptop and an AWS Linux box located in Dublin, Ireland. The LTE connection was created through USB tethering to a smart-phone. The delay was measured using

Wireshark timestamps and 20000 packets were sent over each link at each location. The packets were sent back to back, and not paced. Using this trace data we then measured the performance of the SOS, EDF and SEDPF schedulers.

Figure 4.7 compares the measured 95th percentile and mean object delay performance of the SOS against SEDPF and EDF schedulers at the three measurement locations as the object size is varied. Figure 4.7(a) shows the same general trend as Figures 4.5 and 4.6, with SOS offering improvements in both the 95th percentile and mean object delay over SEDPF. Figure 4.7(c) shows similar behavior, apart from a small dip in the 95th percentile improvement for object size 100 packets. In Figure 4.7(b) the behavior is somewhat different, with the improvement in the 95th percentile object delay greatest for smaller object sizes while the improvement in the mean object delay increases with object size. This is due to a few extreme outliers in the packet delays at this location that heavily influenced the standard deviation, and resulted in SEDPF staying clear of the WiFi link, despite the fact that it had lower 95th percentile object delay. In SOS, we don't consider the standard deviation, but rather the 95th percentile object delay and the minimum, and as such it is more resilient to outliers.

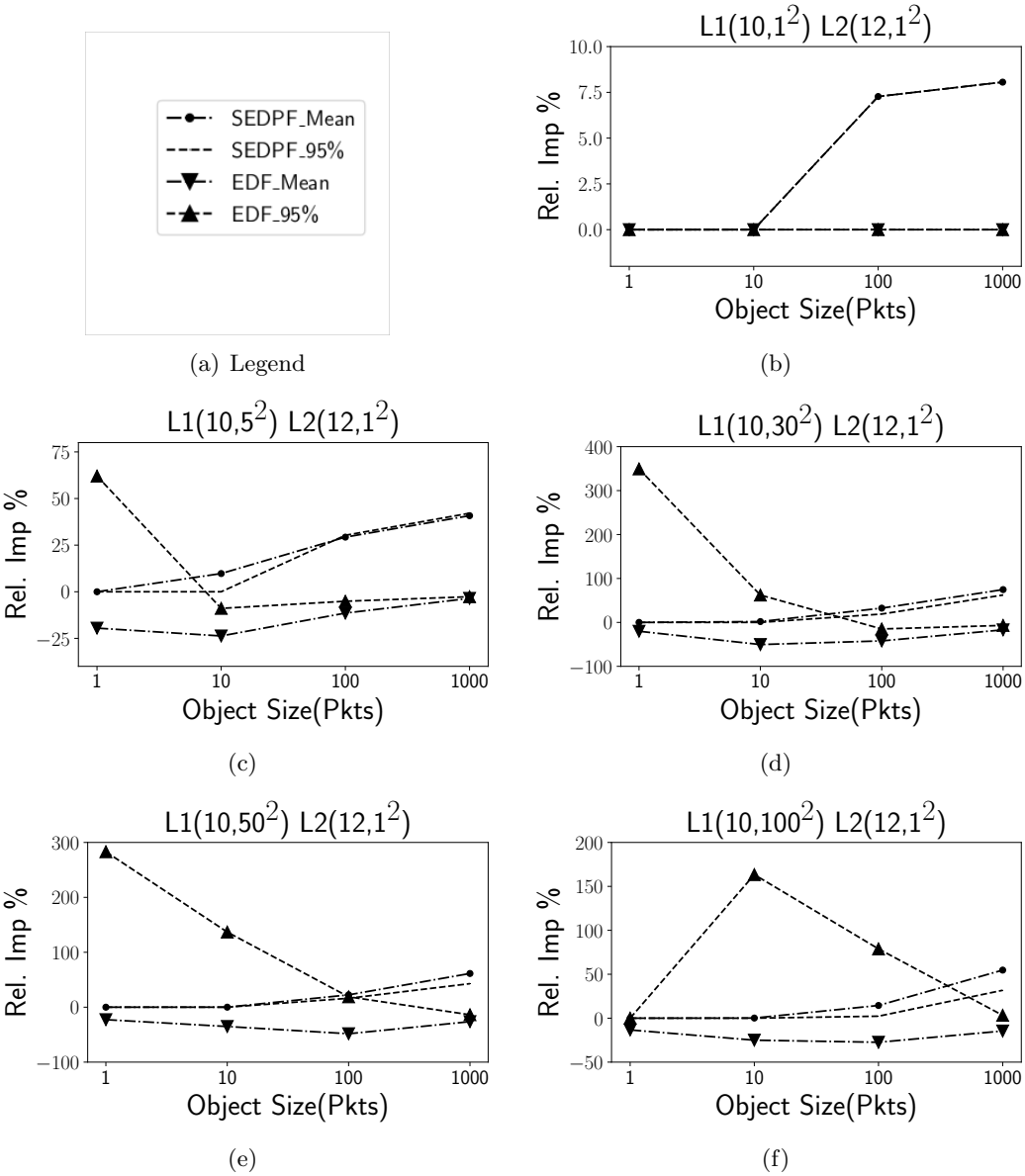
It can be also be seen that the 95th percentile object delay performance of SOS vs EDF behaves similar to that in Figure 4.6, but unlike in Figure 4.6 this does not come at the cost of reduced mean object delay performance. This is because the more fluctuating link here was on average also slower, similar to 4.5 but with an even higher discrepancy between the speeds of the two link ,resulting in minor gains in mean object delay for EDF using it more often.

4.4 Opportunistically Lowering Latency

4.4.1 Motivating Example

The SOS scheduler introduced in Section 4.3.2 allocates packet transmissions for an object amongst the available paths so as to minimise the 95th percentile object delay, thereby ensuring that an object is with high probability delivered with low delay. However, we can also sometimes be "lucky" in the sense that the packet delays actually realised on one or more paths happen to be lower than usual.

For example, suppose we have two paths with 95th percentile packet delay $T_U^{(1)}(n)$ and $T_U^{(2)}(n)$ as indicated in Figure 4.8 and the specific random packet delay realisation $T^{(2)}(n)$ for path 2 corresponding to the lower line in the figure. Suppose for simplicity also that the



(a) Legend

(b)

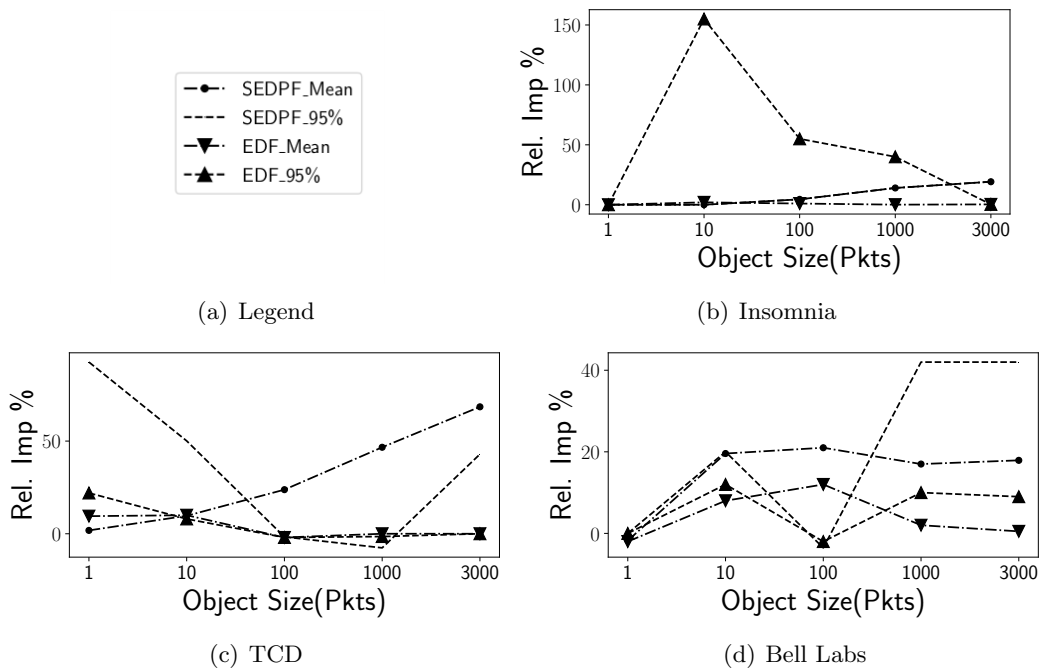
(c)

(d)

(e)

(f)

Figure 4.6: Comparing SOS against SEDPF and EDF. Plot shows relative improvement in object delay by SOS at 95th percentile and mean delay over synthetic links generated using Gamma distributed inter-packet delays.



(a) Legend (b) Insomnia

(c) TCD (d) Bell Labs

Figure 4.7: Comparing SOS against SEDPF and EDF. Plot shows relative improvement in object delay by SOS at 95th percentile and mean delay over experimentally measured packet delay data collected at 3 locations.

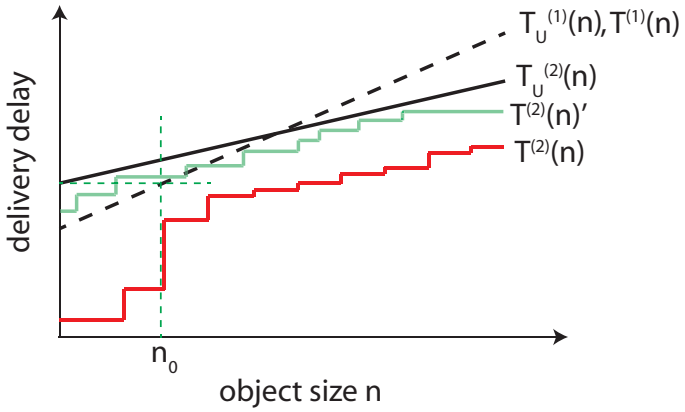


Figure 4.8: Illustrating how a "lucky" realisation $T^{(2)}(n)$ of the random packet delay on a path (the lower line) can potentially allow object delay to be reduced compared to a more "typical" realisation $T^{(2)}(n)'$.

path 1 packet delay is deterministic and so the packet delay realisation $T^{(1)}(n) = T_U^{(1)}(n)$ i.e. lies on the dashed line in Figure 4.8. For an object consisting of $n < n_0$ packets it can be seen that to minimise the 95th percentile object delay we should send all n packets on path 1. However, in this example we are lucky and happen to see a sequence of low packet delays on path 2 so that $T^{(2)}(n) < T^{(1)}(n)$ for all $n < n_0$. That is, if we had instead sent all n packets on path 2 rather than path 1 then we would have achieved a lower object delay. Note that a different realisation of the random packet delays, such as that marked $T^{(2)}(n)'$ on the figure, might lead to scheduling on path 1 yielding the lower object delay.

Of course we only discover the packet delay realization on a path *after* transmitting our packets. Nevertheless, by transmitting additional redundant (coded) packets, over

and above the minimum number n required to communicate an object, we can still opportunistically reduce the object delay by exploiting path low packet delay realizations when they occur. For example, suppose we construct as many redundant packets as required by using a random linear code[†] such that, roughly speaking, on receipt of any n coded and uncoded packets then the object can be reconstructed at the receiver. Considering again the example in Figure 4.8, we now schedule n packets on path 1 and n on path 2 i.e. $2n$ packets in total. Then we still minimise the 95th percentile object delay, as before, since we send n packets down path 1. However, by sending additional packets down path 2 then if we are lucky and experience a packet delay realization as in the figure then are able to take advantage of this to achieve a lower object delivery time. Observe that in this way we are trading off network capacity (since we send more packets) for the chance of lower object delay.

4.4.2 SOS-RED Opportunistic Scheduler

To take advantage of path realizations with low packet delay we modify the SOS scheduler from Section 4.3.2 to send $n+\delta$ packets, $\delta \geq 0$, where receipt of any n of the $n+\delta$ packets sent is sufficient to allow the object to be reconstructed at the receiver. On each path $j = 1, \ldots, m$ we send $n^{(j)} + \delta^{(j)}$ packets where $n^{(j)}$ is selected as in Section 4.3.2 and $\sum_{j=1}^{m} \delta^{(j)} = \delta$.

It remains to decide the value δ for the number of additional packets sent. Recall that the 95th-percentile delay to send $\eta^{(i)}$ packets on path i is bounded by $T_U^{(i)}(\eta^{(i)}) = \eta^{(i)}\mu^{(i)} + \sqrt{\eta^{(i)}}w^{(i)}$ where $\mu^{(i)}$ is the mean inter-packet delay and $w^{(i)}$ is a delay variation parameter. When path i has a lucky packet delay realisation then this bound is overly conservative. We can therefore capture this by rescaling the bound to $\eta^{(i)}\mu^{(i)} + \gamma\sqrt{\eta^{(i)}}w^{(i)}$ where $\gamma = 1 - \rho$, $0 \leq \rho < 1$ is a design parameter. Roughly speaking, ρ is the probability of a lucky delay sequence occurring. When $\rho = 0$ then we recover the previous setup and when ρ is close to 1 then the 95th-percentile delay bound on delay becomes close to the mean delay $\eta^{(i)}\mu^{(i)}$.

For each path $i = 1, \ldots, m$ we solve the following optimization

$$\vec{\eta}^{*} \in \arg\min_{\vec{\eta} \in \mathbb{N}^m} \quad D_U^{(i)}(\vec{\eta}) \quad s.t. \quad \sum_{j=1}^{m} \eta^{(j)} = n \tag{4.7}$$

[†]In more detail, we treat the packets forming an object as symbols in an appropriate finite field and construct coded packets as weighted linear combinations of these symbols where the weights are selected uniformly at random from the finite field.

where $D_U^{(i)}(\vec{\eta}) = \max_{j\in\{1,\ldots,m\}} T_U^{(i,j)}(\eta^{(j)}) + P^{(j)})$ and

$$T_U^{(i,j)}(\eta^{(j)}) = \begin{cases} \eta^{(i)}\mu^{(i)} + \gamma\sqrt{\eta^{(i)}}w^{(i)} & j = i \\ \eta^{(j)}\mu^{(j)} + \sqrt{\eta^{(j)}}w^{(j)} & j \neq i \end{cases} \tag{4.8}$$

We then select $n^{(i)} + \delta^{(i)} = (\eta^*)^{(i)}$. That is, we select the number $n^{(i)} + \delta^{(i)}$ of packets to send on path i to be equal to the i'th element of the optimal split vector $\vec{\eta}^*$ when the packet delay fluctuations on path i are a factor γ better than the 95th percentile. The intuition for this approach is that the γ parameter functions as a dampening factor applied to $\eta^{(i)}\mu^{(i)}$ resulting in $T_U^{(j)}(\eta^{(j)})$ expressing a more lucky realization of packet delay on that link. We refer to this scheduler as SOS-RED (since it uses redundant packets). Pseudo-code is given in Algorithm 2.

This approach has a number of appealing properties. Firstly, we note that it always selects $\delta^{(i)} \geq 0$ since $\gamma < 1$ reduces the delay $T_U^{(i,i)}$ in (4.8) compared to $T_U^{(i)}$ and this will increase the number of packets assigned to path i. Secondly, while we might generalise (4.8) to

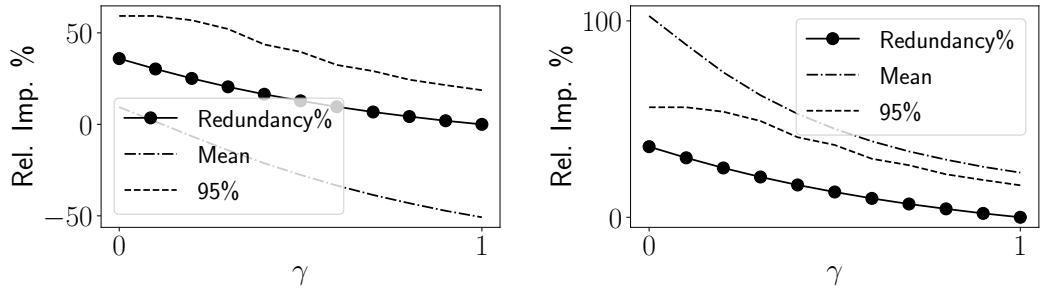
$$T_U^{(i,j)}(\eta^{(j)}) = \eta^{(j)}\mu^{(j)} + \gamma^{(i,j)}\sqrt{\eta^{(j)}}w^{(j)} \tag{4.9}$$

with $0 \leq \gamma^(i,j) \leq 1$ with a view to exploiting combinations of low packet delay realisations in multiple paths, in fact the solution to (4.8) will always perform equal to or better than the solution to (4.9) when $\gamma^{(i,j)} = \gamma$ (for consistency with (4.8)). To see this observe that selecting $\gamma^{(i,j)} < 1$ for paths $j \neq i$ will tend to increase the number of packets assigned to path j. Since the total number of packets $\sum_{j=1}^{m} \eta^{(j)} = n$ in (4.7), this means that the number of packets assigned to path i will tend to decrease. Hence, (4.8) yields packet allocation to each path which is at least as large as would be assigned by (4.9).

4.4.3 Performance Evaluation

Evaluation Setup

We utilized the same simulation setup used previously for evaluating the SOS scheduler in Section 4.3.3



(a) EDF (b) SEDPF

Figure 4.9: Comparing SOS-RED against EDF and SEDPF: plot shows relative improvement in object delay at 95% percentile and mean, and also the amount of redundancy δ, vs the γ parameter. The inter-packet delay of the two paths is Gamma distributed with means $\mu_1 = 10ms, \mu_2 = 12ms$ and standard deviations $\sigma_1 = 50ms, \sigma_2 = 1ms$ The object size is 100 packets.

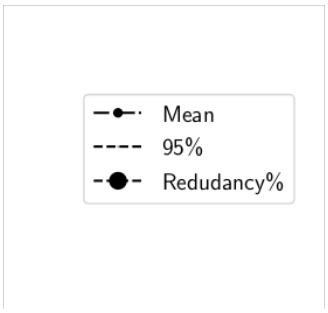
Varying the Gamma parameter

In Figure 4.9, we measure the performance of the SOS-RED scheduler using a range of values for the γ parameter. The inter-packet delay distribution for the two links and the object size is kept static across the tests. As expected the amount of redundancy is reduced as the value of γ goes to 1, where the amount of redundancy becomes 0 at $\gamma = 1$. There's more significant improvement in the mean object delay compared to the 95th% percentile, as the mean improvement increases by 75% vs SEDPF and 50% vs EDF across the range of γ values used, while the 95% percentile improvement is approximately 25% and 35% vs EDF and SEDPF respectively. This is because the scheduling decisions of SOS are already optimized for the 95% percentile and as such the improvement due to FEC would mostly arise if the instances of "unlucky" realizations of packet delay on one path occur simultaneously with the "lucky" realizations on the other paths. The value of the redundancy δ increased by roughly 35% as the value of the γ parameter varied from 1 to 0.

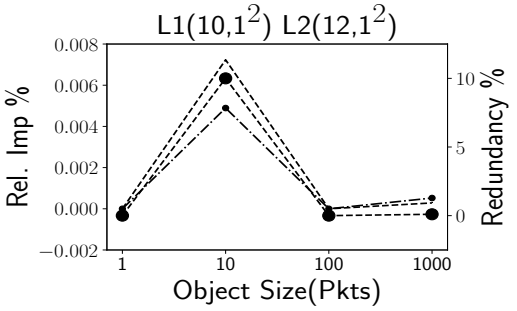
Synthetic Delay Data

As before, we begin by evaluating performance using synthetic Gamma distributed inter-packet delay data.

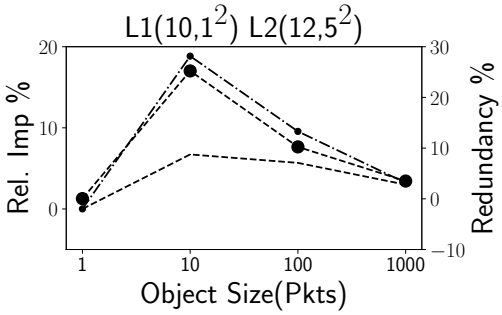
Figures 4.10 and 4.11 compares the object delay performance of SOS-RED and SOS for a range of path conditions and object sizes. We observe that the 95% percentile object delay performance of SOS-RED offers no improvement over SOS for single packet objects and a small improvement for larger objects, the maximum being 20%. This is expected because SOS sacrifices mean object delay for maintaining the 95% percentile. Additionally, we can see significant improvement in mean object delay delay, especially for single packet objects, peaking at around 600%. Of course, this comes at the cost of



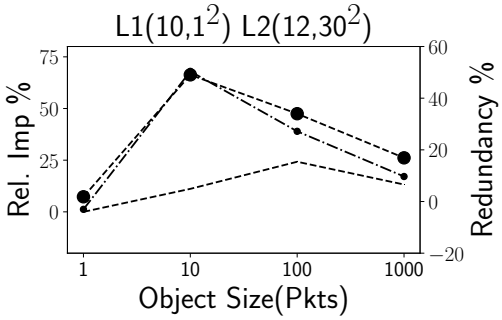
(a) Legend



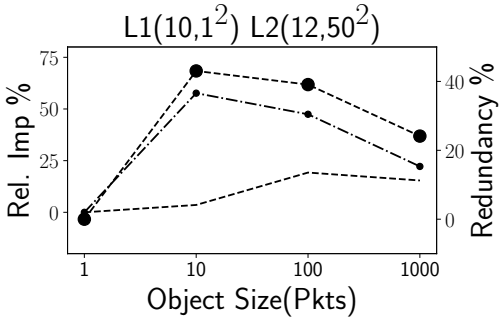
(b)



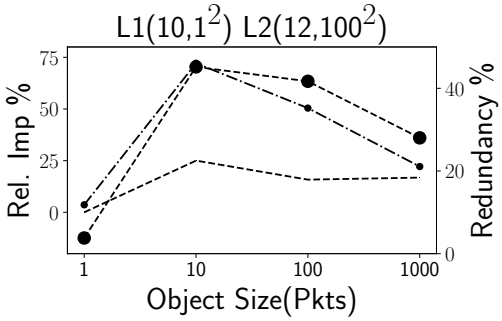
(c)



(d)

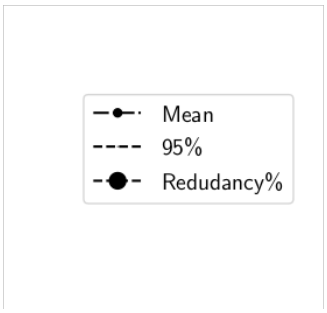


(e)

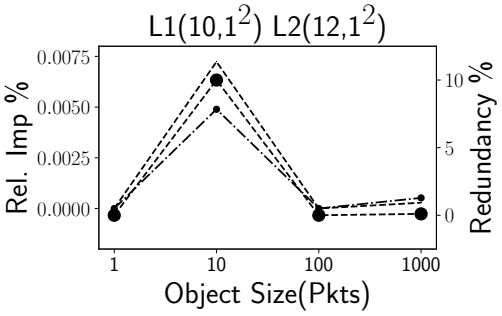


(f)

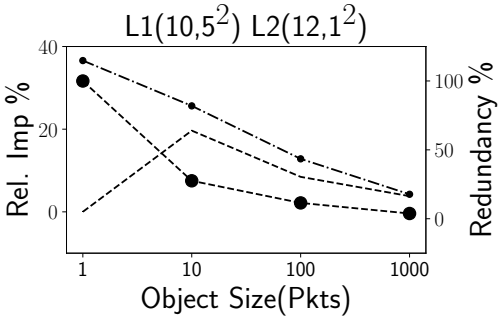
Figure 4.10: Comparing SOS-RED against SOS. Plot shows relative improvement in object delay by SOS-RED at mean and 95th percentile, and also the level of redundancy δ.



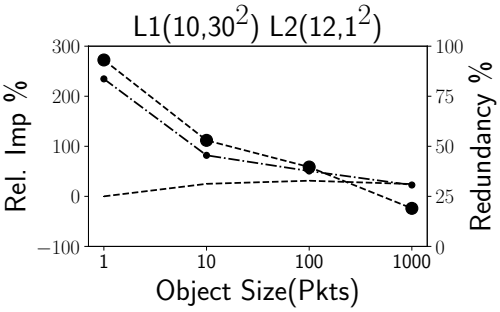
(a) Legend



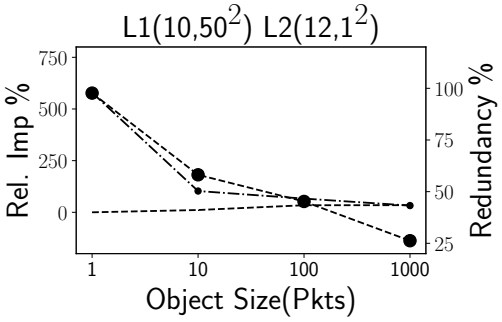
(b)



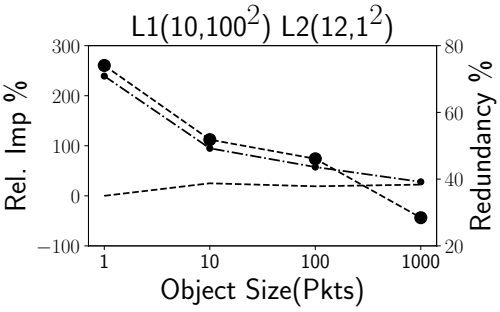
(c)



(d)

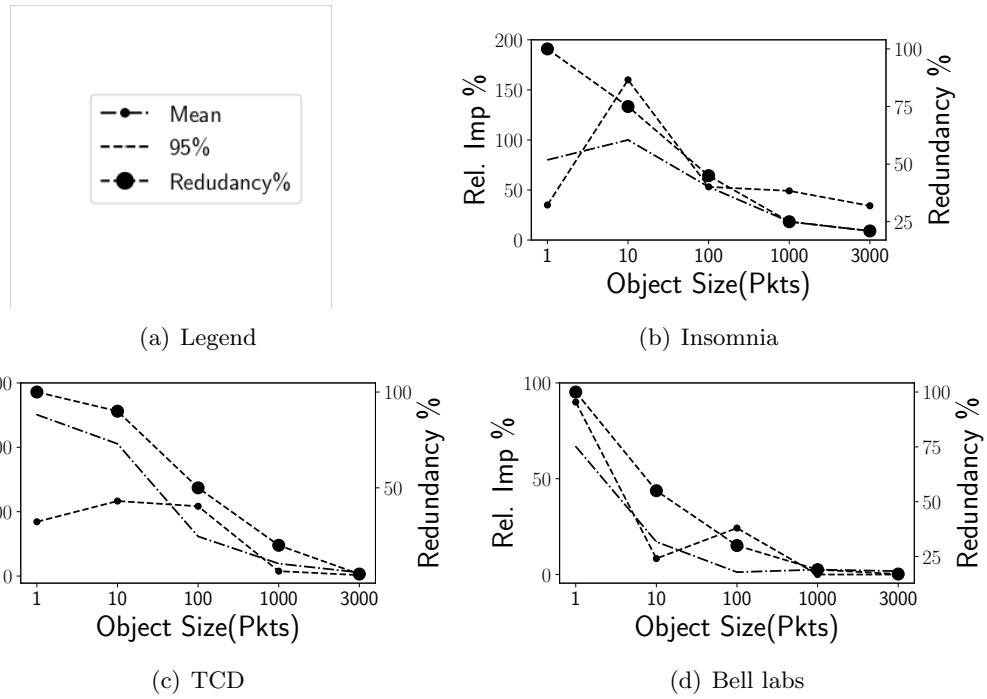


(e)



(f)

Figure 4.11: Comparing SOS-RED against SOS. Plot shows relative improvement in object delay by SOS-RED at mean and 95th percentile, and also the level of redundancy δ.



(a) Legend

(b) Insomnia

(c) TCD

(d) Bell labs

Figure 4.12: Comparing SOS-RED against SOS. Plot shows relative improvement in object delay by SOS-RED at 95th percentile and mean, and the level of redundancy, over experimentally measured packet delay data collected at 3 locations.

added redundancy, in this case sending roughly two packets instead of a single packet for each object. It can also be seen that the amount of redundancy used by SOS-RED scales with the path packet delay variability, as might be expected, and also tends to decrease with increasing object size.

Experimental Measurements

Figure 4.12 compares the object delay performance of SOS-RED and SOS for packet delay traces measured at three locations in Dublin, Ireland i.e. the same packet delay measurements as used in Section 4.3.3. It can be seen that, compared to SOS, SOS-RED achieves a larger improvement in both mean and 95th percentile object delay for smaller object sizes. This is because for smaller object sizes SOS tends to stay clear of variable paths even when they have significantly lower mean packet delay whereas SOS-RED is able to utilize those paths without compromising the 95th percentile object delay. Similarly to the synthetic data, the redundancy decreases as the object size increases, until it levels off at 3000 packet objects where there's almost no redundancy.

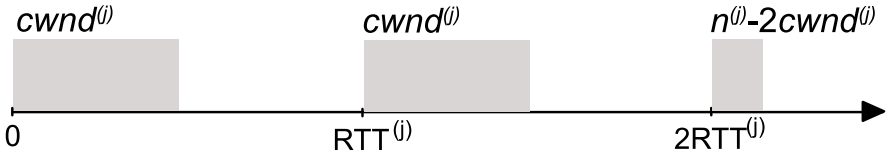


Figure 4.13: Illustrating extra packet delay introduced by cwnd-based congestion control. Since only $cwnd^{(j)}$ packets can be sent per RTT on link j is less than the number $n^{(j)}$ of packets to be sent then it takes multiple RTTs to send the packets.

4.5 Interfacing SOS with Cwnd-Based Congestion Control

In the previous sections we assumed that the $n^{(j)}$ packets of an object partition sent over link j are committed to the link at once, implying that number of packets that can be in-flight (the cwnd) is bigger than or equal to $n^{(j)}$. However, when SOS is used with cwnd-based congestion control then the congestion control may limit the number of in-flight packets to be less than $n^{(j)}$. This can introduce extra packet delay as the $n^{(j)}$ packets will then need to be split across multiple cwnds, and so multiple RTTs. In order to accurately estimate the object partition delay when running on top of cwnd-based congesiton control SOS therefore has to explicitly consider the value of cwnd when making scheduling decisions. In this section we implement a cwnd-aware version of SOS using a user-space network stack. We compare its performance to that of MPTCP/minRTT across a range of network conditions and object sizes and measure the improvement in the 95% percentile and the mean delay of each object. Due to issues of space and implementation details, we leave the evaluation of SOS against other congestion-control aware algorithms to future work.

4.5.1 Cwnd-aware SOS

Delay Estimation

Cwnd-based congestion control restricts the number of packets that can be sent across a link to cwnd packets per RTT. Hence, when the cwnd $cwnd^{(j)}$ on link j is less that the number $n^{(j)}$ of packets to be sent over link j then it takes multiple RTTs to send the $n^{(j)}$ packets, see Figure 4.13. Denoting the cwnd on link j by $cwnd^{(j)}$ and the RTT by $rtt^{(j)}$ and assuming that $cwnd^{(j)}$ and $rtt^{(j)}$ stay roughly constant over the time taken to send an object, then the time taken to send $n^{(j)}$ packets over the link is

$$T^{(j)}(n^{(j)}) = N^{(j)} rtt^{(j)} + T^{(j)}(n^{(j)} - N^{(j)} cwnd^{(j)}) \tag{4.10}$$

where $N^{(j)} = \lfloor n^{(j)}/cwnd^{(j)} \rfloor$ and $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ is the time taken to transmit the $n^{(j)} - N^{(j)}cwnd^{(j)}$ packets which occupy only part of the link j cwnd. We accommodate fluctuations in the round-trip time and $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ as follows. Firstly, assuming that fluctuations in the round-trip time are i.i.d. then we can replace $N^{(j)}rtt^{(j)}$ by $\sum_{i=1}^{N^{(j)}} RTT_i^{(j)}$ where $RTT_i^{(j)}$ is a random variable equal to the round-trip time during the i'th cwnd taken to transmit the $n^{(j)}$ packets. We can then bound this value using the Chernoff-Hoefdding bound similarly to eqn (5.4.3). Secondly, by recording the transmission time of a packet, the number of packets in-flight when it was transmitted, and the time the acknowledgement for this packet was received we can estimate the distribution of inter-packet delay $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$. We remove old samples as new ones arrive in order to keep the estimate up to date and to prevent noise from skewing the system for long. Using this inter-packet delay distribution we can bound $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ at the same level of reliability ϵ as the Chernoff-Hoefdding bound. In this way we can upper bound the inter-packet delay using

$$\begin{aligned} T_U^{(j)}(n^{(j)}) = N^{(j)}E[rtt^{(j)}] + w^{(j)}\sqrt{N^{(j)}} \\ + T_U^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)}) \end{aligned} \tag{4.11}$$

where $w^{(j)}$ is as in eqn (4.3) with $a^{(j)} \leq RTT^{(j)} \leq b^{(j)}$, and $T_U^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ is the estimated upper bound on $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$.

Re-Scheduling

Since transmission of the $n^{(j)}$ packets over link j proceeds in $N^{(j)}$ rounds each of cwnd packets, the opportunity exists to re-evaluate our schedule periodically and adjust the value of $n^{(j)}$ (so long as we do not reduce it below the number of packets already transmitted). In our tests we re-evaluate the schedule upon the receipt of an ack from any link, as this presents a new transmission opportunity. It is however possible to perform more sophisticated re-evaluation of the schedule at higher granularity based on the expected arrival time of an ack.

4.5.2 Performance Evaluation

Experimental Setup

We transmitted objects using two python scripts, one acting as a client sending a request for an object and the other acting as a server sending the object in response to a request.

The object delay was calculated as the time between sending the request for an object and receiving the full object from the server. We allocated the maximum size of receive and send buffers for both the client and the server sockets in order to prevent the MPTCP penalization mechanism from under-utilizing a link. The server transmitted five different object sizes, namely 1, 100, 1000, 3000 and 10000 packets. Objects of each size being sent repeatedly over a long lived connection for 5 minutes. The reason we used one long-lived connection is to minimize the impact of the capacity estimation on the performance of the schedulers.

We used the Linux `tc` tool with its extension `netem` to emulate an LTE and a WiFi link on top of two 1Gbps ethernet cables. The first "LTE" link was configured to have static packet delay of 40ms and a rate of 40Mbps. The second "WiFi" link was configured to have truncated Gaussian-distributed delay with 10ms mean and standard deviation of 0, 10, 50, 100 and 200ms[‡].

As a baseline for comparison we used MPTCP/minRTT since, to the best of our knowledge, it is the only multipath scheduler plus congestion control that is widely used in production networks. The version of MPTCP used is that for the Linux kernel 4.9, with default configuration parameters and Reno congestion control.

We implemented Cwnd-aware SOS in a user-space network stack.

Results

Figure 4.14(a) present measurements of the object delay performance improvement of SOS over MPTCP/MinRTT when transmitting a single packet. We observe that MPTCP/MinRTT didn't always utilize the faster link when the standard deviation of the inter-packet delay is low, occasionally transmitting packets through the slower link which caused both the mean and 95% transmission object delay to drop relative to that of SOS. However, for higher levels of inter-packet delay variance MPTCP/minRTT and SOS have similar object delay performance (MPTCP/MinRTT being slightly better since the way SOS was implemented added a certain amount of fixed packet delay due to the usage of a virtual interface).

Figure 4.14(b) shows the performance when transmitting 100 packets. When the faster link has fixed inter-packet delay, MPTCP/minRTT sends packets across the slower link leading to higher than necessary object delay. As the level of inter-packet delay variance

[‡]This is generated by the `netem` tool. The inter-packet delay is constrained to be non-negative, and for positive values the inter-packet delay is derived from a lookup table and so has also fixed upper bound. Packet ordering is preserved.

increases MPTCP/minRTT always sends some packets across the faster fluctuating link, although the number sent falls as the variance increases, causing the object delay to increase as level of variance is increased.

In Figure 4.14(c) we show measurements for the transmission of 1000 packet objects. It can be seen that SOS initially outperforms MPTCP/minRTT but as the inter-packet delay variance increases the difference in performance reduces. This is because SOS reduces its usage of the fluctuating link more quickly than does MPTCP/minRTT, but eventually both schedulers avoid using the fluctuating link.

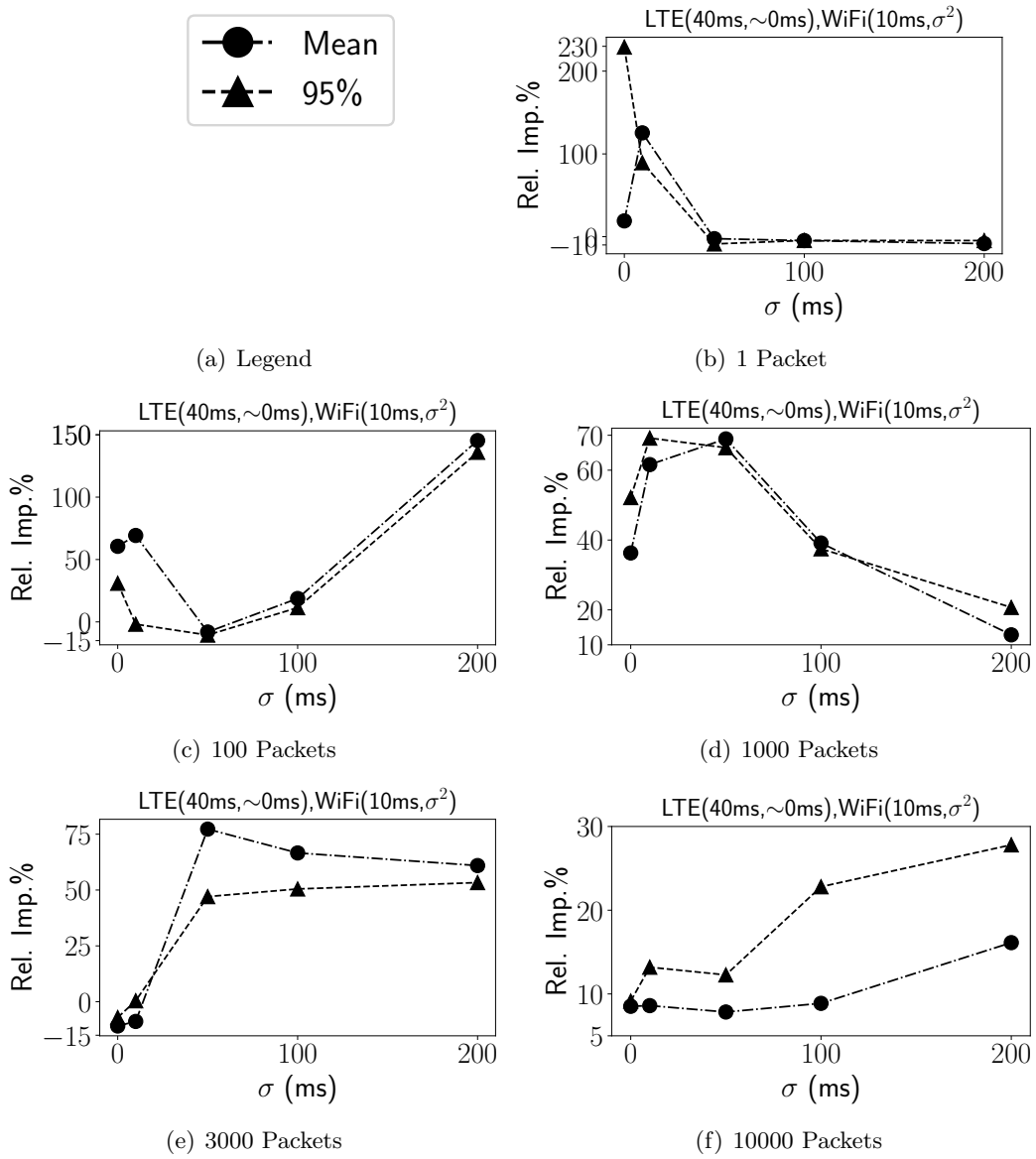
In Figure 4.14(d), for 3000 packet objects, it can be seen that initially SOS and MPTCP/minRTT perform similarly as the object spans multiple cwnds which allows time for MPTCP/minRTT to adapt, but as the inter-packet delay variance increases, the relative performance of MPTCP/minRTT degrades. In Figure 4.14(e), for 10000 packet objects, it can be seen that SOS consistently outperforms MPTCP/minRTT. However, the difference in performance is reduced compared to smaller object sizes since the difference in inter-packet delay variance doesn't cause a substantial difference in the object delay since the inter-packet delay variance is small compared to the overall transmission time.

It can be seen that the performance of SOS relative to MTCP/minRTT exhibits quite complex behaviour. The observed fluctuations are not statistical effects or noise but rather reflect repeatable algorithmic behaviour of MPTCP. We note that the MPTCP scheduler which makes use of a number of interacting heuristics, and its performance has been the subject of a number of studies.

In summary, across a range of network conditions and object sizes our measurements indicate that SOS consistently outperforms MPTCP/minRTT.

4.6 Summary and Conclusions

In this chapter we consider the task of scheduling packet transmissions amongst multiple paths with uncertain, time-varying packet delay. We make the observation that the requirement is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the packet delay which is important. This has fundamental implications for multipath scheduler design. We introduce SOS (Stochastic Object-aware Scheduler), the first multipath scheduler that considers application layer object sizes and their relationship to path uncertainty. We demonstrate that SOS reduces the 95% percentile object delay by 50-100% over production WiFi and



(a) Legend

(b) 1 Packet

(c) 100 Packets

(d) 1000 Packets

(e) 3000 Packets

(f) 10000 Packets

Figure 4.14: Comparing performance of SOS against MPTCP/minRTT over two emulated paths, one WiFi and the other LTE. Plot shows relative improvement in object delay by SOS at 95th percentile and mean over 5 different object sizes while varying the standard deviation of the inter-packet delay on the WiFi path.

LTE links compared to state-of-the art schedulers. We extend SOS to utilize FEC and show how it can opportunistically lower mean object delay without sacrificing the 95% percentile object delay and while utilizing minimal redundancy. We also show how SOS can be adapted to take account of cwnd-based congestion control and present performance measurements using MPTCP/minRTT as a baseline. These measurements show that across a range of network conditions and object sizes SOS consistently outperforms MPTCP/minRTT.

4.7 Appendix

4.7.1 SOS

Pseudo-code (See Algorithm 1)

Algorithm 1 SOS

function CBound(p,ϵ,t,γ) **return** $p * t.avg + \gamma\sqrt{p}\sqrt{\frac{ln(1/\epsilon)(t.max-t.min)^2}{2}}$
end function
function FindOptimalSplit(n,m,T,O,ϵ,$\Gamma = [1] \times m$)
    $P \leftarrow$ *generateIntegerPartitions(n,m)*
    $D_{min} \leftarrow$ *-1*
    for p in P **do**
        for $i = 0$ to m **do**
            if p[i] != 0 **then**
                $L[i] \leftarrow$ CBound($p[i]$, $\frac{\epsilon}{m}$, $T[i]$) + $O[i]$,$\Gamma[i]$)
            else
                $L[i] \leftarrow 0$
            end if
        end for
        $D \leftarrow max(L)$
        if $Dmin = -1$ or $Dmin > D$ **then**
            $Dmin \leftarrow D$
            $s \leftarrow p$
        end if
    end forreturn s
end function

Explanation

The CBound function takes the following parameters.

1. p: number of packets scheduled on a link
2. ϵ: reliability parameter

3. t: data structure containg the average, max and min inter-packet inter-delay on that link

and returns a bound on the $(1-\epsilon)$-percentile delay for transmitting p packets across the link with t delay profile.

The FindOptimalSplit function takes the following parameters.

1. n: number of packets in an object
2. m: number of links
3. T: array containing link profiles, in particular each element in the array contains the average, max and min inter-packet delay on that link
4. O: array containing propagation delay for each link
5. ϵ: the reliability parameter

It generates all integer partitions P of size m that sum up to n, representing all the possible ways of scheduling n packets across m links. For each partition, the array of delay bounds L is calculated for each link using the CBound function. The object delay is then calcuated by taking the maximum per-link delay in L. The minimum is stored and remembered, and the partition that minimises object delay is returned at the end of the function.

4.7.2 SOS-RED

Pseudo-code (See Algorithm 2)

Algorithm 2 SOS-RED

```
1: function FindFecSplit(n,m,T,ϵ,O,γ)
2:     Γ ← [1] × m
3:     for i = 0 to m do
4:         Γ[i] ← γ
5:         A[i] ← FindOptimalSplit(n, m, T, O, ϵ, Γ)
6:         Γ[i] ← 1
7:     end for
8:     for i = 0 to m do
9:         F[i] ← 0
10:        for j = 0 to m do
11:            F[i] ← max(A[j][i], F[i])
12:        end for
13:    end forreturn F
14: end function
```

Explanation

The FindFecSplit function takes the following parameters.

1. n: number of packets in an object
2. m: number of links
3. T: array containing link profiles, in particular each element in the array contains the average, max and min inter-packet delay on that link
4. O: array containing propagation delay for each link ϵ: the reliability parameter
5. γ value of gamma parameter.

It iterates through all links, assigning the γ modifier of the current link to the parameter received and all other links to 1, and finds the optimal split $A[i]$ under those conditions. It then returns the final FEC split F that assigns each link the maximum number of packets sent on that link during all iterations.

Chapter 5

BOOST: Transport-Layer Multi-Connectivity Solution for Multi-WAN Routers

5.1 Introduction

Multi-WAN Routers (MWR) provide WAN/Internet connectivity by combining two or more backhaul links (e.g. 2 LTE connections) for improved capacity and reliability. They are commonly used in public transportation services, such as trains, buses, ships and even airplanes to provide seamless Internet connectivity to a large number of passengers using wireless backhauls, which are known to deteriorate under high mobility. For example, providers of train-to-ground (T2G) communication combine LTE/5G from different carriers to improve the capacity for on-board internet access and increase reliability for mission-critical applications such as communication based train control (CBTC) or closed circuit TV (CCTV) [93].

MPTCP is used by some commercial MWRs as a transport layer solution for allowing individual TCP connections from user devices to utilize the capacity of all the available backhauls [62]. An MPTCP MWR needs to either transparently split incoming TCP connections into a TCP and an MPTCP portion, like a proxy, or encapsulate them inside MPTCP, like a tunnel. In both cases, a network-edge proxy-server/tunnel-server is used to translate/decapsulate the traffic back into TCP to be forwarded to the content server. Figures 5.1 and 5.2 show the architecture and network topology for the proxy and tunnel variants respectively.

In the proxy variant of the MWR, shown in Figure 5.1, each user TCP connection is

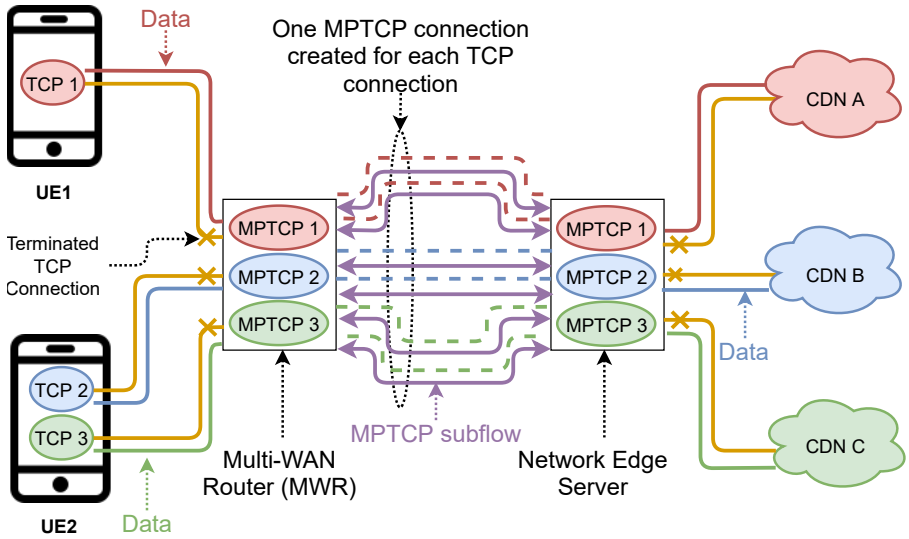


Figure 5.1: Proxy based MPTCP MWR

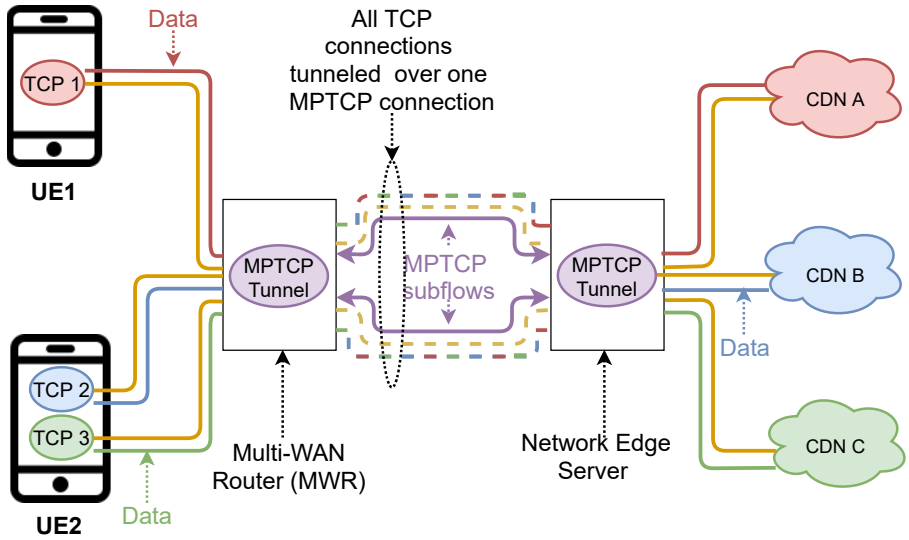


Figure 5.2: Tunnel based MPTCP MWR

terminated at the MWR and its packets are transmitted using MTCP to the network-edge proxy. Thus for each TCP connection, a corresponding MPTCP connection is created. Our measurements of passenger traffic in a train to ground (T2G) MWR show that this results in thousands of concurrent MPTCP connections, each with their independent congestion control on each link plus their own multi-path scheduler, and we find that this lead to low capacity utilization and high loss, as will be explained in more detail later.

In the tunnel variant of the MWR, shown in Figure 5.2, the packets from user TCP connections are encapsulated and transmitted over all the available backhauls using a single persistent MPTCP connection. That is, a single MPTCP connection carries all of the user TCP connections over the backhaul. This eliminates the large number of connections observed with proxy variant but creates new issues. In particular, the tunnel stacks two reliability layers (MPTCP and TCP), each with its own congestion control and

retransmission mechanism for lost packets. This can lead to so-called TCP meltdown [12], where a high number of spurious re-transmissions are constantly triggered. Additionally, there is cross-flow Head of Line (HoL) blocking between separate user flows sharing the same MPTCP connection.

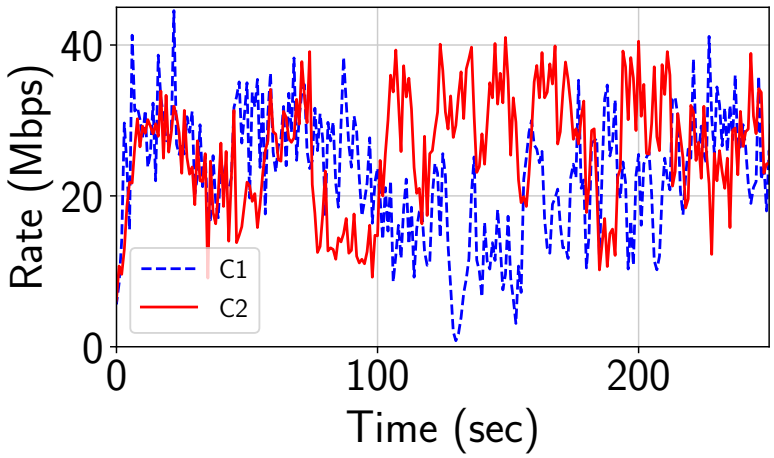
A second key issue that is common to both architectural variants is frequent HoL blocking due to the heterogeneity of the wireless links used for multi-path scheduling. This type of blocking occurs when packets arrive out of order, leading to them being placed into a reordering queue until they can be delivered in-order, increasing delay. This delay disproportionately impacts short flows (e.g. web browsing) as they are much more latency-sensitive compared to long flows (e.g. file download). HoL blocking is also more prevalent when scheduling over heterogeneous links with different delay profiles, which are commonly used in MWRs. For example, a T2G MWR tends to utilize multiple LTE links from different carriers, which are likely to have different delay profiles due to different coverage, cross-traffic and core-latency.

Based on our analysis of the MPTCP proxy and tunnel issues in MWR, we propose a new multi-path proxy solution more suited for MWR, called BOOST. This solution eliminates the problems with both the MPTCP proxy and tunnel approaches by multiplexing passenger traffic over a single, persistent and selectively-reliable multi-path connection, allowing UDP as well as TCP to benefit from multi-connectivity. It splits TCP connections, similarly to a proxy, but transmits the traffic over the backhaul using a single BOOST connection, similarly to a tunnel. BOOST is based on QUIC and re-uses many of its streaming features for flow multiplexing, while adding new features for multi-path transmission and selective reliability. BOOST also takes a hybrid approach to multi-path scheduling. Namely, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity, thereby improving efficiency and robustness. The scheduler also provides prioritization and redundant multi-path scheduling for high priority traffic.

Concretely, we make the following contributions

- We collect hours of traces from a production T2G communication system that utilizes an MPTCP proxy MWR and use this to uncover performance issues with the MPTCP proxy approach. We also highlight issues with MPTCP tunnel in light of previous work and our own trace-driven emulation results.

- Based on the outlined key issues with MPTCP, we propose BOOST, a novel multi-connectivity solution that addresses the issue present in both variants of the MPTCP



(a) Link capacity over time

Figure 5.3: (a) shows estimated capacity over time for two different LTE carriers (C1 and C2) used by a MWR in a production Train-to-Ground communication system. Capacity drop of C1 at the 120th second likely due to a handover

architecture. We evaluate BOOST performance against both an MPTCP proxy and tunnel, and against single-path TCP in a trace-driven emulation setup based on real LTE link traces and real passenger traffic traces. The results of this evaluation show that BOOST significantly improves average passenger flow throughput and reduced CCTV delay while experiencing much lower loss and out of order delivery.

5.2 Measurements collection

We collected hours of packet traces from a production Train-to-Ground (T2G) communication system that uses an MPTCP proxy MWR with two different public LTE connections as a backhaul. The system used a transparent Shadowsocks[87] proxy client inside an MWR that is running an MPTCP kernel, and a corresponding Shadowsocks server placed in a network-edge server that was also running an MPTCP kernel. The MWR was configured as the default gateway for the on-board WiFi system which provided free internet for all passengers. TCP connections entering the MWR were redirected to the Shadowsocks proxy. Packet traces were collected from the network-edge server and the MWR. The setup was used to measure capacity for both LTE links as well, collect the passenger traffic traces, and probe internal MPTCP state variable, all of which will be used for analysis and evaluation in later sections.

5.2.1 Estimating capacity

Since most passenger traffic was downlink, our interest is to measure the downlink capacity. Since the passenger traffic may underflow the capacity of the links due to lack of traffic or the action of the MPTCP congestion control, simply observing the downlink throughput to the MWR would provide inaccurate capacity estimates. To address this, two parallel UDP iPerf[41] flows, each assigned to one LTE link, were transmitted from the network-edge server towards the MWR. The goal was to ensure that there was a constant bit rate stream of data flowing through both links at all times. The choice of UDP eliminates the possibility of TCP congestion control underflowing the link capacity. The iperf throughput was limited to 2mbits, which is below the capacity of the typical LTE link, to not overload the links and disturb the passenger traffic,

Capacity is estimated by analyzing the iPerf packet trains. iPerf transmits a train of packets periodically, depending on the assigned rate, and injects the same timestamp into all packets belonging to the same train, allowing the receiver to identify a packet train. The difference between the arrival time of the first and last packets in each train as observed by the receiver, and the size of the train, was is to estimate of the capacity at the time of transmission of the train. This estimate is then used to extrapolate the capacity until the arrival of the first packet in the next train.

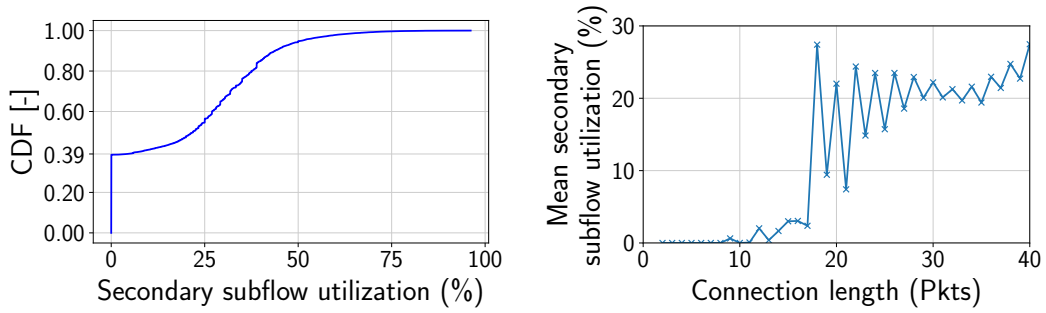
More formally, let c_i represent the capacity estimate between the arrival time of the first packet in train i, and the firs packet in following train. c_i is calculated as follows

$$c_i = \frac{\sum_{j=0}^{n_i} s_i^j}{t_i^{last} - t_i^{first}} \tag{5.1}$$

where t_i^{first} and t_i^{last} are the arrival times of the first and last packet of train i respectively, s_j^j is the length of the j th packet in the train, and n_i is the total number of packets in the train. In cases where $t_i^{last} = t_i^{first}$ i.e. the first and last packets in a packet train arrived at the same time, that packet train is ignored for capacity calculation and the previous capacity estimate is re-used. Figure 5.3(a) shows the computed capacity for the two LTE links over 250 seconds.

5.2.2 MPTCP kernel probe

To monitor MPTCP internal state, we created a kernel probe and deployed it in an MPTCP kernel running on the network-edge server. The probe is based on the Linux Kernel FTrace interface [21], and collects events and state variables in the MPTCP stack.



(a) (b)

Figure 5.4: Fig. (a) shows the CDF the ratio of utilization of link 1 (L1) and link 2 (L2). Fig (b) shows the utilization ratio vs flow length. Fig (c) shows the CDF of of flow length. Fig (d) shows the utilization ratio vs mean burst size. Fig (e) shows the CDF of time it took the second subflow to be established.

In particular, we modified the kernel to add events to the TCP FTRACE interface that track MPTCP penalization events, transmission events, receive events and re-transmission events. The probe also recorded changes in SRTT, CWND and the size of the MPTCP meta re-ordering buffer.

5.3 Analysis of the performance of MPTCP Proxy MWR

As briefly discussed in the introduction, an MPTCP proxy MWR variant establishes one MPTCP connection for each TCP connections, as shown in Figure 5.1. According to measurements of passenger traffic collected in section 5.2, this results in the creation of thousands of uncoordinated MPTCP connections, most of which are short or on/off (i.e. long-lived with low frequency of data exchange such as HTTP connections with keep-alive). In this section we analyze the performance of the proxy variant in terms of link utilization, multi-path scheduling and congestion control.

5.3.1 Link utilization

For the MPTCP default scheduler, MinRTT, to utilize more than one subflow, it first needs to establish the default subflow using the standard TCP 3-way handshake and then establish the secondary subflows by performing a similair handshake over the other available paths, which takes at least one round trip time. Before the second subflow is established, only the default subflow is used for data transmission. Once the secondary subflow is established, the connection will have to fill the CWND of the subflow with the lowest estimate for smooth round trip time (SRTT) before using other subflows.

Short connections that transmit less than 10 packets will never use the secondary subflow since all their packets will fit in the initial CWND [15] of the default subflow.

Also connections that transmit more than 10 packets may still not utilize any secondary subflow as they may transmit multiple CWND worth of data on the default subflow before the secondary subflows are established. Figure 5.4(b) shows the link utilization ratio versus the flow length It can be seen that flows with a length smaller than 18 almost never utilized the secondary subflow. Beyond reducing utilization, since the first subflow is always established over the default route configured in the operating system routing table, this will cause the performance of short connections to be highly sensitive to the choice of default route.

On/off connections that periodically transmit small bursts of data, such as HTTP connection with keep-alive, may also never make use of more than one link if they do not have enough data in flight at once to completely fill the CWND of the link with the lowest SRTT. Figures 5.5(a) and 5.5(a) show two connections that were transmitting data during the same time period but were only making use of one link each.

Figure 5.4(a) shows the CDF of secondary subflow utilization based on the measurements from the production T2G system as described in section 5.2. It shows that roughly 40% of MPTCP connections did not use the secondary subflow.

5.3.2 Multi-path Scheduling

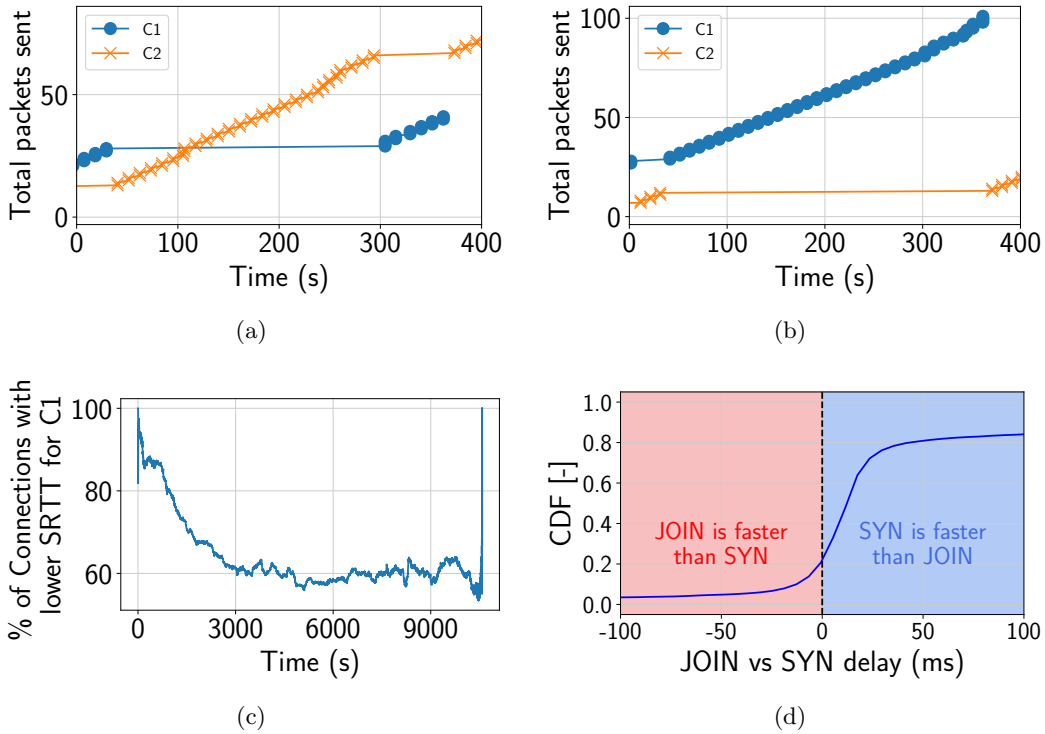


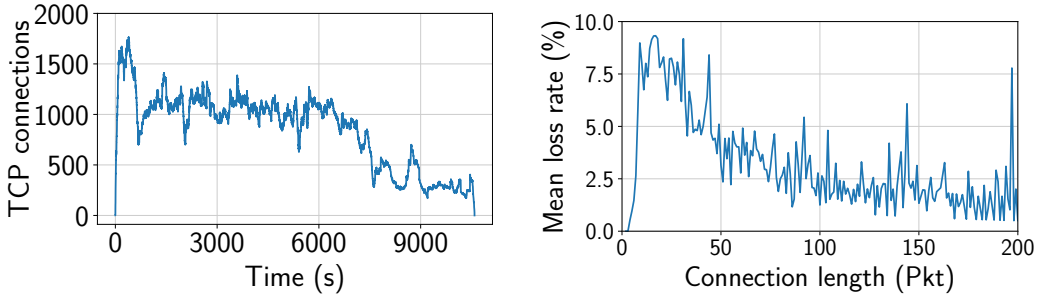
Figure 5.5: (a)(b) show the total packets transmitted from both LTE links (C1 and C2) during the same time period from 2 different MPTCP connections. (c) shows the proportion of connections that with a lower SRTT estimate for C1 during a train trip. (d) shows the CDF for the difference in delay between the JOIN packet and the SYN packet in the same connection.

MinRTT relies on data transmission for estimating the SRTT of its subflows which is needed for optimal scheduling. Every time the underlying subflow transmits a packet and receives its acknowledgement, it will calculate the RTT for that packet and use it to update the estimate of the SRTT. In an MPTCP proxy MWR, all connections will perform isolated estimations of the links and will not share information between each other.

On/off connections that do not have enough traffic to probe all links may transmit data on a slower route for a long period of time. Figure 5.5(a) and 5.5(b) show the packets transmitted on LTE links from two carriers (C1 and C2) during the same time period by two MPTCP connections. It can be seen that both connections transmitted traffic on opposite links exclusively for over 200 seconds without ever probing the second link to update SRTT value, which will lead to higher delay for at least one of the connections. Figure 5.5(c) shows the proportion of all MPTCP connections with a lower SRTT value for C1 during a train trip. It can be seen that for the majority of the time, 40% of the MPTCP connections had a lower SRTT estimate for C1, which means they were making mostly the opposite scheduling decisions compared to the remaining 60% of the connections, indicating that a significant amount of connections are affected by this.

New connections will always transmit the SYN packet over the default subflow, which may be slower. Figure 5.5(d) shows the CDF for the difference in delay between the JOIN packet and the SYN packet transmitted by the same MPTCP connection, which is a reasonable estimate for the difference in delay between the 2 available links at the moment of the start of the connection . It can be seen that for 20% of the connections, it would have been faster to initiate the connection over the secondary subflow, which for short flows can significantly affect completion time.

5.3.3 Congestion control



(a) (b)

Figure 5.6

The congestion control relies on data transmission for estimating the CWND parameter

which decides the sending rate. The multi-path scheduler also uses this parameter for deciding how much to transmit on each link. Similar to the SRTT, in an MPTCP proxy MWR, this estimation is also done by each connection independently resulting in a less accuracy. Having an inaccurate estimate for the CWND can cause undershooting of capacity, wasting capacity, or overshooting of capacity, causing losses.

When using loss-based congestion control (e.g. CUBIC [26]), an MPTCP connection will need to experience a loss on that link before lowering the sending rate. Figure 5.6(a) shows the number of active TCP/MPTCP connections during a train trip. It can be seen that there was around 1000 connections active for most of the trip, each with their own congestion control. This will result in a large number of un-necessary losses. These losses will disproportionately affect short connections that always start with the same value for the CWND potentially resulting in early losses that increase their completion time significantly. Figure 5.6(b) shows the mean loss rate for different connection lengths. It can be seen that the shorter connections, starting from around 8 packets in length, had significantly higher loss rate and it decreased as the connection length became longer.

5.3.4 Issues with a MPTCP Tunnel MWR

MPTCP tunnel MWRs are based on encapsulating TCP, and possibly UDP, connections and transmitting them over a single persistent MPTCP connection, as shown in Figure 5.2. While this approach eliminates the high number of parallel MPTCP connections in the proxy variant, it introduces new performance issues due to stacking multiple layers of reliability and cross-flow HoL blocking. However, since we were not able to acquire measurements from a production communication system that relies on an MPTCP tunnel MWR, as we did for the MPTCP proxy MWR, we discuss the issues in light of previous work and confirm them later in the evaluation section.

Stacking two layers of reliability

When a new TCP connection is created, it will often have a shorter re-transmission timer than the established MPTCP tunnel connection, as the latter is more adapted to the link properties. When data is lost in the MPTCP tunnel, it will queue up a re-transmission and increase its timeout. However, since the MPTCP tunnel will be blocked during the re-transmission period, the TCP connection does not get any acknowledgments, and it will thus queue up its own re-transmission of the same data. Since the timeout in the tunnel is less than the TCP connection, it may queue up even more re-transmissions causing the

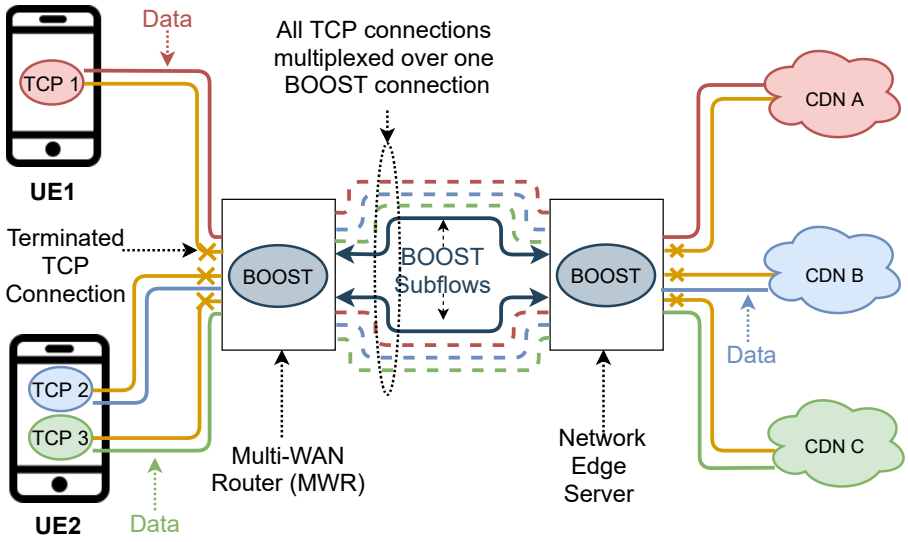


Figure 5.7: BOOST MWR

transmission of new data to stall completely, and every re-transmissions further compounds this problem. This behavior is referred to as TCP meltdown [12]. In wireless links under mobility, such as in trains, which experience frequent losses due to capacity fluctuations and handovers, frequent TCP meltdowns will occur.

Cross-flow HoL blocking

When data is received out of order due to loss or multi-path transmission over heterogeneous/time-varying links, received packets are held in a reordering buffer until they can be delivered in order. This is referred to as HoL blocking. In an MPTCP tunnel MWR, there is a single reordering buffer for all TCP connections being transported. Therefore, packets belonging to one TCP connection will have to wait in the re-ordering buffer needlessly until packets from another connection can be delivered in-order simply because packets for the latter were sent first. This issue is similar to the well-known HTTP head of line blocking problem [31].

5.4 BOOST design and implementation

The proposed solution, referred to as BOOST, is based on two main design features: multiplexing application flows over a single persistent multi-path connection and hybrid multi-path scheduling. In this section we will discuss both in detail.

5.4.1 Persistent multi-path connection with multi-plexing

The first feature of BOOST is that it transparently proxies and multiplexes TCP and UDP flows over a single persistent and selectively-reliable multi-path connection between the MWR and a network-edge server, as shown in Figure 5.7. When the BOOST client first starts, it establishes a connection with the BOOST server and then establishes all subflows (details of the protocol are discussed in the next section). The BOOST connection has a single multi-path scheduler and a single instance of congestion control for each link. When new TCP flows are routed through the MWR, BOOST terminates each TCP connection and multiplex the data over the existing multi-path BOOST connection. Similarly, UDP flows are also multiplexed but without enforcing reliability or in-order delivery. Data from individual TCP and UDP flows are tagged using a unique stream identifier to allow the receiver to distinguish between them.

This feature of BOOST addresses the problems in the variants of the MPTCP MWR as follows.

Underutilization of capacity Since all application traffic is handled by one multi-path scheduler, it allows the scheduler to optimally distribute all application traffic across the available links, eliminating the underutilization of capacity by short flows and on/off flows.

Suboptimal scheduling and increased loss rate BOOST maintains a single estimate for the CWND and the SRTT on each link that is generated by data transmitted from all traffic. This allows it to derive a much more accurate estimate of these parameters, leading to lower loss, better capacity utilization and more optimal multi-path scheduling. Additionally, since there is only one single congestion control process, a single loss on a link is sufficient for BOOST to reduce its sending rate, compared to distributed congestion control where each flow has to experience an independent loss.

Stacking multiple layers of reliability BOOST functions as a proxy by splitting TCP connections and thus does not stack multiple layers of reliability

Cross-flow HoL blocking BOOST stores a unique stream identifier inside each packet that links to its parent application flow and hence can avoid cross-flow HoL blocking.

5.4.2 The BOOST protocol

The BOOST protocol is based on QUIC with modifications to enable multiplexing of TCP and UDP connections, and multi-connectivity.

Multi-connectivity

To enable multi-connectivity, we adopt an existing proposal [13] for a multi-path version of QUIC (MPQUIC) that modifies the protocol in two ways. First, the main QUIC packet header includes a new PathID field which identifies which subflow the packet belongs to. Furthermore, the existing ConnectionID field is used to link a path to the original connection. There are no handshakes needed to establish secondary subflows like in MPTCP. Second, acknowledgement (ACK) packets also include the PathID field to identify which path the ACK packets belong to. Finally, the PacketNumber field is specific to each path.

Multiplexing

QUIC already includes a multi-streaming functionality, so we leveraged this to enable our multiplexing feature. In the BOOST implementation of QUIC, we modify the use of streams to refer to TCP connections instead of the traditional application streams. Each TCP connection is assigned a unique StreamID by BOOST.

To enable each proxied TCP connection to be forwarded to its correct final destination, BOOST implements a new QUIC extension/frame called SET_FORWARD_ADDRESS. It contains the following information: ConnectionID, StreamID, and the application destination ip, port and Protocol. This frame is transmitted with the first packet sent for an application flow.

Toggling reliability and in-order delivery

BOOST does not impose reliable delivery on UDP connections that it is carrying. In order to inform the BOOST server about the reliability of a particular stream, the BOOST client leverages the FIN flag in the first STREAM frame sent for a particular stream to indicate reliability to the BOOST server. An unreliable stream will not wait for in-order delivery before being forwarded. Additionally, neither the BOOST client nor server will re-transmit lost UDP packets, but the congestion control will observe the loss and reduce rate appropriately.

5.4.3 Hybrid multi-path scheduling

The second key feature of BOOST is the hybrid multi-path scheduling approach which uses multi-path load balancing for short flows in order to minimize delay by preventing HoL blocking and opportunistic multi-path scheduling for longer flows to make use of left-over capacity. BOOST also supports replicating data from selected flows over all available links.

The scheduler is split into 4 phases as follows

Phase 1 (Flow-link assignment)

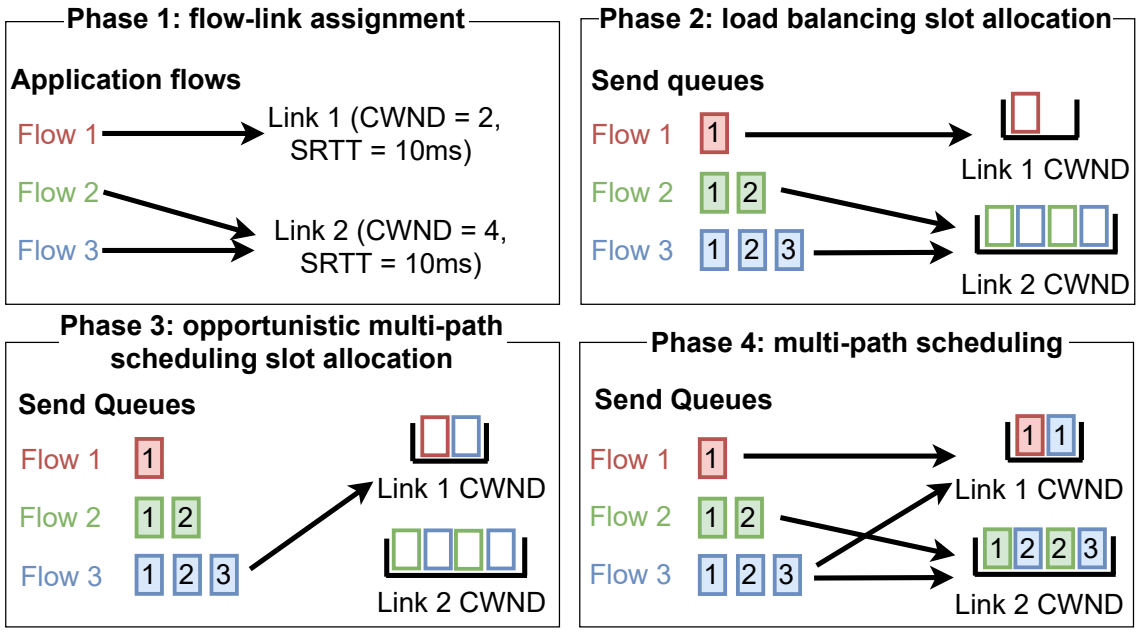
The scheduler assigns application flows to available links according to the ratio R_i, which is calculated as follows

$$R_i = \frac{t_i}{\sum_{j=0}^{m} t_j} \quad \text{and} \quad t_i = \frac{CWND_i}{SRTT_i} \tag{5.2}$$

where $CWND_i$ is the CWND of link i, $SRTT_i$ is the SRTT of link i, and m is the number of available links. The goal is to load balance flows over links proportionate to the ratio of the throughput of the links. Load balancing occurs periodically every γ milliseconds, where γ is a design parameter we choose to be 100ms, and every time a new connection is detected. Applications flows that have not received new data for over one second, are removed from the assignment. Connections configured to be replicated are assigned to all links.

Phase 2 (load balancing slot allocation)

After assigning a flow to a link, it is inserted into a priority queue ordered by the amount of bytes the connection has transmitted so far. If a flow has not transmitted anything yet, this value is 0. The front flow in the priority queue is then selected and at most one maximum transmission unit (MTU) worth of data, which is typically 1500 bytes, is allocated on the assigned link's send queue for use by this flow. The bytes-transmitted counter for the flow is then incremented by the amount of data allocated on the link and the flow is re-inserted into the priority queue. This process is repeated until all flows assigned to that link have been allocated space equal to the data in their send queue (i.e. they have no more data) or the total allocated data on a link is equal to its available CWND. For connections with replication configured, the assigned links are cycled every time they have a slot allocated.



(a)

Figure 5.8: Hybrid multi-path scheduling example

Phase 3 (Opportunistic multi-path scheduling slot allocation)

This phase of the BOOST scheduler will only trigger if at least one link has space in its CWND and at least one flow has data that has not been allocated space on a link. All eligible application flows are placed in another priority queue that is also ordered by the amount of data transmitted just like before. A multi-path scheduling algorithm will then select between all available links to allocate space for data from the selected connection. For example, the MinRTT scheduler would pick the link with the lowst SRTT amongst the links that have space in their CWND. Afterwards the flow's transmitted data counter is incremented and it is returned to the priority queue. Replicated flows are not selected in this phase.

Phase 4: Multi-path scheduling

In this phase packets are taken from the flow send queue and placed on links with allocated space. If phase 3 did not occur (i.e. a connection only has allocated space on one link) then packets are placed on that link in order of their sequence number. If phase 3 did occur, the same multi-path scheduling algorithm as in phase 3 assigns packets from the flow send queue to the allocated space on the links.

An example

Fig 5.8 shows an example of the hybrid multipath scheduling. In phase 1, there are 3 flows to be assigned to two links. Since link 1 and link 2 have the same SRTT but link 2 has double the CWND, two flows are assigned to link 2, while only one flow is assigned to link 1. In phase 2, each flow is allocated space on the assigned link. Flow 1 is assigned one MTU slot on link 1, while flows 2 and 3 are assigned 2 slots on link 2 each. Note that the flows were alternated in the order of the assignment due to the priority changing after being assigned a slot. In phase 3, the opportunistic multi-path scheduling triggered because flow 3 still had one packet remaining in its send queue and link 1 had a space in its CWND. As such, flow 3 was allocated one slot on link 1. Finally, in phase 4 packets were placed in the allocated spaces by the multi-path scheduling algorithm. In this case, the algorithm placed packet 1 from flow 3 on link and packet 2 from flow 3 on link 2 as both of these slots were likely to be delivered before the last slot on link 2, where it placed the last packet from flow 3.

5.4.4 Implementation

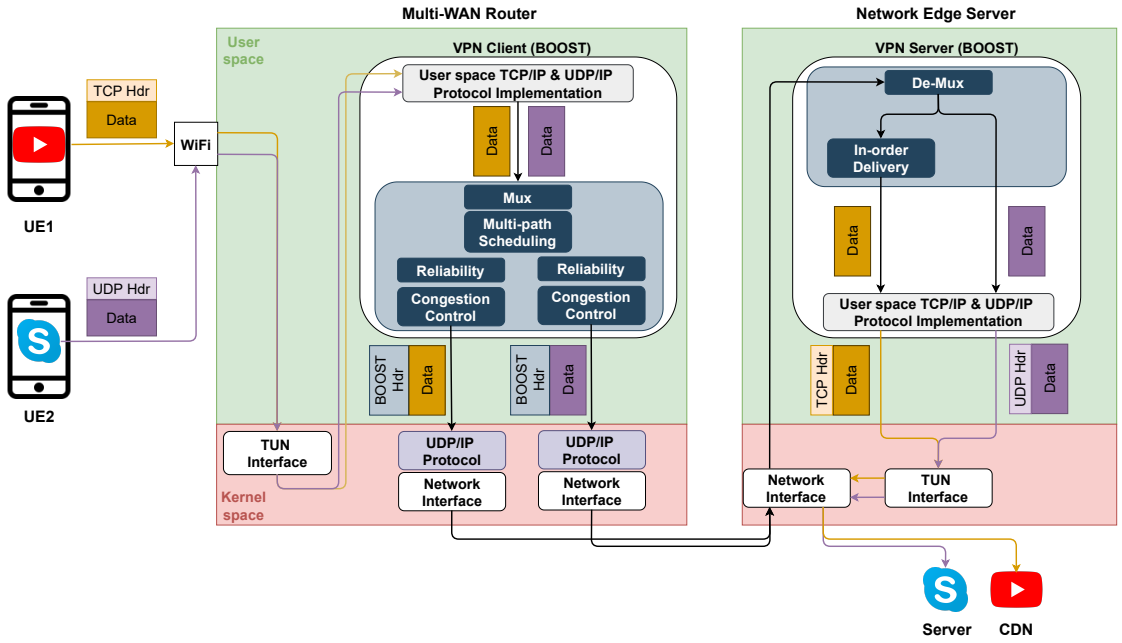
BOOST is implemented in Linux using C++ as a user-space protocol based on the VPN approach outlined in Chapter 3. This allows it to be easily deployable in an MWR as a user-space application without any modifications needed to the MWR's kernel or operating system. Figure 5.9 shows the software architecture during uplink transmission of both TCP and UDP traffic. As shown, BOOST implements its own reliability and congestion control on top of UDP sockets similar to QUIC[49].

5.5 Evaluation

We evaluated BOOST using a trace-driven emulation based on the measurements collected in Section 5.2 against both MPTCP variants as well as a single-path TCP proxy over both links.

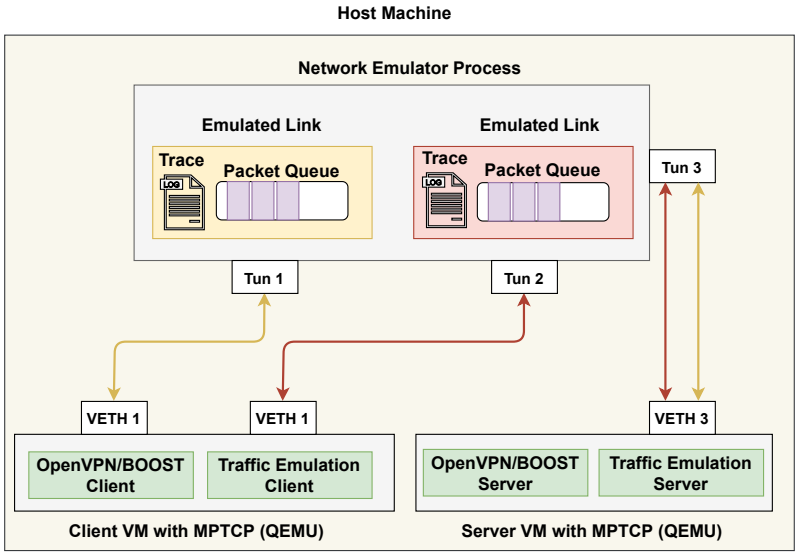
5.5.1 Setup

Two Python scripts, one acting as a client and the other as a server, were used to emulate the passenger side and the content server side respectively. The client script initiated TCP connections at with the same frequency as the client traces and transmitted an identifier to the server script to denote which connection in the traces corresponded to the current



(a)

Figure 5.9: BOOST architecture during uplink transmission



(a)

Figure 5.10: Emulation setup

connection. The server script then transmitted packets over the TCP connection with the same sizes and frequency as the server traces. To understand how the performance of each schemes scales, three versions of the traces were created: one with the same number of connections, one with 3-times the number of connections and one with 5-times the number of connections. The extra connections in each version of the trace were created at the same time as the original and transmitted the same amount of data. In addition to the TCP traffic, the server script also transmitted dummy UDP video traffic to represent a closed-circuit television (CCTV) camera. The CCTV traffic was emulated by transmitting UDP packets at a rate of 1Mbit/s.

The scripts were placed inside two QEMU virtual machines (VM) with a v0.95 MPTCP kernel, as shown in Figure 5.10. MPTCP was configured with the default send and receive buffer sizes, and used the default MinRTT scheduler. The client VM was given two virtual interfaces to emulate the two LTE links in the measurement setup. Data transmitted from or to these virtual interface was forwarded into a corresponding virtual tunnel interface in a network emulator process running in the host. The network emulator throttled data received by or transmitted to each tunnel interface based on the capacity of the corresponding LTE link, as will be explained in more detail below. Afterwards, the packets were forwarded to the server VM or the client VM.

The client script measured the mean throughput received for all TCP connections during their period of activity, as well as the delay of the CCTV traffic. It detected periods of activity by observing a flag set by the server script to indicate the end of a burst of transmission, which are packets that were sent back to back. This is to eliminate the impact of on/off connection on the mean throughput. The CCTV traffic delay was measured by synchronizing the clocks on both client and server VMs, which was easy since they're hosted on the same machine, and attaching a timestamp inside each transmitted UDP packet by the server script.

The MPTCP kernel probe, described in Section 5.2.2, was used in both VMs to calculate the re-transmission rate and the CDF of the out-of-order queue size. The former was calculated by comparing the number of total re-transmission events and the number of transmission events. The latter was calculated by observing enqueuing and dequeuing events into the out-of-order queue. Data on the same performance metrics was also collected for BOOST.

6 different schemes were evaluated: MPTCP C1, MPTCP C2, TCP C1, TCP C2, MPTCP TUN, and BOOST. The first two, MPTCP C1 and C2, represent MPTCP proxy

setups with default route set to carrier 1 and carrier 2 respectively. There was no need for the explicit usage of a proxy here since the MPTCP kernel in the client VM transparently converted TCP connections to MPTCP connections. The second two, TCP C1 and TCP C1, are single-path TCP with default route on carrier 1 and carrier 2 respectively. MPTCP TUN represents the MPTCP tunnel setup. To realize this setup, we used an OpenVPN client on the client VM and OpenVPN server on the server VM. It was configured to tunnel traffic over TCP and all traffic was routed through its tunnel interface. We also stripped the MPTCP options from the non-OpenVpn TCP flows in order to ensure there's only a single MPTCP flow between the two OpenVpn processes on each of the VM. BOOST was evaluated in a similar way to the MPTCP TUN by setting up the BOOST client as a VPN client and the BOOST server as a VPN server. To improve CCTV performance, BOOST was configured to replicate the CCTV and prioritize it.

The whole setup ran on a Dell XPS 15 running Ubbuntu 20 with an Intel i7-7700HQ processor and 16GB of RAM. Each QEMU instance was given 2 different cores and 4 GB of RAM.

5.5.2 Emulating capacity

The emulator was implemented in C++ as a separate user-space process, which intercepts packets transmitted by both the client and server VMs using IPTables [42]. As shown in Figure 5.10, data flowing through a particular interface had its capacity emulated to reflect the corresponding LTE link capacity in the traces. Once packets were received by the emulator, the packets would be added to the queue specific to that LTE link. The queued packet is then delayed sufficiently or dropped to reflect the capacity profile that belonged to that link. A packet would be dropped if the queue size exceeded 100 packets. If the packet is not dropped, it will be removed from the queue once its delay duration expires and forwarded to the destination VM. The emulator performs a busy wait whenever it has packets in the queue in order to eliminate the need of high resolution timers and the in-accuracy associated with them. The other advantage of this emulator compared to the Linux built-in network emulator NetEm [43] is the ability to switch very efficiently between different rates as the emulation progresses in time without relying on expensive system calls.

Calculating packet delay

The emulator calculates delay for each packet based on the current capacity at the time of packet transmission in the emulation and the queuing delay. Using c^i, defined in section 5.2.1, which is the capacity in the period $[t^i_{start}, t^i_{end}]$, the delay d^j for packet of size s^j and transmitted at time t^j_{send} where $t^j_{send} \in [t^i_{start}, t^i_{end}]$ is computed as follows

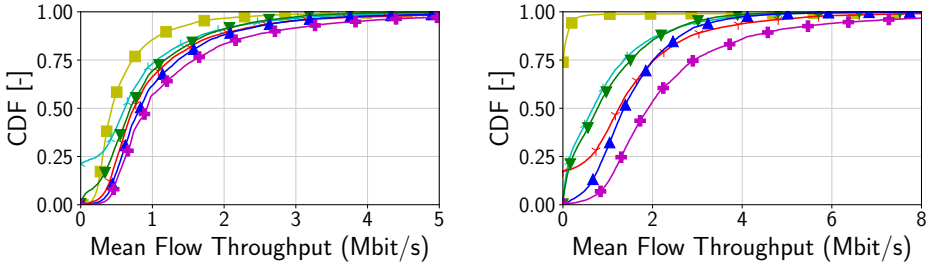
$$d^i = \frac{s_i}{c^i} + q_i \tag{5.3}$$

where q_i is queuing delay resulting caused by buffer buildup. In case where the number of queued packets in the emulator exceeds 100, the packet will be dropped due to buffer overflow.

5.6 Evaluation Results

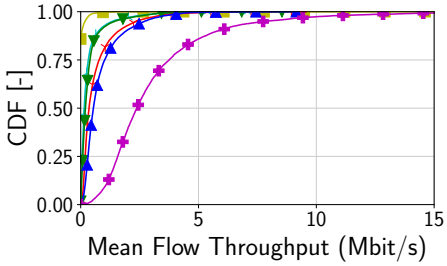
Figures 5.11(a)(b)(c) show the CDF of the mean flow throughput for passenger WiFi traffic with the connection multiplier at 1x, 3x and 5x respectively. We can observe that BOOST consistently outperformed all solutions, with the gap widening as the multiplier increased. In the 5x multipler, BOOST had a mean throughput of more than 2.2. Mbit/s, which is around 10 times higher than any of the MPTCP variants. MPTCP C1 performed very similarly to TCP C1 and MPTCP C2 performed very similarly to TCP C2, which implies that the performance was mainly dependent on the choice of default route and that multi-connectivity provided by MPTCP did not improve performance. MPTCP TUN performed by far the worse of all configurations due to the spurious re-transmissions caused by TCP meltdown discussion in section 5.3.4. This becomes clear when observing the high re-transmission ratio in Figure 5.12(b), where the MPTCP TUN had almost 18% re-transmission rate in the 3x and 5x scenarios. Figure 5.11(d)(e)(f) shows the out-of-order queue size for all schemes. The hybrid multi-path scheduler allowed BOOST to practically eliminate the need for an out-of-order queue since most TCP connections were load balanced over individual links.

Fig. 5.12 (a) show the CCTV delay for all solutions. We observe that MPTCP TUN had the best CCTV delay, with BOOST having similar delay but higher 95% percentile. The reason for this is that CCTV was sent over UDP, and MPTCP TUN had extremely low TCP throughput due to the TCP meltdown effect, hence freeing up almost all the capacity for the CCTV UDP-based traffic. However, it can be seen that BOOST performed significantly better than all the remaining solutions due to the prioritization and the

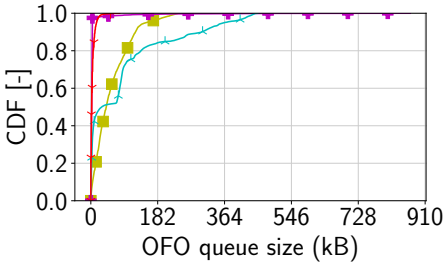


(a) Passenger WiFi 1x

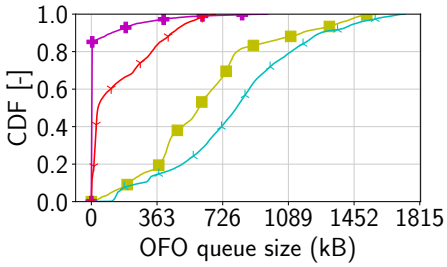
(b) Passenger WiFi 3x



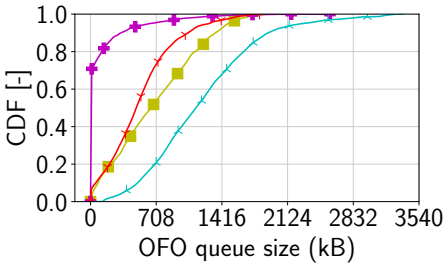
(c) Passenger WiFi 5x



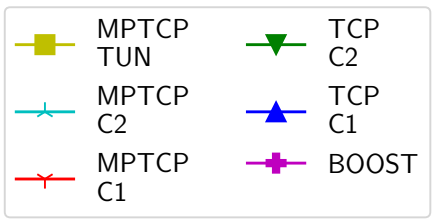
(d) out of order queue size 1x



(e) out of order queue size 3x

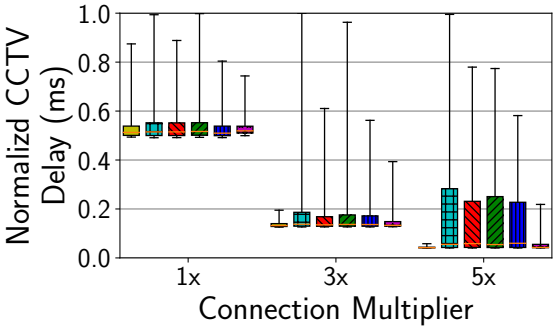


(f) out of order queue size 5x

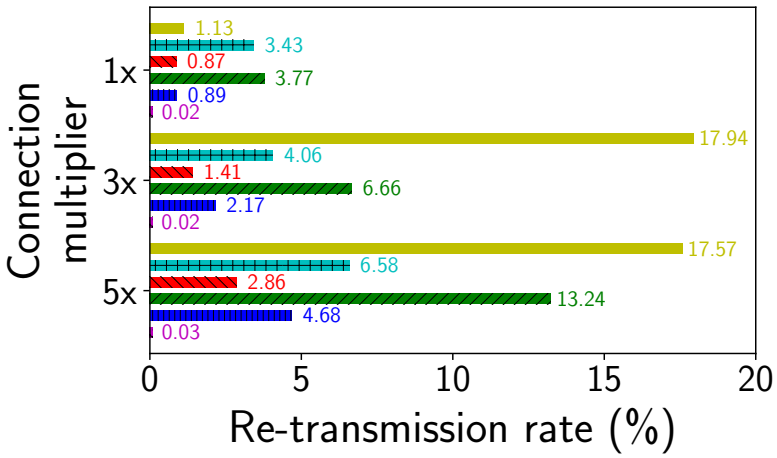


(g) Legend

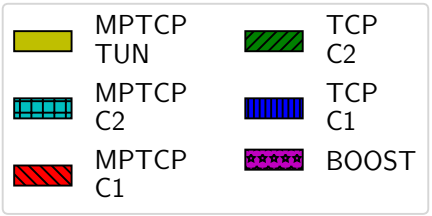
Figure 5.11: (a)(b)(c) shows the CDF of mean flow throughput for all solutions with connection multipliers 1x, 3x and 5x respectively. (e)(f)(g) shows the CDF for the out-of-order queue size for all solutions with connection multipliers 1x, 3x and 5x respectively.



(a) CCTV delay



(b) Re-transmission Rate



(c) Legend

Figure 5.12: (a) shows the boxplot for the CCTV frame delay for all solutions with connection multipliers 1x, 3x and 5x. (b) shows the re-transmission percentage for all solutions with connection multipliers 1x, 3x and 5x.

redundancy.

Overall, BOOST provides better throughput, with lower re-transmissions, and a smaller memory footprint compared to all MPTCP variants for TCP traffic, and improved reliability for UDP traffic.

5.7 Related Work

Two related studies have been done into the performance of MPTCP when handling a large number of parallel flows [84][2]. In [84], authors propose a Software-Defined Networking (SDN) architecture that enables coordinated multi-path routing across different managed networks to address issues with un-coordinated MPTCP scheduling. Specifically, they propose a weighted MPTCP round-robin scheduler that transmits percentages of the traffic per network interface according to a configuration that is communicated through a centralized scheduler. In [2], the authors proposes a client-only approach to load balance TCP connections across multiple network interfaces. They base the validity of this approach on the fact that typical web traffic creates a large number of TCP connections creating ample opportunity for connection-level load balancing to improve network performance.

In the area of optimizing MPTCP performance when dealing with different kinds of flows, authors in [102] make the observation that the MPTCP scheduling approach of one-size-fits-all is not efficient when dealing with smaller flow sizes. They separate flows into Elephant flows, Mice flows and Mosquito. They suggest Elephant flows should be scheduled over all available links using round-robin, mice flows should be scheduled using redundancy, and Mosquito flows should be simply sent on the best available path. They propose a cross-layer solution called Flow Size Aware MPTCP (FSA-MPTCP) where applications signal the kind of flow they are about to transmit to the MPTCP scheduler so it can select the appropriate scheduling strategy.

In [34] authors aim to address the high memory requirement of MPTCP connections created by HoL blocking on IoT devices with limited memory resources. They propose an application-layer scheme that manipulates an underlying MPTCP implementation to allow the application-layer to transmit its objects over different links but ensuring each object is sent over one link only. The application utilizes multiple MPTCP sockets and writes different objects to different sockets. To ensure that the MPTCP implementation does not schedule all the objects across the same link, the application will alternate configuring one link as backup, using MPTCP socket options, for each socket it creates, ensuring that

MPTCP will not use the backup link unless the first link goes down.

5.8 Conclusion

In this chapter, we outlined the issues in the tunnel and proxy variants of MPTCP MWR. The issues in the proxy variant were confirmed using measurements from a production T2G communication system that relies on an MPTCP proxy MWR. In particular, measurements showed that the production system suffers from underutilization of capacity, sub-optimal multi-path scheduling and high loss. Issues in the tunnel variant were confirmed the using trace-driven emulation based on the same measurements collected from T2G system. To address the issues in both variants, we proposed a new multi-path solution more suited to MWR, called BOOST. This solution eliminates the problems with both the proxy and tunnel approaches by multiplexing TCP and UDP connections over a single persistent multi-path connection. BOOST also takes a novel approach to multi-path scheduling that combines multi-path load balancing and scheduling. In particular, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity. Evaluations based on trace-driven emulations of BOOST against both variants of MPTCP as well as single-path TCP showed that BOOST provides better throughput, lower loss than all other schemes and consumes less memory than all MPTCP variants. Additionally, BOOST provided improved latency and reliability for UDP traffic through redundancy and prioritization.

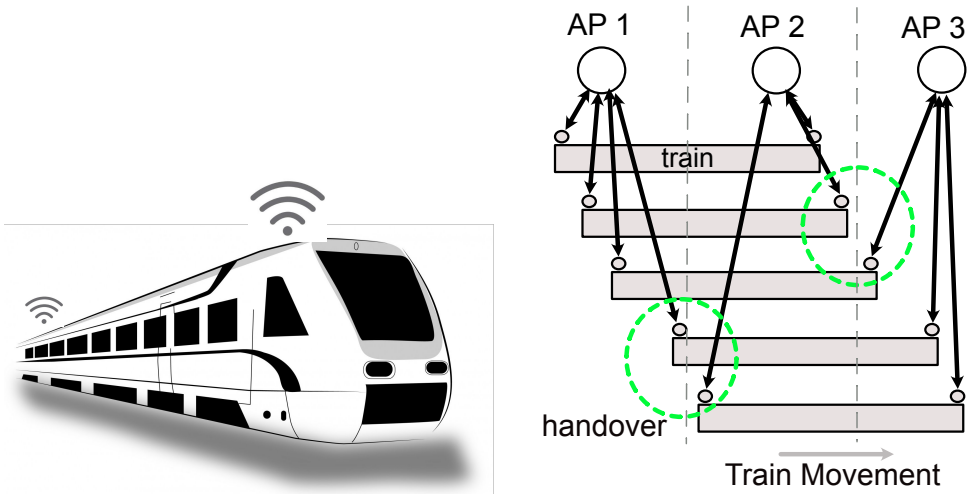
Chapter 6

Smart replication for seamless and efficient real time communication in underground railway train

6.1 Introduction

In this chapter we consider the setup shown schematically in Figures 6.1 and 6.2. A subway train is equipped with two WiFi clients, one at the front and one at the rear of the train. WiFi access points (APs) are placed along the trackside. As the train moves along the track the WiFi clients switch between APs. The network connection of a client is interrupted when it switches between two APs, but by placing the trackside APs less than one train-length apart the handovers of the two clients are de-synchronised i.e. at all times at least one WiFi client on train is connected to an AP, see Figure 6.1(b). When a client switches between APs downlink packets queued at the old AP are discarded, as are any uplink packets that exceed their delivery deadline due to the delay caused by switching. Packet loss can therefore occur at each handover, as illustrated in Figure 6.2(b). The network capacity available to the train also fluctuates greatly over time since sometimes both clients are connected and sometimes only one client, plus the bandwidth of the wireless link(s) is variable as the train moves and the external environment changes.

Our interest is in managing the quality of service at the transport layer using an over-the-top approach. Traffic to and from the client stations on the train is routed via a proxy on the wired network close to the APs, Figure 6.2(a). This creates the freedom to implement new transport layer behavior over the path between proxy and clients. The approach assumes only control of the proxy client and gateway, but no knowledge or



(a) (b)

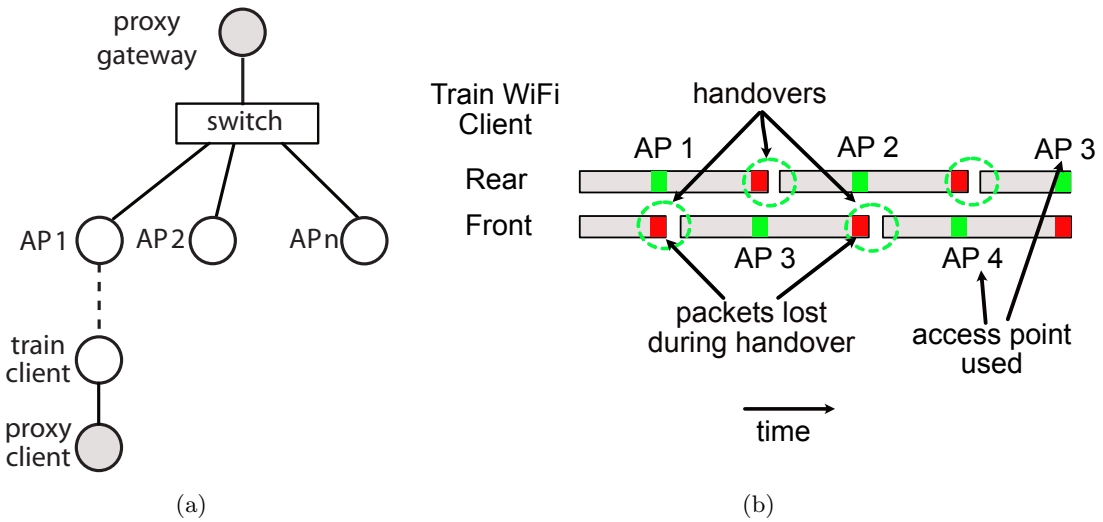
Figure 6.1: Illustrating setup considered. (a) A train is equipped with two 802.11ac WiFi clients, one at the front and one at the rear. WiFi access points are installed along the trackside. (b) The spacing of the trackside access points is selected so that at least one of the train WiFi clients is connected at any time, e.g. for a train of 150m length the trackside access points might be spaced 100m apart.

control of the internal workings of the WiFi clients or the APs. The great advantage of this over-the-top approach is its ease of rollout since no changes are required to existing AP's or WiFi clients. However, the price is that the internal WiFi decision-making on when to switch between APs is hidden i.e. we can observe when switching occurs but we cannot observe or change the switching logic.

The WiFi connection between a subway train and the trackside carries two main classes of traffic: (i) train control-plane traffic (i.e. Communication Based Train Control (CBTC) traffic on driver-less trains, CCTV monitoring for track and on-board video surveillance, mission-critical VOIP) and (ii) passenger data traffic from public WiFi hotspots within the train. Here we focus on (i), where the QoS requirement is for high availability, and low packet loss [90].

Since packets may be lost at each handover, multiple studies have proposed to replicate traffic to multiple wireless clients as a way of ensuring quality of service in train-to-ground (T2G) [52][54][53]. Since one client is, by design, always connected this avoids packet loss. However, this comes at the cost of many duplicate packet transmissions and so wasted network capacity.

A smarter approach is to predict when a handover is about to occur and then only replicate packets during this short period, illustrated by the green shading in Figure 6.2(b), and it is this approach that we study here. By reducing the number of duplicate packets sent the potential exists to free up significant amounts of network capacity for more productive uses, e.g. for carrying passenger data traffic. It is important to stress that our goal is to *predict* when a handover is about to occur and not to trigger the



(a) (b)

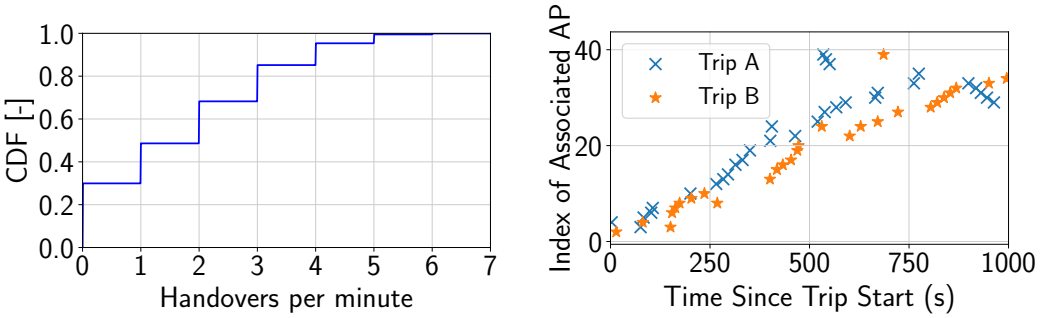
Figure 6.2: (a) Edge proxy setup. (b) Illustrating downlink packet transmissions from the train front and rear WiFi clients. As the train moves the AP to which each client is connected changes, e.g. the rear WiFi client is initially connected to AP1, then to AP2 and so on. During handovers there is a break in connectivity and packets queued at the old access point are lost (indicated in red).

handover since, as already noted, we have no internal access to the WiFi APs/clients and so have no control over handovers.

It turns out that accurate prediction of handovers is surprisingly tricky since, based on extensive measurements from trains operating in a production environment, the WiFi APs/clients exhibit complex behaviour. The sequence of handovers can change substantially between runs over the same track and predictions must take account of periods when the train is stationary (such as when waiting in a station or stopped at a signal), when the train is accelerating/decelerating etc. We propose a novel neural network-based predictor to address these issues and evaluate its performance using experimental data.

A key observation is that a fundamental trade-off exists between the accuracy of the handover prediction, the level of packet replication and the level of packet loss. Namely, false positives (a handover is predicted but does not actually occur) cause unnecessary packet replication whereas false negatives (it is predicted that there will not be a handover, but one occurs) lead to unnecessary packet loss. We can always avoid false negatives trivially by always predicting that a handover is about to occur, and this yields the wasteful strategy of continuously replicating packets to both WiFi clients. We can also avoid false positives by always predicting that no handover will occur, and this yields the strategy of never replicating packets and so results in a high loss rate. Our second main contribution is to characterize the middle ground between these two extremes and quantify the trade-offs.

We propose a simple adaptive packet replication scheme and evaluate its performance. Results show that when application throughput is low compared to link capacity (i.e. extra



(a) (b)

Figure 6.3: Experimental measurements of train-to-trackside WiFi handover events on an underground train.

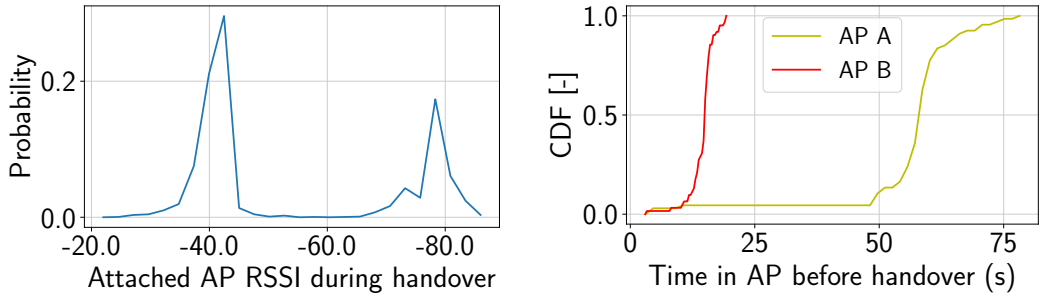
redundancy does not come at the cost of increased congestion losses), our scheme achieves 91% of the loss reduction provided by simple replication on average while using only 13% of the redundancy and experiencing 20% less latency. However, when application throughput is high compared to link capacity, simple replication causes more losses compared to no replication making it counter productive, while our scheme maintains a lower loss rate compared to no replication.

In summary, our main contributions include: (i) development of a neural network-based handover predictor, (ii) quantifying the trade-offs between prediction accuracy, packet replication and loss, (iii) development of a simple adaptive packet replication scheme and its performance evaluation using experimental data.

6.2 How Do T2G WiFi Handovers Behave?

We collected the logs generated by the on-board WLAN client on an underground train using a commercial T2G system during 33 trips over the same track, spanning over 17 hours. The client logged the result of all RSSI scans performed during the trip as a list of MAC addresses with corresponding RSSI values and the current time. It also logged each handover event, recording the MAC address of the newly associated AP and the current time. By analyzing the logs, we were able to parse 58935 RSSI scans and 2455 handover events.

Fig. 6.3(a) shows the measured CDF of the number of handovers per minute i.e. time is divided into one minute slots and the empirical CDF of handovers per slot calculated. Observe that in around 30% of time slots no handovers occur. Note that this data includes time slots where the train is stopped in a station and at signal lights and so when the train is moving the fraction of slots with no handover is significantly smaller. It can also be seen that around 60% of slots experience between 1 and 3 handovers and around 15% of slots



(a) (b)

Figure 6.4: Relationship between handover and the RSSI of the attached AP and the time spent connected.

experience 4-7 handovers. The handover duration i.e. of the time between disassociating from the old AP and associating with the new AP takes at least 40ms and in 99% of handovers take no more than 200ms. During this time network connectivity is lost and following handover packets queued at the old AP are discarded•

Fig. 6.3(b) shows the APs that were connected to during two different trips across the same section of track. Observe that the sequence of APs differs significantly despite the fact that these measurements are for the same section of track. While some of the changes are essentially a shift/offset in the sequence, presumably due to fluctuations in train speed, more complex differences are also evident. For example, APs used on one trip need not be used on the second trip and the order in which APs are connected can be reversed between trips (see the right-hand side of the plot, around time 800-900s).

To try to gain more insight into the factors that affect the complex handover behaviour, Figure 6.4(a) plots the empirical PDF of the RSSI of the AP to which the WLAN client is attached when a handover starts. It can be seen that that handovers are concentrated around two ranges of RSSI values, -40dB and -80dB. It is not surprising that a low RSSI of -80dB increases the probability of a handover since this likely another AP has higher RSSI and so would be preferred. However, Figure 6.4(a) also indicates that a much higher RSSI of -40dB is also likely to be associated with a handover. The reason for this is not clear, presumably it is due to interactions between the complex radio environment and the handover switching logic within the APs/clients, but it is clearly a useful observation for predicting when a handover may occur.

For two different APs Fig 6.4(b) shows the empirical CDF of the time spent attached to the AP before handover occurs. When connected to AP B the train is generally stopped

•There currently exists no WLAN standard to facilitate forwarding data during a handover. While there were previous attempts to create such a standard, namely the Inter-Access Point Protocol (IAPP) which was referred to as 802.11F, they were not successful and the standard has since been withdrawn [37].

in a station and so the time spent attached to the AP is rather long (¿ 1 minute), whereas the train is generally moving when connected to AP A and so the train spends a much shorter time attached to AP A (around 12 seconds). While the overall time spent attached to an AP is therefore highly variable, the minimum time is around 10 seconds and this again can be used to assist with predicting when a handover may occur.

6.3 Predicting Handovers

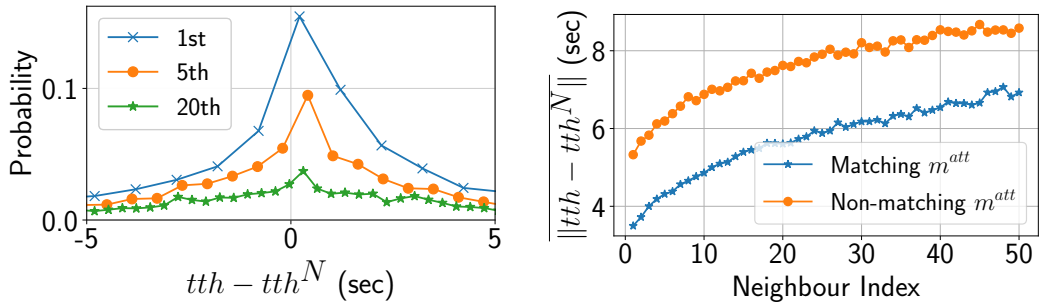
We would like to predict whether a handover will occur on a train WLAN link within α seconds. Here α is a design parameter that reflects the amount of time needed to flush the associated AP's buffer before an upcoming handover and to switch to transmitting packets primarily over the second train WLAN link (perhaps with duplicates still sent over the link that is due to handover). We select α to be 3 seconds. We approach this prediction task as a binary classification problem. Each time a train WiFi client carries out a scan for nearby APs we construct a feature vector X, input this to a classifier which outputs +1 if a handover is predicted to occur with the next α seconds and otherwise outputs -1. Since the train traverses the same route every day, measurements of handovers are readily available for use as training data for the classifier.

6.3.1 Feature Selection

Baseline Features

Based on the data in Figure 6.4, plus our understanding of the wireless system, it is clear that the RSSI of the AP to which a client is attached and the time since the last handover are likely to be useful features for predicting handover. Also we use the MAC address of the attached AP as a feature since we expect there may be consistent differences between APs due to differences, for example, in the local radio environment. Since we know that the train changes speed and, in particular, can spend extended periods time stationary we also include the rate of change of the RSSI as a rough proxy for train speed.

More formally, let $r_{i,j}^{att}$ denote the RSSI of the attached AP during scan i in trip j and $tsh_{i,j} = t_{i,j} - t_{i,j}^{PHO}$ be the time since the last handover, where $t_{i,j}$ is the time of scan i in trip j and $t_{i,j}^{PHO}$ is the time of the last handover before the scan. We encode the AP MAC address using one-hot encoding i.e. as a vector of length n, where n is the number of APs, and with all elements 0 except for the element corresponding to the currently attached AP which has value 1, and is referred to as $\vec{M}_{i,j}^{att}$. Letting $M_{i,j}$ be the set of MAC addresses



(a) (b)

Figure 6.5: Fig (a) shows the PDF of the difference between the time till handover (TTH) associated with a scan and the TTH associated with similar scans/neighbours sorted by Euclidean similarity. The plot shows the PDF for 3 neighbor values: 1, 5 and 20. Fig (b) shows the mean of the same value across different neighbor indexes, both when limiting the comparison space to scans that share the same attached AP, matching m^{att}, and when no limits were placed, non-matching m^{att}.

observed in scan i then the rate of change of RSSI is $s_{i,j} = \sum_{m \in M_{i,j} \cap M_{i-1,j}} \|(r^m_{i,j} - r^m_{i-1,j})\|$ where $r^m_{i,j}$ is the RSSI of the AP with MAC address m.

TTH of nearest neighbours

In addition to the baseline features, we use the time till handover (TTH) associated with scans that have a similar RSSI fingerprint to the current scan as an additional feature. The underlying intuition is that similar RSSI fingerprints will occur in similar physical locations and thus the distribution of their TTH should be similar. Fig 6.5(a) shows the PDF of the difference between the time till handover (TTH) associated with a scan and the TTH associated with similar scans/neighbours sorted by Euclidean similarity. As shown, for the closest neighbour (i.e. 1st) , there is roughly 15% chance of no error at all and a high probability of low error, making it reasonable estimate. Since variations in speed and noise in similarity, caused by fluctuations in RSSI, will contribute error to this estimate, the TTH from the top k similar scans is used , which allows the classifier to assign appropriate weights to them. Additionally, the probability of larger difference/error increases as the index of the neighbor index gets higher, making the choice of k important. We have empirically selected the value of k to be 15 in our implementation.

More formally, let $tth_{i,j} = t^{NHO}_{i,j} - t_{i,j}$ denote the TTH from scan i in trip j, where $t_{i,j}$ is the timestamp of the scan and $t^{NHO}_{i,j}$ is the timestamp of the next handover after that scan. Having defined the tth for all scans, we compute the vector of k tth estimates for scan i, j, $\vec{TTH}_{i,j}$=$[tth_1, tth_2 \ldots tth_k]$, where each tth estimate is associated with the respective RSSI scan in the vector of k most similair RSSI scans, $\vec{N}_{i,j}$=$[\vec{R}_1, \vec{R}_2 \ldots \vec{R}_k]$, such that tth_a is the time till handover after RSSI scan $\vec{R}_a$. An RSSI scan $\vec{R}_a$ is the vector of RSSI values appearing in scan a. $\vec{N}_{i,j}$ is sorted using similarity function D, such

that $D(\vec{R}_{i,j}, \vec{R}_a) \leq D(\vec{R}_{i,j}, \vec{R}_{a+1})$. In our implementation we select D to be Euclidean Similarity. The similarity between any two scans a and b is calculated as follows

$$D(R_a, R_b) = \sqrt{\sum_{m \in M_a \cup M_b} (r_a^m - r_b^m)^2} \tag{6.1}$$

where M_a and M_b are the set of MAC addresses appearing in scan a and scan b respectively, and r_a^m and r_b^m are the RSSI value of AP with MAC address m in scan a and RSSI value of AP with MAC address m in scan b respectively. In cases that $M_a \neq M_b$ (i.e. an AP appears in scan a but not in scan b or vice versa) a sentinel value is assigned for the RSSI of any missing MAC in a scan, such that $r_a^m = -90$ if $m \notin M_b$ and similarly $r_b^m = -90$ if $m \notin M_a$.

To improve accuracy of the tth estimates in $T\vec{T}H_{i,j}$, the comparison space of a scan i, j is limited to the set of scans $C_{i,j}$, defined below, where the currently attached APs match

$$C_{i,j} = \{R_a \quad | \quad m_{i,j}^{att} = m_a^{att}\} \tag{6.2}$$

First, this greatly reduces the number of scans in the comparison space, improving the time complexity. Second, it also improves the accuracy as it avoids the problem of assigning high similarity to two scans where one occur just before a handover and one that occurs just after, and thus have both have a similar RSSI fingerprint but a very different *tth*. 6.5(b) shows the relation between the index of the neighbor on the x-axis and the mean difference between the original scan and the neighboring scan on the x-axis, both when limiting the comparison space to scans that share the same attached AP, Matching m^{att}, and when no limits were placed, Non-matching m^{att}. We can clearly observe that the difference was higher when the attached AP wasn't matched.

6.3.2 Feature Engineering

Normalization

All the input features, except $m_{i,j}^{att}$, are continuous and have different scales and are normalized as follows

$$z_{i,j} = \frac{x_{i,j} - \bar{x}_j}{s_{x_j}} \tag{6.3}$$

where $x_{i,j}$ is the *ith* value of the feature j, $\bar{x}_j$ and s_{x_j} are the mean and standard deviation of the feature over the training dataset, respectively.

Random oversampling of handover events

Since only 4% of the scans are handover events, this results in a severe imbalance in the number of class labels and may cause the cost function of the classifier to be skewed towards predicting non-handovers. To address this problem, random oversampling of handover events is performed to increase the number 25-fold, matching the number of non-handovers.

6.3.3 Training and prediction

Since this is a binary classification problem, the classifier is trained using the input feature vector $\vec{X}_{i,j}$, corresponding to features extracted from scan i in trip j, and the binary output class label $y_{i,j}$ defined as follows

$$y_{i,j} = \begin{cases} 1, & tth_{i,j} \leq \alpha \\ 0, & tth_{i,j} > \alpha \end{cases} \tag{6.4}$$

For prediction, the class probability generated from the classifier, $P(\hat{y}_{i,j} = T|\vec{X}_{i,j})$, is compared against a threshold θ in order to predict the handover as follows

$$\hat{y}_{i,j} = \begin{cases} 1, & P(\hat{y}_{i,j} = T|\vec{X}_{i,j}) \geq \theta \\ 0, & P(\hat{y}_{i,j} = T|\vec{X}_{i,j}) < \theta \end{cases} \tag{6.5}$$

6.3.4 Evaluation metric

The evaluation metric is based on the number of true positive (TP), which indicate correctly identified handover events, and false positives (FP), which indicate incorrectly identified non-handover events. Bad performance on TP will result in losses, while bad performance on FP will result in wasted capacity. Since the dataset is highly biased towards non-handover events, we can't rely on accuracy for measuring the performance of a classifier. Instead, the performance is measured as the area under the curve (AUC) of the receiver operating characteristics (ROC) curve. To generate the ROC curve, 33-fold cross validation is performed with 100 values for θ between 0 and 1, and the mean number of TP and FP is calculated for each value and represents a point in the ROC curve. The rationale for picking 33 folds is that each testing fold consists of data from one trip, and the rest of the trips are used for training. This mimics reality, where data from all previous trips is available for training the model and the resulting model is then tested on the

current trip. The mean number of TP over all testing folds is used in the ROC curve and is calculated as follows

$$\overline{TP} = \frac{\sum_{i=0}^{N_{trips}} \frac{TP_i}{TP_i+FP_i}}{N_{trips}-1} \tag{6.6}$$

where TP_i and FP_i are the TP and FP from the iteration of the cross-validation over trip i and N_{trips} is the total number of trips in the dataset. The mean number of FP is calculated in a similar manner

$$\overline{FP} = \frac{\sum_{i=0}^{N_{trips}} \frac{FP_i}{TP_i+FP_i}}{N_{trips}-1} \tag{6.7}$$

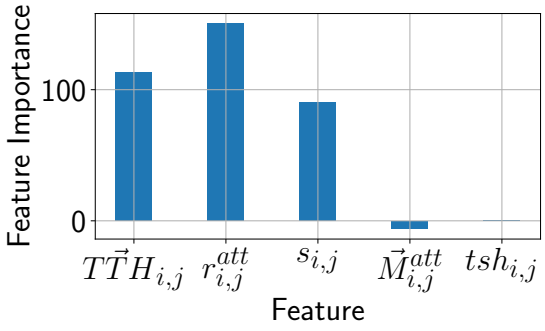
6.3.5 Classifiers

We tested 2 learning approaches for handover prediction: artificial neural network (ANN) and logistic regression (LR). Each approach was tested with three different set of features:

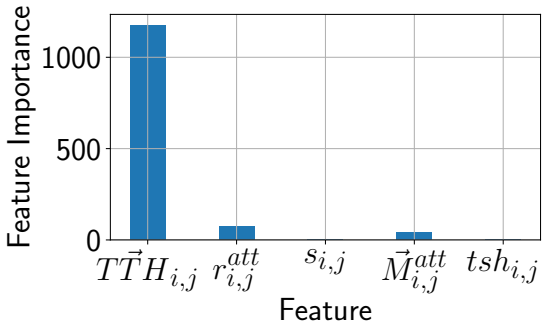
- RSSI-only (C_{RSSI}) – This feature set includes only the RSSI of the currently attached AP i.e. $\vec{X}_{i,j} = [r^{att}_{i,j}]$.
- Baseline features ($C_{Baseline}$) – This set includes the baseline features without any feature engineering i.e. $\vec{X}_{i,j} = [r^{att}_{i,j}, tsh_{i,j}, \vec{M}^{att}_{i,j}, s_{i,j}]$
- Enhanced features ($C_{Enhanced}$) – This is an enhanced feature list with the feature engineering steps including the normalization and random sampling. Additionally, features that are un-important for the specific classifiers are removed using feature importance permutation, which is described in more detail below.

Feature selection based on feature importance

In order to eliminate features that are either useless or have a negative impact on the performance of the classifiers, the importance of each feature is measured using the permutation feature importance method and un-important features are removed from the final set. To do this, the classifier is trained on a multiple versions of the training dataset where each version has the values for one feature randomly shuffled. The performance of each classifier version is compared to the performance on the normal classifier (i.e. without any features shuffled) and if the permuted version produces worse performance, then the feature is important, otherwise it is not. To measure performance of each classifier version, the AUC of the ROC curve is used. Fig 6.6 shows the feature importance score



(a) Artificial Neural Network



(b) Logistic Regression

Figure 6.6: Figure shows feature importance score for (a) Artificial Neural Network and (b) Logistic Regression calculated using the feature importance permutation approach

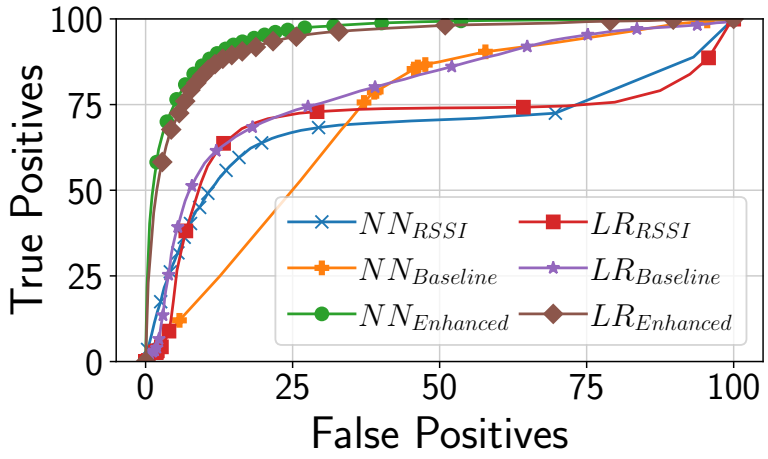
for the ANN and the LR. Based on the results, the extended feature set for ANN will include all features except $\vec{M}^{att}_{i,j}$ and $s_{i,j}$ i.e. $\vec{X}_{i,j} = [r^{att}_{i,j}, tsh_{i,j}, T\vec{T}H_{i,j}]$, while the feature set for the LR will include all features except $r^{att}_{i,j}$ and $s_{i,j}$ i.e. $\vec{X}_{i,j} = [tsh_{i,j}, \vec{M}_{i,j}, T\vec{T}H_{i,j}]$

Hyper parameter tuning for ANN

The ANN is configured to use the logistic activation function, ADAM Solver, cross-entropy cost function, L2 regularization with a coefficient of 0.1, and one hidden layer with 20 nodes. The configuration values were selected using the k-fold cross validation scheme discussed in section 6.3.4.

Hyper parameter tuning for LR

The LR is configured to use the Limited-memory BFGS solver and L2 regularization with a 0.2 coefficient. The configuration values were selected using the k-fold cross validation scheme discussed in section 6.3.4.



(a)

Figure 6.7: ROC curve showing false positives and true positives at different values for the decision threshold, θ, for three Artificial Neural Network classifiers NN_{RSSI}, NN_{BL} and NN_{Ext} trained on the RSSI only feature set, the baseline feature set and the extended feature set respectively, and three Logistic Regression classifiers LR_{RSSI}, LR_{BL} and LR_{Ext} trained on the same three feature sets.

6.3.6 Evaluation

Implementation

The evaluation was implemented in Python using Sickit learn 0.24.1.

Results

Fig 6.7 shows the ROC curve resulting from evaluating the 6 classifiers described in the previous section at 100 different values of θ. We can see that the RSSI only classifiers, LR_{RSSI} and NN_{RSSI}, performed the worst and were fairly close in performance. The NN baseline classifier, NN_{BL}, performed worse than the LR baseline classifier, LR_{BL}, as it may have been more affected by the imbalance of the training dataset. The two best classifiers, LR_{Ext} and NN_{Ext}, performed much better than the rest and were fairly close to each other, with the NN_{Ext} having slightly better performance. Based on the results, we will use the NN_{Ext} classifier in the implementation of the smart replication system.

6.4 Smart Replication System

The proposed Smart Replication system combines multi-path replication based on handover prediction and multi-path load balancing in order to minimize handover losses using the least possible redundancy while fully utilizing the leftover capacity for non-redundant traffic. Figure 6.8 shows the system architecture.

6.4.1 Architecture

The system is implemented using the VPN approach discussed in 3. The VPN client is placed in the on-board gateway and server is placed inside the network-side gateway. Both transparently re-route all traffic exchange between the train and the ground through the VPN tunnel. The VPN client includes a handover prediction module that monitors the WLAN client logs and uses the Neural Network prediction model described in the previous section to predict handovers. An overview of the architecture is shown in figure 6.8,.

6.4.2 Load Balancing

The load balancing module assigns application flows to one of the available WiFi links. Each application flow is identified by the IP 5-tuple (source IP, destination IP, source port, destination port and transport protocol). The link assignment is done in a weighted fashion with each class of application traffic having its unique weight. A weight reflects an application's expected throughput. For example VOIP, CCTV and CBTC are three different traffic classes, with CCTV having the highest weight since it has the highest expected throughput. A flow is associated with a traffic class by consulting the policy database, which contains mappings from any combination of the IP 5-tuple to a weight. The goal of the load balancer is to distribute the flows across the links to minimize the discrepancy in weight assigned between links. This will result in approximately the same amount of data being transmitted over each link.

The assumptions behind the load balancing approach are two fold:

- all the on-board WiFi links will experience approximately the same capacity profile over the duration of the trip since they traverse the same space and associate with similar APs.

- the expected throughput of each traffic class is easily defined beforehand given how the applications are configured. This is a reasonable assumption in the case of real-time traffic, such as CCTV, VOIP or CBTC. For example, the throughput of a CCTV camera can be accurately estimated by knowing the configured resolution, Frames Per Second (FPS) and bit rate.

6.4.3 Handover Prediction

The handover prediction module performs two main actions. First, it downloads logs from all the available WiFi clients via SSH. Second, it parses the logs to extract the features

and execute the previously trained model to derive a prediction for the handover. The module exposes a simple function for polling the handover probability for each link.

6.4.4 Smart Replication

Before each packet is transmitted, the smart replication module polls the handover prediction module for the current probability of handover over the packet's assigned link. It consults the policy database for the probability threshold, θ, associated with the flow the packet belongs to. In the case that the probability returned by the handover prediction process exceeds the policy associated with the flow, the VPN client replicates the packet over one of the other links. If more than one link is available, i.e. the train has more than 2 links, it sorts the available links by assigned weight, and then picks the first link where the current handover probability associated with the link is below the threshold required by the application, as defined in the policy database. If no available links fulfill that condition, then it picks the link with the least weight assigned to it.

Since the VPN server does not perform its own handover prediction, is notified of upcoming handovers through the Handover Probability field in the protocol header as shown in Fig 6.9 available in every packet received from the client. The value in this field is in the range 1-100 and represents a probability of a handover on the link it was received on in the next α seconds. This field is assigned the same handover probability value associated with the link by the client at the moment of transmitting the packet. If the client has not transmitted any packet over a link in over 50 milliseconds, it will transmit an empty packet in order to update the server with the handover probability. The rest of the smart replication process is identical to the client.

6.5 Experimental Setup

The evaluation environment was created inside a Dell XPS 15 9500 with 32 Gbyte of ram and an Intel Core i7-10750H 2.60GHz CPU with 6 cores and 12 threads. The VPN client, VPN server and the emulator were hosted within the main machine, and communicated using local host. The on-board environment and the network-side environment were hosted within two QEMU [77] virtual machines. The WiFi links were emulated using the same custom emulator previously described in 5.5.2.

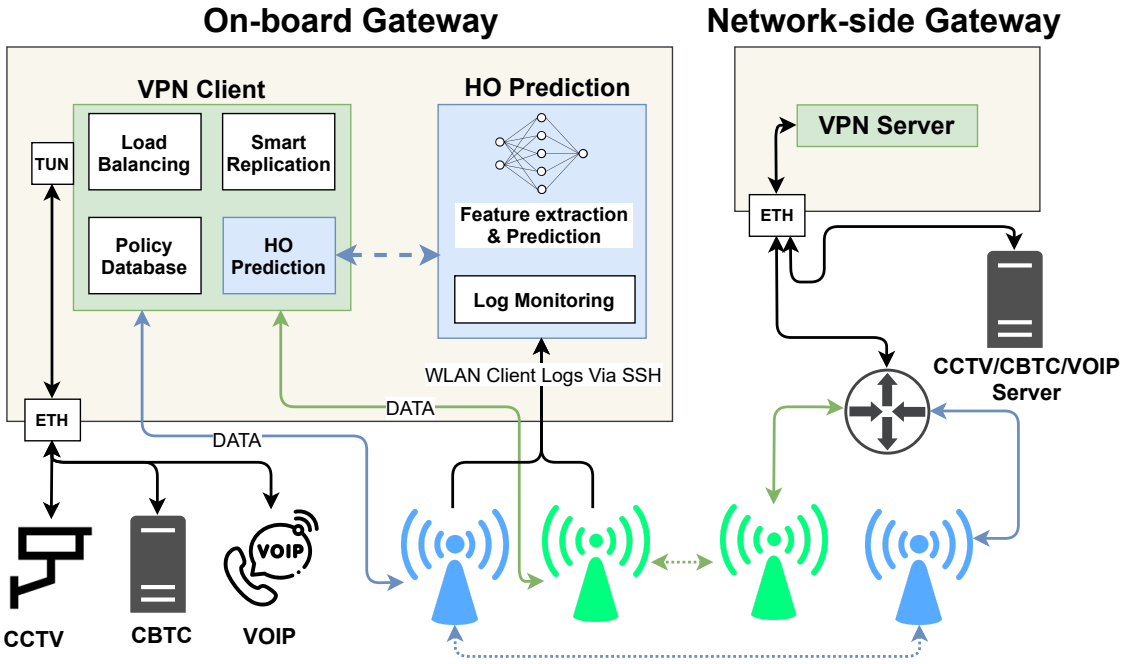


Figure 6.8: Smart Replication System Architecture

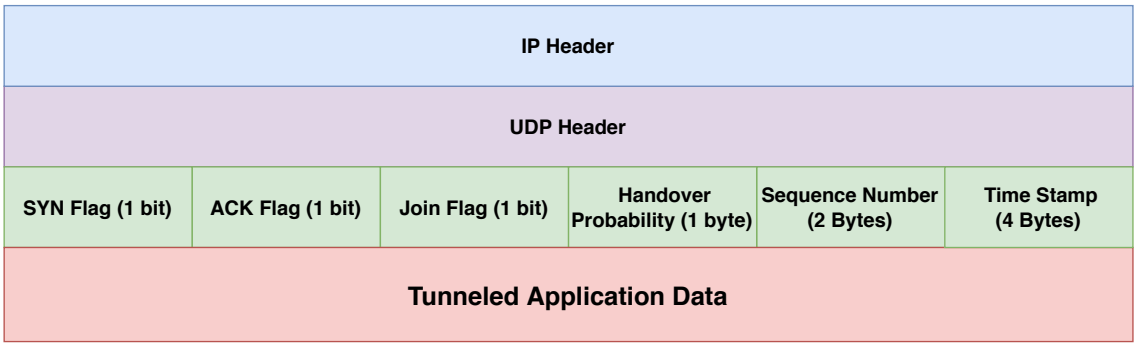


Figure 6.9: VPN Protocol Header Structure

6.5.1 Emulator

We used the custom network emulator described in 5.5.2 to emulate, in real-time, the capacity experienced by a WiFi link during a trip using capacity traces collected from a production system. Additionally, the emulator was modified to create a time-truncated version of the WiFi client log from that trip that is polled by the handover prediction module, such that it produces a prediction that matches the emulation. The emulator exposes an API for monitoring a truncated version of the WLAN client log for the trip currently being emulated. The truncation happens based on the current timestamp of the emulation in order to provide the same log that would appear at that point in the real trip. The handover prediction process utilizes the API to obtain the truncated version of the log, and uses it to predict handovers as it would in a real trip.

6.5.2 Evaluation configurations

We evaluated the performance of the smart replication system using 5 different values for the θ parameter: 0,0.3,0.5,0.8 and 1. When $\theta = 1$, the system will never replicate packets, and when $\theta = 0$, the system will replicate every packet. With all configurations, the load balancer described in the previous section is used.

6.5.3 Application traffic

Since we're concerned with mission-critical real-time traffic in T2G, we evaluated the performance of VOIP calls and CCTV transmission.

VOIP setup

We used the SIPP [88] performance testing tool to create multiple parallel VOIP flows that are replaying a PCAP capture from a previous VOIP call we recorded. The performance of the system was evaluated with 5, 10 and 20 parallel VOIP calls. For each of those configurations, 5 separate experiments were performed, on 5 different trips, each lasting for 30 minutes.

CCTV setup

We used FFServer [18] to create multiple parallel RTP video streams that replay Big Buck Bunny [6] with three different bit rates: low bit rate (LBR) at 10 mbit/s, medium bit rate (MBR) at 15 mbit/s, and high bit rate (HBR) at 20 mbit/s. For each configuration, 5 separate experiments were performed, on 5 different trips, each lasting for 30 minutes.

6.5.4 Evaluation metrics

We collected three metrics in the experiments: loss rate, redundancy rate and latency.

Loss

We measure losses by having a packet dump on each of the virtual interfaces connecting the QEMU VMs to the host machine and analyzing the dump at the end of experiment. Since both VOIP and CCTV are RTP traffic, we were able to track the sequence number on the packets on both virtual interfaces to verify if packets with that sequence number were delivered successfully or not.

Latency

Similar to loss, we measure latency by using information from the packet dump collected. We calculated the difference in timestamps between packets with the same sequence numbers at both virtual interfaces. Since both packet dumps were running on a single host machine, there were no issues with synchronization and we can simply subtract the timestamps to calculate the latency.

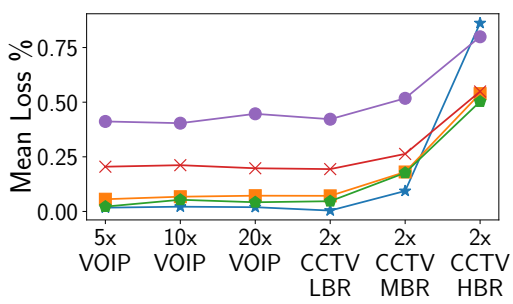
Redundancy

For the redundancy, we simply observed how many packets beyond what the originated from the VOIP/CCTV sender were sent by the VPN client. All the extra packets are considered redundancy.

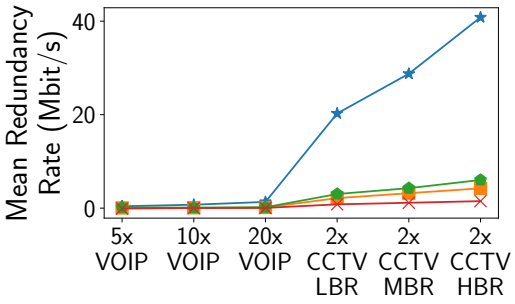
6.5.5 Evaluation Results

Fig 6.10 shows measurements of loss, redundancy and latency for all the schemes with all traffic types.

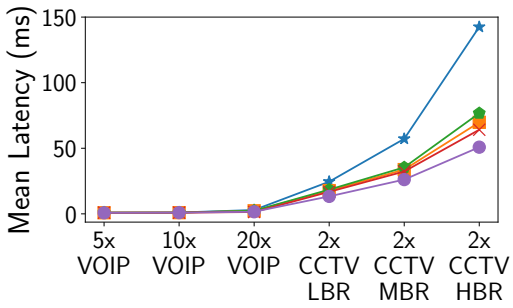
From Fig(a), it can be seen that mean loss for all schemes across all traffic types. The mean loss for $\theta \leq 0.5$ is very similar (¡0.1%) when handling 5, 10 and 20 VOIP calls, and the 2x LBR CCTV video, and were all significantly lower than when $\theta \geq 0.8$. In the 2x CCTV MBR CCTV video test case, all schemes have increased loss due to increased congestion loss. However, for the values of $\theta \leq 0.5$, the increase was higher due to increased number of congestion losses caused by the higher level of redundancy used. This pattern becomes clear in the 2x HBR CCTV test case as the losses in the full replication configuration ($\theta = 0$) exceed the no replication configuration ($\theta = 1$), and the losses of the $\theta = 0.8$ configuration is equal to the $\theta = 0.5$ and $\theta = 0.3$ as the congestion



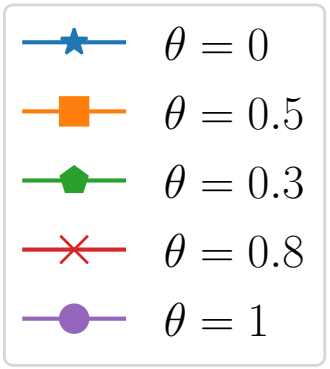
(a) Loss



(b) Redundancy



(c) Latency



(d) Legend

Figure 6.10: (a) mean loss rate vs θ and traffic load. The x-axis shows the traffic load used: 5,10 and 20 parallel VOIP calls, and 2 CCTV flows at 10 Mbit/s each (2x CCTV LBR), 2 CCTV flows at 15 Mbit/s each (2x CCTV MBR) and 2 CCTV flows at 20 Mbit/s each (2x CCTV MBR). (b) mean redundancy rate of replication and smart replication schemes. (c) mean latency rate.

loss and the handover loss balance out. It can be seen that as the application throughput gets closer to the capacity of the link, higher values of θ perform better.

Fig(b) shows mean rate of redundancy for $\theta \leq 0.8$, which are the configurations that use redundancy. In the VOIP test cases, the redundancy from all schemes is fairly small as the VOIP traffic is very light. In the CCTV test cases, the redundancy from the $\theta \geq 0.3$ configurations is less than 10% of the full replication configuration (i.e. $\theta = 0$).

Fig(c) shows the mean latency for all configurations. We can see that the latency overhead caused by replication in all configurations was negligible in all VOIP test cases and in the 2x LBR CCTV case. However, in the remaining cases the latency overhead of the full replication configuration ($\theta = 0$) grows significantly, reaching over 100 ms in the 2x HBR CCTV case. Meanwhile, in the $\theta \geq 0.3$ configuration the overhead is around 5-10 ms.

Overall, it can be seen that in that in all test scenarios, smart replication offers similar or better loss reduction as full replication while providing significantly less redundancy and lower latency. At lower throughput levels, smart replication with $\theta = 0.3$ achieves 91% of the loss reduction provided by simple replication on average while using only 13% of the redundancy and experiencing 20% less latency. As throughput grows relative to capacity, full replication (i.e. $\theta = 0$) causes more congestion losses than it reduces handover losses, making the scheme counterproductive. On the other hand, at $\theta \leq 0.8$ maintains around 34% loss reduction compared to no replication (i.e. $\theta = 1$), but the higher value of θ is able to achieve the same loss reduction with less redundancy, indicating that as throughput grows a higher value of θ performs better as it induces less congestion loss.

6.6 Related work

There have been multiple works investigating ways of minimizing the impact of WLAN handovers, both in the high and low mobility environments. We have identified four main categories in the literature. The first category aims to mitigate handovers without attempting to predict them [52][54]. The second category aims to improve the handovers in WLAN by devising better ways of triggering them [48][16][95]. Those approaches are naturally harder to deploy as they require modifying the WLAN clients. The third category are works that combine better handover triggers with cross-layer optimization [96][23]. The final category, and the closest to our work, are the ones that aim to predict handover, rather than trigger them, and also perform cross-layer optimization [35][20][45].

6.6.1 Handover mitigation without prediction

In two of the reviewed approaches authors employed constant redundancy to minimize losses from handovers [52][54].

In [52], authors designed a new multipath protocol RMPTCP based on MPTCP [79] that is better adapted for T2G. RMPTCP replicates all packets by default across all available paths. Congestion control is removed from the subflow level and kept only at the multipath level. TCP re-transmissions are modified such that they only happen if a packet timeout occurs on all the subflows.

In [54], authors propose a redundant mode for MPTCP to be used to transmit data from Supervisory Control and Data Acquisition (SCADA) systems used in trains. The SCADA traffic is latency sensitive and so the authors propose some tuning of MPTCP to make it more friendly to real-time traffic. Amongst the optimizations proposed is turning off the Nagle [64] algorithm, and effectively eliminating re-transmissions by preventing fast re-transmissions and placing a minimum limit of 64 seconds on return-trip timeout (RTO) retransmissions.

6.6.2 Improving handover triggers

Three of the reviewed works dealt with improving handover triggers [48][16][95]. In [48] and [16] authors triggered handovers by determining when the RSSI of the currently associated AP will fall below a certain pre-determined threshold. In [48] they used a weighted running average of the RSSI and in [16] they used a neural network with a time-sliding window of the RSSI as input. In [95] authors attempted to address the problem with RSSI variability causing multiple unnecessary handovers, referred to as hysteresis, by relying on link quality rather than RSSI for prediction.

6.6.3 Improving handover triggers with cross-layer optimization

Two of the reviewed works combined improving handover triggers with cross-layer optimizations [96][23].

In [96], authors created a model to predict handovers and use that prediction to lower the bitrate of streamed video before handovers in order to reduce losses and maintain uninterrupted playback. To predict handovers, they compared the double exponential weighted moving average (DEWMA) of the RSSI of the currently associated AP to one another AP, which and predicted a handover when the two DEWMA crossed.

In [23], authors created a video streaming app that predicts handovers with low granularity, up to 30 seconds, in order to fetch content ahead of time to ensure that no pauses occur during the outage caused by the handover. The predictor collects measurements from the user's device, such as velocity, RSSI, location and direction. It sent this information to a centralized server that hosts a database of historical mobility patterns, which it will then use to try to predict the future location of the device. Once the future location is determined, the server used connectivity map to predict outages based on the location. The server then transmitted the prediction back to the client to decide if it needs to pre-fetch data.

6.6.4 Handover prediction and cross-layer optimization

Three of the works reviewed combined handover prediction and cross-layer optimization [45][35][20].

In [45] authors designed a reactive approach to predicting handovers during VOIP calls. The approach relies on the rate of WLAN MAC layer retransmissions over a window of time. If a high rate of retransmissions is detected, the proposed system will replicate the traffic over all other available WLAN links. If one of the other available links experiences less MAC layer re-transmissions compared to the current link, the VOIP traffic will be shifted over to that link and the replication will stop.

In [35] authors devised a learning-based approach using a neural network with multiple inputs from smartphone sensors, including RSSI, to predict whether a WiFi handover will occur within 15 seconds. When a handover is predicted, the MPTCP stack would establish a connection over the LTE and use deploy the MPTCP's redundancy scheduler, which will replicate traffic across all links. In [20], authors take the same general approach with handover prediction using a neural network, but they enlarge the scope of inputs used for the prediction to any information that is available beyond sensors, such as link throughput or loss rate.

None of the works reviewed dealt with the challenge of predicting handovers in the underground T2G environment. In particular, the high granularity required of prediction required due to the speed, the more chaotic radio environment due to the tunnels, and the over-the-top deployment inside existing infrastructure.

6.7 Conclusion

In this chapter, we proposed an over the top approach for minimizing packet loss due to handovers in underground trains equipped with multiple WiFi clients by predicting handovers and replicating packets just before they occur. We devised a learning based handover prediction model based on features extracted exclusively from the WiFi client log. Evaluations of the model on traces from a production system showed that it is able to predict handovers with high accuracy. We integrated our prediction model into a simple adaptive replication scheme and experimentally evaluated variants of the scheme together with no replication and a full replication scheme. Based on the results, we concluded that the smart replication offered a similar or better loss reduction compared to the full replication scheme in all test cases, while utilizing significantly lower redundancy and maintaining lower latency.

Chapter 7

Conclusion and future work

7.0.1 Conclusion

In this thesis we uncovered issues with the state-of-the-art in Transport-layer Multi-connectivity (TLMC) and proposed a collection of algorithms and system architectures that enhance the deployability and performance of TLMC in multiple real-world applications.

In Chapter 3, we were motivated by enabling the new Access Traffic Steering, Switching & Splitting (ATSSS) [89] feature in 5G release 16, which allows an MNO to simultaneously manage user access to 3GPP as well as non-3GPP wireless access technologies, to avail of a higher-layer steering function for enforcing its policies. We observed that even though MPTCP was selected by the 3GPP as a higher-layer steering function for ATSSS, it is still not widely adopted in smart devices due to its kernel based implementation. We instead proposed a new user-space approach for deploying TLMC that is more readily deployable and extensible compared to a kernel implementation, requires no changes to the user device, and is backward compatible with all existing applications and device. The approach is based on creating a UDP based user-space multi-path transport protocol implementation that uses the built-in VPN framework, available on all major mobile operating systems, to transmit all user traffic using TLMC on the path between the user device and a network edge server. We created an implementation of this approach in Android and evaluated its performance and CPU overhead versus the kernel network stack. Results showed that performance is comparable to the kernel and the overhead is minor.

Chapter 4, investigated the task of multi-path scheduling over multiple wireless links, which typically have time-varying delay. The challenge here is that packets sent over these

links frequently arrive out of order and need to be buffered at the receiver to allow in-order delivery, leading to head of line blocking and introducing substantial delay. We make the observation that the requirement of multi-path scheduling is to deliver application layer objects (web pages, images, video frames etc) with low latency and hence it is the object delay rather than the packet delay that is important. The implication of this is that a scheduler can make use of links with fluctuating per-packet delay as long as they allow the entire object, which typically consists of many packets, to be delivered sooner. We proposed a Stochastic Object-ware Scheduler (SOS), a multi-path scheduler that takes into consideration the size of application layer objects and the uncertainty of the packet delay in order to deliver objects with a reliable bound on delay. We then extended SOS to utilize Forward Error Correction (FEC) in the form of linear block codes that match the object to allow it to opportunistically exploit periods of fluctuating path delay to further reduce object delivery delay at the cost of slightly reduced throughput capacity. We created an implementation of SOS that interfaces with CWND based congestion control in order to optimize scheduling when an object partition is split across multiple CWNDs. We performed extensive evaluations using simulations and trace-driven emulations and established that SOS and SOS-FEC significantly improve the 95% delay and mean object delivery delay compared to state of the art schedulers.

Chapter 5 analyzed the performance of MPTCP, when used inside Multi-WAN routers (MWR) to aggregate multiple WAN/Internet connections, and highlighted issues with both architectural variants used in MPTCP MWR (i.e. proxying and tunneling). In particular, we showed that the large number of parallel uncoordinated MPTCP connections created in the proxying variant result in low link utilization, sub-optimal multi-path scheduling and high loss rate by using measurements collected from a production train-to-ground (T2G) system that relies on a proxy MPTCP MWR. In the case of the tunneling variant, we showed that it suffers from a high rate of spurious re-transmissions as result of stacking two reliability layer (TCP and MPTCP) and from cross-flow HoL blocking by using a trace-driven emulation based on the same T2G measurements. We proposed a new multi-path solution more suited to MWR, called BOOST, which multiplexes TCP and UDP flows over a single persistent multi-path connections, allowing it to avoid issues present in both architectural variants of MPTCP. BOOST also takes a hybrid approach to multi-path scheduling. Namely, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity, thereby improving efficiency and robustness. Evalua-

tions showed that BOOST provides better throughput, lower losses, and retransmissions while requiring less memory compared to both MPTCP variants.

In the final chapter, we considered the task of transporting mission-critical traffic, with strict QoS requirements for high availability and low packet loss, in underground trains that are equipped with multiple WiFi backhauls. Instead of simple replication approach taken in other studies to mitigate losses from frequent handovers, we take the approach of predicting handovers and only replicating packets shortly before they happen to reduce amount of redundancy needed and free up network resources. It turns out that accurate prediction of handovers is surprisingly tricky since, based on extensive measurements from trains operating in a production environment, the WiFi APs/clients exhibit complex handover behaviour due to variations in train speed, wait time at the station and signal strength fluctuations. To address this we proposed a novel neural network predictor, based only on features extracted from the WiFi client log, that predicts when a handover is about to occur with high accuracy. We used this predictor as a basis for an adaptive packet replication scheme and evaluated its performance using a trace-driven emulation. Evaluation with light traffic loads showed that all variants of the proposed scheme reduced losses from handovers nearly as much as the simple replication approach while using a small fraction of the redundancy. In heavier traffic loads, the proposed scheme outperformed the simple replication scheme even in terms of handover loss reduction as it suffered significantly less from self-induced congestion losses caused by the added redundancy in the simple replication approach.

7.0.2 Future work

The following are a number of problems that extend from the work done in this thesis

Uplink offloading for network optimization

In chapter 3, we proposed an approach for deploying TLMC that can be centrally managed by an MNO for network optimization, and that would allow it to leverage asymmetric routing for off-loading uplink transmission entirely to one of the links. Given the potential for this technique to improve downlink transmission by removing uplink contention from one link, increasing uplink aggregation, or improving utilization of air time, it is worth exploring how a network of multiple users can be optimized by using assymmetric routing.

Real-time multi-path scheduling

In chapter 4, we proposed SOS, which aimed to minimize latency for application layer objects. This is an ideal strategy for best effort applications such as web browsing and buffered video streaming. However, real-time applications, such as VOIP or video conferencing, transmit objects, such as video frames, which have interactivity deadlines after which they will be discarded when delivered. The goal of these applications is to deliver the objects within the deadline with high reliability. As such it is useful to design a real-time multi-path scheduler that, instead of minimizing delay with pre-defined reliability, maximizes the reliability of delivering an application layer of object within a pre-defined deadline. This scheduler can further utilize FEC to increase the probability of delivery of the object within the deadline.

Optimal trade-off between redundancy and handover loss in underground train-to-ground communication

In chapter 6, we showed that simple replication is a counter-productive strategy for mitigating handover losses when transmitting throughput heavy traffic since it induces more congestion losses than it reduces handover losses. The handover prediction model proposed, which acted as a basis for the adaptive replication scheme, relied on a probability threshold parameter, θ, to decide when to replicate traffic. Lower values of θ resulted in more redundancy being used but mitigated more handover losses, while higher values of θ used less redundancy and had more handover losses. In order to minimize both congestion loss and handover loss, it is worth investigating how to compute an optimal value of *theta* given the capacity of a link and the throughput required for the traffic.

Bibliography

[1] *ACKSYS Communications & Systems*. Acksys. URL: https://www.acksys.fr/en/ (visited on 12/28/2020).

[2] A. Al-Najjar et al. "Flow-based load balancing of web traffic using OpenFlow". In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017 27th International Telecommunication Networks and Applications Conference (ITNAC). Nov. 2017, pp. 1–6. DOI: 10.1109/ATNAC.2017.8215411.

[3] *Android 5.0 APIs*. Android Developers. URL: https://developer.android.com/about/versions/android-5.0 (visited on 03/07/2021).

[4] J. D. Beshay et al. "Link-Coupled TCP for 5G networks". In: *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS). June 2017, pp. 1–6. DOI: 10.1109/IWQoS.2017.7969170.

[5] G. Bianchi. "Performance analysis of the IEEE 802.11 distributed coordination function". In: *IEEE Journal on Selected Areas in Communications* 18.3 (Mar. 2000), pp. 535–547. ISSN: 1558-0008. DOI: 10.1109/49.840210.

[6] *Big Buck Bunny*. URL: https://peach.blender.org/ (visited on 03/01/2021).

[7] C.J. Bovy et al. "Analysis of end-to-end delay measurements in Internet". In: Jan. 1, 2002.

[8] K. Chebrolu and R. Rao. "Communication using multiple wireless interfaces". In: *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609)*. 2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609). Vol. 1. Mar. 2002, 327–331 vol.1. DOI: 10.1109/WCNC.2002.993516.

[9] *Cisco VNI predicts bright future for Wi-Fi towards 2022*. Wi-Fi NOW Global. Feb. 22, 2019. URL: https://wifinowglobal.com/news-and-blog/new-cisco-vni-numbers-predict-bright-future-for-wi-fi-towards-2022/ (visited on 02/04/2021).

[10] *Consumer Intelligence Research Partners*. Consumer Intelligence Research Partners. URL: https://www.cirpllc.com (visited on 03/07/2021).

[11] Irfaan Coonjah, Pierre Clarel Catherine, and K. M. S. Soyjaudah. "An Investigation of the TCP Meltdown Problem and Proposing Raptor Codes as a Novel to Decrease TCP Retransmissions in VPN Systems". In: *Information Systems Design and Intelligent Applications*. Ed. by Suresh Chandra Satapathy et al. Advances in Intelligent Systems and Computing. Singapore: Springer, 2019, pp. 337–347. ISBN: 9789811333293. DOI: 10.1007/978-981-13-3329-3_32.

[12] Irfaan Coonjah, Pierre Clarel Catherine, and K. M. S. Soyjaudah. "An Investigation of the TCP Meltdown Problem and Proposing Raptor Codes as a Novel to Decrease TCP Retransmissions in VPN Systems". In: *Information Systems Design and Intelligent Applications*. Ed. by Suresh Chandra Satapathy et al. Advances in Intelligent Systems and Computing. Singapore: Springer, 2019, pp. 337–347. ISBN: 9789811333293. DOI: 10.1007/978-981-13-3329-3_32.

[13] Quentin De Coninck and Olivier Bonaventure. "Multipath QUIC: Design and Evaluation". In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. CoNEXT '17. New York, NY, USA: Association for Computing Machinery, Nov. 28, 2017, pp. 160–166. ISBN: 978-1-4503-5422-6. DOI: 10.1145/3143361.3143370. URL: https://doi.org/10.1145/3143361.3143370 (visited on 01/21/2021).

[14] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC5246. RFC Editor, Aug. 2008, RFC5246. URL: https://www.rfc-editor.org/info/rfc5246 (visited on 06/11/2021).

[15] Nandita Dukkipati et al. "An Argument for Increasing TCP's Initial Congestion Window". In: *Computer Communication Review* 40 (June 22, 2010), pp. 26–33. DOI: 10.1145/1823844.1823848.

[16] M. Feltrin and S. Tomasin. "A Machine-Learning-Based Handover Prediction for Anticipatory Techniques in Wi-Fi Networks". In: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. 2018 Tenth International

Conference on Ubiquitous and Future Networks (ICUFN). July 2018, pp. 341–345. DOI: 10.1109/ICUFN.2018.8436796.

[17] S. Ferlin et al. "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks". In: *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. 2016 IFIP Networking Conference (IFIP Networking) and Workshops. May 2016, pp. 431–439. DOI: 10.1109/IFIPNetworking.2016.7497206.

[18] *ffserver – FFmpeg*. URL: https://trac.ffmpeg.org/wiki/ffserver (visited on 03/01/2021).

[19] A. Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC6824. RFC Editor, Jan. 2013, RFC6824. URL: https://www.rfc-editor.org/info/rfc6824 (visited on 06/11/2021).

[20] Alexander Frömmgen et al. "Poster: Use your Senses: A Smooth Multipath TCP WiFi/Mobile Handover". In: Sept. 7, 2015. DOI: 10.1145/2789168.2795171.

[21] *ftrace - Function Tracer*. URL: https://www.kernel.org/doc/Documentation/trace/ftrace.txt (visited on 08/15/2020).

[22] A. Garcia-Saavedra, M. Karzand, and D. J. Leith. "Low Delay Random Linear Coding and Scheduling Over Multiple Interfaces". In: *IEEE Transactions on Mobile Computing* 16.11 (Nov. 2017), pp. 3100–3114. ISSN: 1536-1233. DOI: 10.1109/TMC.2017.2686379.

[23] Joost Geurts et al. "Transparent Handover Using WiFi Network Prediction for Mobile Video Streaming". In: *Advances in Intelligent Systems and Computing* 353 (Jan. 1, 2015), pp. 937–946. ISSN: 978-3-319-16485-4. DOI: 10.1007/978-3-319-16486-1_93.

[24] Doru-Cristian Gucea and Octavian Purdila. *Shaping the Linux kernel MPTCP implementation towards upstream acceptance*. undefined. 2015. URL: /paper/Shaping-the-Linux-kernel-MPTCP-implementation-Gucea-Purdila/1b0eec662775ddbc724249e829d3a (visited on 03/12/2021).

[25] Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. "Deploying safe user-level network services with icTCP". In: *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*. OSDI'04. USA: USENIX Association, Dec. 6, 2004, p. 22. (Visited on 03/12/2021).

[26] Sangtae Ha, Injong Rhee, and Lisong Xu. “CUBIC: a new TCP-friendly high-speed TCP variant”. In: *Operating Systems Review* 42 (July 1, 2008), pp. 64–74. DOI: 10.1145/1400097.1400105.

[27] Harald Haas. “LiFi is a paradigm-shifting 5G technology”. In: *Reviews in Physics* 3 (Nov. 1, 2018), pp. 26–31. ISSN: 2405-4283. DOI: 10.1016/j.revip.2017.10.001. URL: http://www.sciencedirect.com/science/article/pii/S2405428317300151 (visited on 02/04/2021).

[28] Bo Han et al. “MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath”. In: *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’16. New York, NY, USA: ACM, 2016, pp. 129–143. ISBN: 978-1-4503-4292-6. DOI: 10.1145/2999572.2999606. URL: http://doi.acm.org/10.1145/2999572.2999606 (visited on 11/23/2019).

[29] *Home | Phantom Auto - Enabling Autonomy Through Teleoperation*. Phantom Auto. URL: https://phantom.auto/ (visited on 02/04/2021).

[30] Michio Honda et al. “Rekindling network protocol innovation with user-level stacks”. In: *ACM SIGCOMM Computer Communication Review* 44.2 (Apr. 8, 2014), pp. 52–58. ISSN: 0146-4833. DOI: 10.1145/2602204.2602212. URL: https://doi.org/10.1145/2602204.2602212 (visited on 03/12/2021).

[31] *How does HTTP/2 solve the Head of Line blocking (HOL) issue*. URL: https://community.akamai.com/customers/s/article/How-does-HTTP-2-solve-the-Head-of-Line-blocking-HOL-issuelanguage=en-US (visited on 08/15/2020).

[32] Junxian Huang et al. “An in-depth study of LTE: effect of network protocol and application behavior on performance”. In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. SIGCOMM ’13. New York, NY, USA: Association for Computing Machinery, Aug. 27, 2013, pp. 363–374. ISBN: 978-1-4503-2056-6. DOI: 10.1145/2486001.2486006. URL: https://doi.org/10.1145/2486001.2486006 (visited on 06/11/2021).

[33] Per Hurtig, Wolfgang John, and Anna Brunstrom. “Recent Trends in TCP Packet-Level Characteristics”. In: Jan. 1, 2011, pp. 49–56.

[34] Jaehyun Hwang and Joon Yoo. “A Memory-Efficient Transmission Scheme for Multi-Homed Internet-of-Things (IoT) Devices”. In: *Sensors* 20 (Mar. 6, 2020), p. 1436. DOI: 10.3390/s20051436.

[35] Jonas Höchst et al. "Learning Wi-Fi Connection Loss Predictions for Seamless Vertical Handovers Using Multipath TCP". In: Oct. 1, 2019, pp. 18–25. DOI: 10.1109/LCN44214.2019.8990753.

[36] *IDC - Smartphone Market Share - OS*. IDC: The premier global market intelligence company. URL: https://www.idc.com/promo/smartphone-market-share (visited on 03/03/2021).

[37] *IEEE 802.11F-2003 - IEEE Trial-Use Recommended Practice for Multi-Vendor Access Point Interoperability Via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation*. URL: https://standards.ieee.org/standard/80211F-2003.html (visited on 01/03/2021).

[38] *IETF Proceedings*. URL: https://www.ietf.org/proceedings/93/mptcp.html (visited on 03/07/2021).

[39] *Improving Network Reliability Using Multipath TCP | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/foundation/urlsessionconfiguration/improving_network_reliability_using_multipath_tcp (visited on 02/04/2021).

[40] *Introducing Network.framework: A modern alternative to Sockets - WWDC 2018 - Videos*. Apple Developer. URL: https://developer.apple.com/videos/play/wwdc2018/715 (visited on 03/24/2021).

[41] *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool*. URL: https://iperf.fr/ (visited on 08/16/2020).

[42] *iptables(8) - Linux man page*. URL: https://linux.die.net/man/8/iptables (visited on 03/01/2021).

[43] A. Jurgelionis et al. "An Empirical Study of NetEm Network Emulation Functionalities". In: *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. 2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN). July 2011, pp. 1–6. DOI: 10.1109/ICCCN.2011.6005933.

[44] MEHMET Karakas. "Determination of network delay distribution over the internet". Master Thesis. The middle east technical university, 2003. URL: https://etd.lib.metu.edu.tr/upload/1223155/index.pdf.

[45] Shigeru Kashihara and Yuji Oie. “Handover management based on the number of data frame retransmissions for VoWLAN”. In: *Computer Communications*. Special Issue Concurrent Multipath Transport 30.17 (Nov. 30, 2007), pp. 3257–3269. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2007.01.014. URL: https://www.sciencedirect.com/science/article/pii/S0140366407000394 (visited on 03/01/2021).

[46] A. N. Krasilov et al. “Performance Evaluation of TCP Data Transmission in 5G mmWave Networks”. In: *Journal of Communications Technology and Electronics* 65.6 (June 1, 2020), pp. 735–740. ISSN: 1555-6557. DOI: 10.1134/S1064226920060194. URL: https://doi.org/10.1134/S1064226920060194 (visited on 03/09/2021).

[47] S. Kucera, K. Fahmi, and H. Claussen. “Latency As a Service: Enabling Reliable Data Delivery over Multiple Unreliable Wireless Links”. In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall). Sept. 2019, pp. 1–5. DOI: 10.1109/VTCFall.2019.8891579.

[48] K. Kumar, Pravin Angolkar, and Ramakrishnan Ramalingam. “SWiFT: A Novel Architecture for Seamless Wireless Internet for Fast Trains”. In: June 14, 2008, pp. 3011–3015. DOI: 10.1109/VETECS.2008.322.

[49] Adam Langley et al. “The QUIC Transport Protocol: Design and Internet-Scale Deployment”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’17. New York, NY, USA: Association for Computing Machinery, Aug. 7, 2017, pp. 183–196. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098842. URL: https://doi.org/10.1145/3098822.3098842 (visited on 08/15/2020).

[50] Yeon-sup Lim et al. “ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths”. In: *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’17 Abstracts. New York, NY, USA: ACM, 2017, pp. 33–34. ISBN: 978-1-4503-5032-7. DOI: 10.1145/3078505.3078552. URL: http://doi.acm.org/10.1145/3078505.3078552 (visited on 08/02/2018).

[51] J. Ling et al. “Enhanced capacity and coverage by Wi-Fi LTE integration”. In: *IEEE Communications Magazine* 53.3 (Mar. 2015), pp. 165–171. ISSN: 1558-1896. DOI: 10.1109/MCOM.2015.7060499.

[52] I. Lopez, M. Aguado, and E. Jacob. "End-to-End Multipath Technology: Enhancing Availability and Reliability in Next-Generation Packet-Switched Train Signaling Systems". In: *IEEE Vehicular Technology Magazine* 9.1 (Mar. 2014), pp. 28–35. ISSN: 1556-6080. DOI: 10.1109/MVT.2013.2295072.

[53] I. Lopez et al. "Exploiting redundancy and path diversity for railway signalling resiliency". In: *2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT)*. 2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT). Aug. 2016, pp. 432–439. DOI: 10.1109/ICIRT.2016.7588765.

[54] I. Lopez et al. "SCADA Systems in the Railway Domain: Enhancing Reliability through Redundant MultipathTCP". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015 IEEE 18th International Conference on Intelligent Transportation Systems. Sept. 2015, pp. 2305–2310. DOI: 10.1109/ITSC.2015.372.

[55] Feng Lu et al. "CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks". In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. HotMobile '15. New York, NY, USA: Association for Computing Machinery, Feb. 12, 2015, pp. 45–50. ISBN: 978-1-4503-3391-7. DOI: 10.1145/2699343.2699345. URL: https://doi.org/10.1145/2699343.2699345 (visited on 03/09/2021).

[56] *Mac OS X Developer Tools Manual Page For setsockopt(2)*. URL: https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man2/setsockopt.2.html (visited on 03/07/2021).

[57] R. Madan, A. Sampath, and N. Khude. "Enhancing 802.11 carrier sense for high throughput and QoS in dense user settings". In: *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*. 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC). Sept. 2012, pp. 253–259. DOI: 10.1109/PIMRC.2012.6362785.

[58] Jeffrey Mogul et al. "Unveiling the transport". In: *ACM SIGCOMM Computer Communication Review* 34.1 (Jan. 1, 2004), pp. 99–106. ISSN: 0146-4833. DOI: 10.1145/972374.972392. URL: https://doi.org/10.1145/972374.972392 (visited on 03/12/2021).

[59] *MPTCP Korea Telecom deployment*. URL: https://datatracker.ietf.org/doc/slides-93-mptcp-3/ (visited on 02/04/2021).

[60] *MultiPath TCP - Linux Kernel implementation : Main - Home Page browse*. URL: https://www.multipath-tcp.org/ (visited on 03/02/2021).

[61] *Multipath TCP for FreeBSD | FreeBSD Foundation*. URL: https://freebsdfoundation.org/project/multipath-tcp-for-freebsd/ (visited on 02/04/2021).

[62] *Multipath TCP solutions for better connectivity*. Tessares. URL: https://www.tessares.net/ (visited on 02/04/2021).

[63] *multipath-tcp/mptcp*. Jan. 28, 2021. URL: https://github.com/multipath-tcp/mptcp (visited on 02/04/2021).

[64] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC0896. RFC Editor, Jan. 1984, RFC0896. URL: https://www.rfc-editor.org/info/rfc0896 (visited on 06/11/2021).

[65] *Netdev | Netdev 2.2 - Session*. URL: https://netdevconf.info/2.2/session.html?tazaki-mptcp-talk (visited on 03/12/2021).

[66] *Network Extension | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/networkextension (visited on 03/05/2021).

[67] Binh Nguyen et al. "Towards understanding TCP performance on LTE/EPC mobile networks". In: *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*. AllThingsCellular '14. New York, NY, USA: Association for Computing Machinery, Aug. 22, 2014, pp. 41–46. ISBN: 978-1-4503-2990-3. DOI: 10.1145/2627585.2627594. URL: https://doi.org/10.1145/2627585.2627594 (visited on 03/09/2021).

[68] D. Ni et al. "Fine-grained Forward Prediction based Dynamic Packet Scheduling Mechanism for multipath TCP in lossy networks". In: *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*. 2014 23rd International Conference on Computer Communication and Networks (ICCCN). Aug. 2014, pp. 1–7. DOI: 10.1109/ICCCN.2014.6911726.

[69] *Outdoor Wireless Ethernet Radio Networks and Solutions*. Fluidmesh. URL: https://www.fluidmesh.com/ (visited on 12/28/2020).

[70] *PackageManager*. Android Developers. URL: https://developer.android.com/reference/android/content/pm/PackageManager (visited on 03/08/2021).

[71] G. Papastergiou et al. "De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives". In: *IEEE Communications Surveys Tutorials* 19.1 (2017), pp. 619–639. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2626780.

[72] Bahar Partov and Douglas J. Leith. "Experimental Evaluation of Multi-path Schedulers for LTE/Wi-Fi Devices". In: *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*. WiNTECH '16. New York, NY, USA: ACM, 2016, pp. 41–48. ISBN: 978-1-4503-4252-0. DOI: 10.1145/2980159.2980169. URL: http://doi.acm.org/10.1145/2980159.2980169 (visited on 08/02/2018).

[73] Parveen Patel et al. "Upgrading transport protocols using untrusted mobile code". In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. SOSP '03. New York, NY, USA: Association for Computing Machinery, Oct. 19, 2003, pp. 1–14. ISBN: 978-1-58113-757-6. DOI: 10.1145/945445.945447. URL: https://doi.org/10.1145/945445.945447 (visited on 03/12/2021).

[74] R. Poorzare and A. C. Augé. "Challenges on the Way of Implementing TCP Over 5G Networks". In: *IEEE Access* 8 (2020), pp. 176393–176415. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3026540.

[75] J. Postel. *Transmission Control Protocol*. RFC0793. RFC Editor, Sept. 1981, RFC0793. URL: https://www.rfc-editor.org/info/rfc0793 (visited on 06/11/2021).

[76] *Pre-installed apps: Samsung Galaxy S8*. T-Mobile Support. URL: https://www.t-mobile.com/support/devices/android/samsung-galaxy-s8/pre-installed-apps-samsung-galaxy-s8 (visited on 03/08/2021).

[77] *QEMU*. URL: https://www.qemu.org/ (visited on 03/01/2021).

[78] *RADWIN | Wireless Broadband, Mobility and Backhaul Solutions*. RADWIN. URL: https://www.radwin.com/ (visited on 12/28/2020).

[79] Costin Raiciu et al. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29. URL: http://dl.acm.org/citation.cfm?id=2228298.2228338 (visited on 08/02/2018).

[80] *RazorLink Smart Networking - SD-WAN with Hybrid WAN and WAN Optimisation*. Livewire Digital. URL: https://www.livewire.co.uk/products/razorlink/ (visited on 02/04/2021).

[81] *Release 17*. URL: https://www.3gpp.org/release-17 (visited on 02/22/2021).

[82] Swetank Kumar Saha et al. "MuSher: An Agile Multipath-TCP Scheduler for Dual-Band 802.11Ad/Ac Wireless LANs". In: *The 25th Annual International Conference on Mobile Computing and Networking*. MobiCom '19. New York, NY, USA: ACM, 2019, 34:1–34:16. ISBN: 978-1-4503-6169-9. DOI: 10.1145/3300061.3345435. URL: http://doi.acm.org/10.1145/3300061.3345435 (visited on 11/23/2019).

[83] G. Sarwar et al. "Mitigating Receiver's Buffer Blocking by Delay Aware Packet Scheduling in Multipath Data Transfer". In: *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. 2013 27th International Conference on Advanced Information Networking and Applications Workshops. Mar. 2013, pp. 1119–1124. DOI: 10.1109/WAINA.2013.80.

[84] T. De Schepper et al. "Software-defined multipath-TCP for smart mobile devices". In: *2017 13th International Conference on Network and Service Management (CNSM)*. 2017 13th International Conference on Network and Service Management (CNSM). Nov. 2017, pp. 1–6. DOI: 10.23919/CNSM.2017.8256043.

[85] *SD-WAN Connectivity Solutions for Mobility & Specialized Markets*. Peplink. URL: https://www.peplink.com/mobility-and-specialized-solutions/ (visited on 02/04/2021).

[86] *Service Name and Transport Protocol Port Number Registry*. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (visited on 03/08/2021).

[87] *Shadowsocks*. In: *Wikipedia*. Jan. 20, 2021. URL: https://en.wikipedia.org/w/index.php?title=Shadowsocksoldid=1001551254 (visited on 01/21/2021).

[88] *SIPp/sipp*. Feb. 28, 2021. URL: https://github.com/SIPp/sipp (visited on 03/01/2021).

[89] *Specification # 23.793*. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3254 (visited on 02/04/2021).

[90] Mario Spiezio et al. *Fast Radio Technologies For Uninterrupted Train To Track Side Communications*. EU, Mar. 27, 2018. URL: https://www.fasttracks.eu/wp-content/uploads/2018/03/deliverable-1-1-fast-tracks-wireless-networks-architecture.pdf (visited on 12/24/2020).

[91] *Starlink*. Starlink. URL: https://www.starlink.com (visited on 02/04/2021).

[92] R. Stewart. *Stream Control Transmission Protocol*. RFC4960. RFC Editor, Sept. 2007, RFC4960. URL: https://www.rfc-editor.org/info/rfc4960 (visited on 06/11/2021).

[93] *Train to Ground*. Nokia. URL: https://www.nokia.com/networks/solutions/train-to-ground/ (visited on 03/13/2021).

[94] *Transparent proxy support — The Linux Kernel documentation*. URL: https://www.kernel.org/doc/html/latest/networking/tproxy.html (visited on 03/24/2021).

[95] Tsungnan Lin, Chiapin Wang, and Po-Chiang Lin. "A neural network based context-aware handoff algorithm for multimedia computing". In: *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Vol. 2. Mar. 2005, ii/1129–ii/1132 Vol. 2. DOI: 10.1109/ICASSP.2005.1415608.

[96] Carlo Vallati, Enzo Mingozzi, and Carmine Benedetto. "Efficient handoff based on link quality prediction for video streaming in urban transport systems: Efficient handoff for video streaming in urban transport systems". In: *Wireless Communications and Mobile Computing* 16 (June 1, 2016). DOI: 10.1002/wcm.2684.

[97] Esme Vos. *Why Wi-Fi sucked at the Mobile World Congress: so what else is new?* MuniWireless: WiFi, hotspots, LTE. Feb. 18, 2011. URL: http://muniwireless.com/2011/02/18/why-wifi-sucked-at-the-mobile-world-congress/ (visited on 03/09/2021).

[98] *VPN*. Android Developers. URL: https://developer.android.com/guide/topics/connectivity/vpn (visited on 03/05/2021).

[99] *What is 5G New Radio (5G NR)*. URL: https://5g.co.uk/guides/what-is-5g-new-radio/ (visited on 02/22/2021).

[100] *Wi-Fi Alliance® introduces Wi-Fi 6 | Wi-Fi Alliance*. URL: https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-wi-fi-6 (visited on 02/22/2021).

[101] Alyssa Wilk et al. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. URL: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00 (visited on 08/02/2018).

[102] Y. Xing et al. “MPTCP Meets Big Data: Customizing Transmission Strategy for Various Data Flows”. In: *IEEE Network* 34.4 (July 2020), pp. 35–41. ISSN: 1558-156X. DOI: 10.1109/MNET.011.1900438.

[103] F. Yang, P. Amer, and N. Ekiz. “A Scheduler for Multipath TCP”. In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. 2013 22nd International Conference on Computer Communication and Networks (ICCCN). July 2013, pp. 1–7. DOI: 10.1109/ICCCN.2013.6614091.

[104] F. Yang, Q. Wang, and P. D. Amer. “Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP”. In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*. 2014 28th International Conference on Advanced Information Networking and Applications Workshops. May 2014, pp. 749–752. DOI: 10.1109/WAINA.2014.122.

[105] Changhe Yu et al. “QoS-aware Traffic Classification Architecture Using Machine Learning and Deep Packet Inspection in SDNs”. In: *Procedia Computer Science*. Recent Advancement in Information and Communication Technology: 131 (Jan. 1, 2018), pp. 1209–1216. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.04.331. URL: https://www.sciencedirect.com/science/article/pii/S1877050918307129 (visited on 03/08/2021).

[106] Z. Zhong et al. “CDBE: A cooperative way to improve end-to-end congestion control in mobile network”. In: *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). Oct. 2018, pp. 216–223. DOI: 10.1109/WiMOB.2018.8589175.