Nouvelles approches pour les communications multichemins

Matthieu Coudron

UNIVERSITÉ PIERRE ET MARIE CURIE
Laboratoire d'Informatique de Paris 6

Nouvelles Approches pour les Communications Multichemins

Doctoral Dissertation of:
Matthieu Coudron

Advisor:
Dr Stefano Secci

2016

Thèse

Présentée pour obtenir le grade de docteur
de l'Université Pierre et Marie Curie
Spécialité: Informatique

Matthieu COUDRON

Nouvelles Approches pour les Communications Multichemins

Soutenue le 12 décembre 2016 devant le jury composé de:

Rapporteurs	Prof. Catherine ROSENBERG	Université de Waterloo, Canada
	Prof. André-Luc BEYLOT	ENSEEIHT, France
Examinateurs	Dr Christophe LOHR	Télécom Bretagne, France.
	Dr Mathieu BOUET	Thales Communications & Security, France.
	Prof. Guy PUJOLLE	Université Pierre et Marie Curie, France.
Directeur de thèse	Dr Stefano SECCI	Université Pierre et Marie Curie, France.

UNIVERSITÉ PIERRE ET MARIE CURIE
Laboratoire d'Informatique de Paris 6

Novel Approaches for Multipath Communications

Author: Matthieu COUDRON

Defended on December 12, 2016, in front of the committee composed of:

Referees: Prof. Catherine ROSENBERG (University of Waterloo, Canada).
Prof. André-Luc BEYLOT (ENSEEIHT, France)

Examiners: Dr. Christophe LOHR (Télécom Bretagne, France)
Dr Mathieu BOUET (Thales Communications & Security, France).
Prof. Guy PUJOLLE (Université Pierre et Marie Curie, France).

Advisor: Dr Stefano SECCI (Université Pierre et Marie Curie, France).

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé et soutenu tout au long de ces presque quatre années de thèse.

En premier lieu, je remercie mon directeur de thèse M. Stefano Secci, maître de conférence à l'université Pierre et Marie Curie pour m'avoir guidé et conseillé tout au long de ces années dans les arcanes du monde de la recherche.

Je remercie M. Guy Pujolle, professeur à l'université Pierre et Marie Curie et responsable de l'équipe PHARE (ProcHAine génération de REseaux), qui m'a accueilli à bras ouvert au sein de son équipe.

Je remercie M. Achille Pattavina et M. Guido Maier pour leur chaleureux accueil au sein du département d'électronique et de l'information de Polytechnique Milan, en tout particulier à Marco Savi pour son enthousiasme.

Je remercie M. Olivier Bonaventure ainsi que toute son équipe à l'Université Catholique de Louvain, Benjamin Hesmans, Fabien Duschenes, Nicolas Laurent, Raphaël Bauduin, Vanessa Maons... sans qui je n'aurai peut être jamais usurpé l'identité de Saint Nicolas pendant les cours magistraux de master. Je remercie Christophe Paasch, Sebastien Barré et Gregory Detaal pour leur aide précieuse autour du noyau MPTCP. Sans eux, Multipath TCP ne serait pas ce qu'il est aujourd'hui.

Je remercie Tom Henderson, Tommaso Pecarolla, Hajime Takizaki, Peter Barnes pour leur aide sur NS-3.

Je remercie Evan Huus, Peter Wu et Alexis La Goutte pour m'avoir aidé tout au long du processus d'adoption au sein du programme wireshark de mes améliorations liées à Multipath TCP (MPTCP).

Je remercie tous mes collègues de l'équipe PHARE, en particuler Patrick Raad, Dallal Bellabed, Mamadou Tahirou Bah, Dung Phung Chi, Oussama Stiti, Jennifer Torres pour avoir enchanté notre lieu de travail.

Je remercie *Coffee Man* alias Bruno Martin, le thermos sur le coeur,

pour m'avoir aidé à développer puis entretenir une addiction au café.

Je remercie aussi ma famille, mes parents mais aussi mes neveux Antoine et Olivier qui me rappellent régulièrement ce qui compte le plus au monde: les dinosaures bien sûr !

Abstract

The criticity of the Internet keeps increasing with a very high number of services depending on its infrastructure. The Internet is expected to support services with an increasing tangible impact on the physical world such as the Internet of Things (IoT) or autonomous vehicles. It is thus important to address the current infrastructure shortcomings in terms of scalability, confidentiality and reliability.

Multipath communications are one possible solution to address this challenge. The transition towards multipath technologies is not obvious, there are several challenges ahead. Some network devices block unknown protocols, thus preventing the emergence of new technologies, which plays a part in what is often referred to as the *ossification* of the Internet. Moreover, due to technical reasons, there are cases for which multipath technologies perform worse than their single path counterpart. In this thesis, we are interested in addressing some of these cases and limit their impact, so that multipath communications perform better than single path communications as often as possible. More specifically, we propose enhancements to Multipath TCP (MPTCP).

After a detailed survey of multipath communications across all layers, we propose an answer as to the question of how many paths to use and how to ensure proper forwarding. Moreover, motivated by the intuition that packet arrival disorder can be mitigated by the knowledge of one way latencies, we propose a latency estimator with sender-side modifications only. Furthermore, as throughput maximization is in general solved regardless of the interface cost or user preferences, we elaborate a framework capable of presenting more complex strategies if for instance the user wants to enforce throughput even on less efficient paths. Finally, we develop and present a complete simulation model of MPTCP.

Résumé en langue française

La dépendance des différentes infrastructures vis-à-vis du réseau Internet va croissant. D'abord la convergence des médias mais bientôt l'Internet des objets ou les véhicules autonomes peut-être vont contribuer à augmenter la criticité d'Internet. Il est donc important de résoudre les problèmes liés à l'infrastructure actuelle, en terme de passage à l'échelle, de confidentialité ou bien de fiabilité.

Les communications multichemins font partie des possibilités pour attaquer ce défi. Pour autant la transition vers ces technologies n'est pas sans difficulté. En effet certains équipements bloquent les protocoles inconnus, empêchant ainsi l'émergence de nouvelles technologies. C'est un phénomène en partie responsable de l'ossification d'internet. D'autres considérations techniques limitent l'intérêt de recourir à des technologies multichemins dans certains cas, puisque celles-ci peuvent alors présenter des performances moindres que les technologies monochemins.
Dans le cadre de cette thèse, nous proposons des réponses à certains de ces cas afin de maximiser le spectre d'application des technologies multichemins, en particulier du protocole Multipath TCP (MPTCP).

Plus précisément, après une revue détaillée du domaine des communications multichemins, nous proposons une réponse au problème de découverte des chemins. De plus, motivés par l'intuition que les ordonnanceurs peuvent s'appuyer sur les latences unidirectionelles, afin de lutter contre l'arrivé de paquets dans le désordre, nous proposons une technique qui ne modifie que l'envoyeur de données pour estimer cette métrique. En outre, nous proposons un outil qui maximise le débit tout en prenant en compte des politiques utilisateur par exemple pour forcer l'envoi d'une partie du trafic sur un chemin peu performant mais qui va coûter moins cher à l'utilisateur. Finalement, nous développons et évaluons un modèle de MPTCP.

Contents

List of Figures

List of Tables

Glossary

ACK ACKnowledgements.

ALTO Application Layer Traffic Optimization.

AOLIA Adapted Opportunistic Linked Increases Algorithm.

APDU Application Protocol Data Unit.

API Application Programming Interface.

AS Autonomous System.

ATLB Arrival-Time matching Load-Balancing.

BBR Bottleneck Bandwidth and RTT.

BDP Bandwidth Delay-Product.

BERP Bandwidth Estimation Based Resource Pooling.

BFS Backward Fast Subflow.

BGP Border Gateway Protocol.

BMP Buffer Management Policy.

BSD Berkeley Software Distribution.

CMT Concurrent Multipath Transfer.

CMT Concurrent Multipath Transport.

CPU Central Processing Unit.

CWA Congestion Window Adaptation.

DAC Delayed ACK for CMT.

DACK Data Acknowledgment.

DBAS Deplorable Bandwidth Aggregation System.

DC Data Center.

DCCP Datagram Congestion Control Protocol.

DFZ Domain Free Zone.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

DSN Data Sequence Number.

DSS Data Sequence Signal.

DWC Dynamic Window Coupling.

ECMP Equal Cost Multipath.

ECN Explicit Congestion Notification.

ECT Equal Cost Tree.

EDPF Earliest Delivery Path First.

EID Endpoint Identifier.

ELP Explicit Locator Path.

FEC Forward Error Correction.

FFS Forward Fast Subflow.

FOSM Flow-Oriented Scheduling Mode.

FPS Forward Prediction Scheduling.

FQDN Full Qualified Domain Name.

FSM Finite State Machine.

GPL General Public License.

GPS Global Positioning System.

GSB GapAck-Induced Sender Buffer Blocking.

HE Happy Eyeballs.

HIP Host Identity Protocol.

HoL Head-of-Line blocking.

HTTP-RP HTTP Request Pipelining.

HTTP-RRR HTTP Range Retrieval Request.

HTTP/2 HyperText Transfer Protocol 2.

I2RS Interface to the Routing System.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

ILP Integer Linear Programming.

IoT Internet of Things.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

ISP Internet Service Provider.

ITR Ingress Tunnel Router.

JSON JavaScript Object Notation.

KCM Kernel Connection Multiplexor.

LACP Link Aggregation Control Protocol.

LAN Local Area Network.

LEDBAT Low Extra Delay Background Transport.

LIA Linked Increase Algorithm.

LISP Locator/Identifier Separation Protocol.

LTE Long Term Evolution.

MPLOT Multi-Path LOss-Tolerant protocol.

MPTCP Multipath TCP.

MR Map Resolver.

MS Map Server.

MSS Maximum Segment Size.

MTU Maximum Transmission Unit.

NAT Network Address Translation.

NIC Network Interface Controller.

NR-SACK Non-Renegotiable Selective Acknowledgments.

NSC Network Simulation Cradle.

NTP Network Time Protocol.

OF OpenFlow.

OLIA Opportunistic Linked Increase Algorithm.

OOO Out of Order.

OR Opportunistic Retransmission.

OS Operating System.

OWD One Way Delay.

PAT Port Address Translation.

PCA Per-Conversation Allocation.

PCE Path Computation Element.

PCEP PCE communication Protocol.

PET Packet-Pair based EDPF for TCP applications.

PFA Per-Flow Allocation.

POSIX Portable Operating System Interface.

POSM Packet-Oriented Scheduling Mode.

pTCP parallel TCP.

PTP Precision Time Protocol.

QoE Quality of Experience.

QoS Quality of Service.

QUIC Quick UDP Internet Connections.

RAN Radio Access Network.

RLOC Routing LOCator.

RP Resource Pooling.

RR Round Robin.

RRB Reordering-Induced Receiver Buffer Blocking.

RTO Retransmission Timeout.

RTR Reencapsulating Tunnel Router.

RTT Round-Trip Time.

RWB Window-Induced Receiver Buffer Blocking.

SACK Selective Acknowledgment.

SCTP Stream Control Transmission Protocol.

SDN Software Defined Network.

SPB Shortest Path Bridging.

SPTCP Single Path TCP.

SSN Subflow Sequence Number.

STP Spanning Tree Protocol.

TAPS Transport Area Protocol Services.

TCP Transmission Control Protocol.

TDF Time Dilation Factor.

TED Traffic Engineering Database.

TLS Transport Layer Security.

TRILL Transparent Interconnection of a Lot of Links.

TSB Transmission-Induced Sender Buffer Blocking.

TSN Transmission Sequence Number.

TSO TCP Segmentation Offload.

TTL Time To Live.

TTM Test Traffic Measurement.

UCL Université Catholique de Louvain.

UDP User Datagram Protocol.

VM Virtual Machine.

VoIP Voice over IP.

WAN Wide Area Network.

WRR Weighted Round Robin.

XMP eXplicit MultiPath.

List of Software

DIG the Domain Information Groper, a tool to query DNS servers.

LIG the Internet Groper, a tool to query Map Servers.

mptcpanalyzer is a tool to help analyze packet capture files..

mptcpnumerics can compute optimal congestion windows for a specific connection.

mptcptrace a program to help analyze MPTCP packet captures.

wireshark a network protocol analyzer available at `www.wireshark.org`.

DCE Direct Code Execution; allows to execute unmodified C code in NS-3.

GCC the GNU C compiler.

iperf2 a multi-platform C program to do active network measurements available at `http://iperf.sourceforge.net/`.

libnl a Netlink C library.

LibOS a fork of the Linux kernel that can run in DCE.

Linux a C kernel widely deployed through Android devices and servers.

LISPmob a userspace Linux LISP router.

Mathematica A commercial symbolic mathematical computation program.

Mininet an emulator based Linux containers.

Mininet Hi-Fi *High Fidelity* version of mininet enforcing a better isolation of the host resources.

MPTCP Linux kernel a fork of Linux that implements MPTCP.

NS-2 is a network simulator now made obsolete by NS-3..

NS-3 is a popular GPL discrete time event-based network simulator..

NTP client the reference client that illustrates the NTP RFCs.

OpenLISP a FreeBSD LISP router with a kernel dataplane.

OpenVSwitch A multilayer virtual switch.

OWAMP does active measurement to determine one way measurement between hosts (`http://software.internet2.edu/owamp/`).

QEMU a generic and open source machine emulator and virtualizer.

tstat A TCP packet capture analysis program.

VMSimInt Controls I/O of Qemu to run real code in discrete time.

Chapter 1

Introduction

Smart devices or servers equipped with multiple network interfaces are becoming commonplace. Nevertheless, even though multiple interfaces can be used to connect to the Internet, their capabilities have not been fully utilized yet because the default TCP/IP stack supports only a single interface for communication.

The criticity of the Internet keeps increasing with ever more services depending on its infrastructure. The digital convergence today mostly transports information but in the future, the Internet might support services with a more direct impact on the physical world such as the IoT autonomous vehicles. It is thus important to be able to address the current infrastructure shortcomings in terms of scalability, confidentiality, reliability. Multipath communications are one possibility to do so.

1.1 Multipath communications: Incentives

The original Internet was designed as a two-connected network to intrinsically support multipath transmission. Nevertheless, computers with multiple network interfaces were not an immediate design priority at the early stage. Only the routers were equipped with several physical network interfaces. However, the Internet has since then evolved significantly. For example, most servers are equipped with more than one network interface nowadays. The abundance of network resources from the server domain has spurred the adoption of multipath transmission in data center networks. In the consumer electronics domain, the proliferation of mobile devices equipped with cellular (e.g., 3G and Long Term Evolution) and WiFi in-

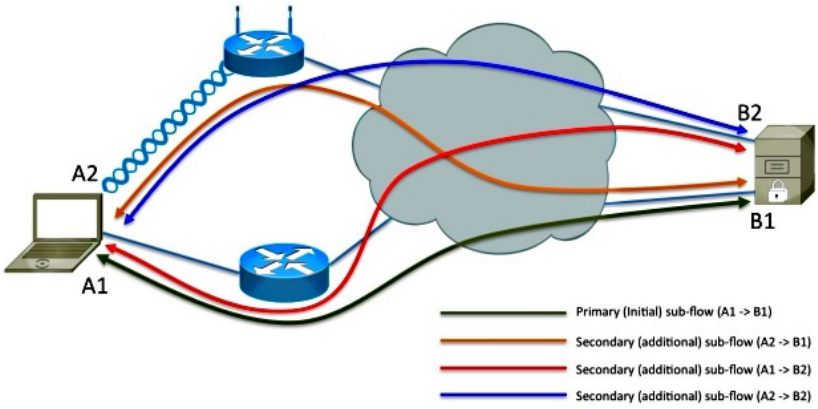

Figure 1.1: An example of server and client multihoming (Source: Cisco documentation).

terfaces, represented by smart phones, brings with it a growing number of multi-homed hosts onto the Internet. Even desktop computers often propose WiFi along with wired connectivity as pictured in Figure 1.1: a laptop utilizes its interfaces A1 and A2 to exchange data over several paths (i.e., "subflows") with the server.

Thus, there exists a mismatch between single-path transport and the multitude of available network paths: according to Agarwal, Chuah, and Katz [1], at least 60% of the stub domains are multihomed to two or more providers. Even when Border Gateway Protocol (BGP) reduces the advertised path diversity as it selects a single best route towards the destination, flows might follow different paths as shown in [2]. Those multi-interface devices require multipath capability to improve end-to-end communication performance and resilience.

Meanwhile, technological advancement has shown benefits from multipath transmission. We list some of the major benefits as follows.

1.1.1 Reliability

Multipath transmission can enhance the reliability of data transfer because additional paths can continue to keep the connection alive in the case of a low path performance or even partial path failure. For instance, MPTCP supports *break before make* scenarios, i.e., it can lose the connectivity on

all its paths, buffer the packets from the application while reestablishing connectivity on another path and finally resume the connection. This is especially useful in the context of mobility where the user can not always predict connectivity loss.

1.1.2 Bandwidth aggregation

One of the most attractive benefits of multipath transmission is to utilize multiple flows, initiated from different TCP/IP layers, for bandwidth aggregation. Expectedly, the bandwidth aggregation can potentially multiply the experienced throughput by the number of available paths. If efficient bandwidth aggregation can be achieved in this manner, a multi-homed device can obtain a much better throughput.

1.1.3 Confidentiality

If a single flow of data is split and forwarded on different paths, it should become harder for a malicious adversary to reconstruct the flow since it would need several probes to capture all traffic or one probe on a shared hop by all paths.

1.1.4 An alternative vision to Resource Pooling

Engineers have designed over the years several mechanisms such as load balancing, statistical multiplexing, failure resilience in order to increase reliability, flexibility and efficiency. These approaches can be considered as Resource Pooling (RP) approaches as they try to make a collection of resources behave like a single pool of resource. These different mechanisms are not perfect: as an example it has been demonstrated that injecting more specific prefixes to support multihoming does not scale[3], also in general they require a few seconds to recover from failures (e.g., hard handover, routing convergence). Instead of handling per path resource independently, a revised multipath vision of the RP principle advocates to harness the responsiveness of multipath-capable end systems, making improved use of multiple path resources by allowing separate paths to act as if they were a single large resource. This approach solves most of the previously mentioned problems. It is a significant step towards a practical multipath-aware end

system, especially when joint congestion control algorithms were introduced to offer both RP and TCP-friendliness features.

As a short summary, the demand from the abundance of network resources from the server and mobile devices domain, and the drive of technological advancements, has made multipath transmission a hot topic in recent years.

1.2 Challenges

While multipath communications exhibit interesting advantages, their deployment is hampered by several factors we describe thereafter. Also, some features presented as incentives can also impede deployment depending on the point of view: the ability to escape a single point of surveillance we previously labeled as an increase in confidentiality can be seen as a danger from a company perspective.

1.2.1 Deployment concerns

The difficulty of deploying new protocols in the current Internet is not a problem specific to multipath communications, yet it has proved to be a real problem and has had much influence on the design of the most recent protocols MPTCP and Quick UDP Internet Connections (QUIC). Internet Protocol version 6 (IPv6) was hard to deploy because of the low incentive compared to the risk of upgrading the hardware and software. A protocol such as SCTP should have been easier to deploy as it was end-to-end, yet number of middleboxes (Network Address Translation (NAT), firewalls) either modify or drop suspicious packets, i.e., a packet that is neither User Datagram Protocol (UDP) or Transmission Control Protocol (TCP).

1.2.2 Heterogeneous Networks

By heterogeneous networks, we refer to paths with different RTTs or loss rates. For some protocols who were designed to deliver data in order, sending successive segments of data on different paths increase the change of Out of Order (OOO) arrivals. For window-based protocols, the system needs to buffer these out of order packets with the possibility, depending on the buffer size, of Head-of-Line blocking (HoL). In this situation, multipath protocols may in fact decrease the throughput.

1.2.3 Pareto-optimality

To be the most convincing, multipath protocols should be Pareto-optimal [4], i.e., upgraded users should always benefit from multipath transport without making any single user worse off. One way to measure the quality of communications is to measure the throughput. As in this thesis, we are mostly interested in transport multipath protocols, Pareto-optimality gains are conditioned by the notion of fairness, which is very relative. The current accepted notion of fairness implies that each TCP flow should get the same share of the bandwidth. As a consequence simply utilizing multiple flows results in an "unfair" share of the bandwidth at the bottleneck; for example, n TCP flows get approximately n times throughput as a competing Single Path TCP (SPTCP) flow does. Therefore in order to comply with the current TCP fairness, multipath transport protocols should consume at bottlenecks the same amount of router buffer than a SPTCP flow. As the detection of shared bottlenecks is a difficult problem, the current MPTCP stance is to be conservative and consider that all paths should be considered to share a bottleneck. This situation obviously makes the aggregation goal harder to reach.

1.2.4 Resource Consumption

The use of multiple paths consume more resources since there are more states to track. This increase in resource usage should be compensated for by the increase of another metric such as the throughput. For instance, it has been shown [5] that enabling multipath for short communications was counterproductive in terms of energy efficiency: in a mobile configuration, a smartphone would power on the energy-hungry cellular interface to enable multipath right before the end of the connection, resulting in an increased energy consumption for no throughput gain. Buffer size is also a resource to monitor carefully as it is directly linked with the number of paths one can use.

1.3 Contributions

Among the challenges we mentioned, some are more relevant than others. The high switching cost might be one of the reasons why the SCTP protocol did not get deployed as much as initially intended: it was a new protocol and

as such was blocked by middleboxes. Also it required to modify applications. Tunneling is a first workaround, at the expense of the Maximum Transmission Unit (MTU), and is used in SCTP/WebRTC combination. The other approach is to look like TCP (for MPTCP) or UDP (for QUIC) and seems to work fine as well.

These two protocols operate at the transport layer. Multipath communications can be envisioned from different point of views and places in the network stack.

While there has been propositions for layer 2 multipath protocols, the transport and application layer seems the most relevant to operate in a multipath context since they have the most information about path heterogeneity, RTTs, information useful to mitigate multipath problems. We intentionally leave out all work that focuses on multipath routing, i.e. the control plane problem of how to compute and select the routes. We refer the readers to articles and recent surveys that cover such work [6]–[10].

In this thesis we focus mostly on the transport layer, more specifically the Multipath TCP protocol, with its associated problems. First and foremost, the transport layer operates at endhost which do not know the path diversity or the path characteristics, which makes it difficult to deduce an efficient number of paths to reach either the confidentiality or throughput goals.

We are interested in identifying these cases and propose solutions to limit their impact, so that multipath communications perform better than single path technologies as often as possible, more specifically Multipath TCP (MPTCP). Our contributions can be resumed as follows:

- We propose a large survey of multipath technologies at the different layers, and present why the transport and application layers appear as the most adequate for such technologies.
- We propose a cross-layer virtual network overlay architecture to ensure maximally disjoint paths and gives MPTCP more visibility on the overlay network topology.
- we propose a integer linear programming framework aimed at determining the optimal receive buffer sizes and optimal congestion windows in order to reach the best throughput under some constraints.
- The previous items show the importance of one way latencies in multipath communications, hence we also propose a technique to compute

the difference in delays between different paths with sender-side modifications only.

- All these contributions showed how difficult experimentation and analysis of multipath protocols could be, hence we decided to implement and share a simulation model along with tools to help with the analysis of MPTCP capture files.

1.4 Structure of the dissertation

The remainder of this dissertation is as follows:

- **Chapter 2** surveys related work around multipath technologies across layers.
- **Chapter 3** describes more in detail the signaling and different aspects of Multipath TCP.
- **Chapter 4** describes an architecture to achieve maximally disjoint paths and illustrates it with a partial implementation based on LISP and MPTCP.
- **Chapter 5** describes one sided sender-only modification to estimate the difference in path OWDs, a metric that can improve the performance of Concurrent Multipath Transport (CMT) protocols.
- **Chapter 6** presents a discrete time event-based simulator that generates two different Integer Linear Programming (ILP) to help with MPTCP window and buffer sizing.
- **Chapter 7** describes our implementation of MPTCP in NS-3 along with some tools to help with MPTCP capture packets analysis.
- **Chapter 8** concludes the dissertation and opens up perspectives for future work.

Chapter 2

Related work

Even though the adoption of concurrent multipath protocols can be considered quite recent, these protocols were designed with the experience learned from past trials and errors. In this chapter, we present the context that led to the current design of protocols, regardless of whether we used them in the rest of the dissertation[1].

2.1 Introduction

The Internet was originally designed as a two-connected network to guarantee that no single failure would cause any non-failed portion of the network to lose connectivity [12]. In essence, any source-destination pair needs to maintain more than one path to assure the reliability and resiliency of the network. Although the rich resources have been existing in the Internet, they have not been fully utilized since the birth of the Internet. The reason lies in the fact that, by default, the conventional TCP/IP only uses a single "best" path according to certain routing metrics; the other available paths remain standby only for backup and recovery purposes.

Nonetheless, this situation has been changing in the past few years, which is indicated by several trends from the standardization organization, academia, and industry. From the standardization perspective, both Institute of Electrical and Electronics Engineers (IEEE) and Internet Engineering

[1]The contents presented in this chapter have been published in [11]. As co-authors, we mostly contributed to the transport and application layers aspects, as well as the aspects related to the coordination between network overlays and transport protocols.

Task Force (IETF) are active on concurrent multipath transmission. There have been several working groups dedicating on the standardization, for example, [13]–[22] [2]. From the academia, hundreds of scientific articles revolving around multipath transmission have been published, covering different network stack layers on various aspects ranging from packet reordering, scheduling to buffer management, fairness, RP. From the industry, several companies have implemented their own link layer aggregation schemes, such as [23]–[25]. Deutsche Telekom offers hybrid access by bundling Digital Subscriber Line (DSL)-line with Long Term Evolution (LTE) in its portfolio [26]. Tessares company [27] tried to develop new innovative network services on top of MPTCP. For example, its first product aims to aggregate the bandwidths of different infrastructure (LTE/DSL). In 2015, OVH company [28] announced a new product called Overthebox. This product combines MPTCP and SOCKS proxies to enable users to bond different DSL lines together. Apple has implemented a variant of MPTCP on part of their Siri servers and allows iOS 7 users to use it in their iPhones [29]. At IETF'93 in 2015, KT Corporation presented Gigapath, a commercial service which can achieve high bandwidth (800 Mbps and more) by combining LTE and WiFi networks on Multipath TCP enabled smart phones [30].

There already exist a few survey articles focusing on different aspects of multipath transmission. For example, [6]–[10] mainly focused on the control plane problem (i.e., multipath routing of how to compute and select paths). [31] covered the control plane problem as well as the data plane problem (i.e., how to split the flow on the chosen paths) in wired networks. [32] assumed multiple paths had been established by routing protocols and focused on load distribution in terms of traffic splitting and path selection. [33] and [34] considered multipath transmission in wireless and wired networks respectively. [35] investigated the common features of various approaches and classified the features into layer-dependent and layer-independent features. In addition, [35] also abstracted common design patterns and proposed a unified networking architecture to enable mobile nodes to make context-aware decisions about how and when to use each or a combination of networks.

We provide a comprehensive survey of multipath transmission, covering various aspects on different layers. Towards that direction, we make several

[2] [22] gives an overview of bandwidth aggregation mechanisms discussed in the context of Banana mailing list `https://www.ietf.org/mailman/listinfo/banana`

Table 2.1: Classification of the research work based on the IP layers.

Stack position	Research work
Link	Multi-Link PPP (MP) [36], [37], strIPe [38], [39], FatVAP [40], IEEE 802.1AX-2008 [13], EtherChannel [23], Aggregated Ethernet [24], Multi-Link Trunking [25], IEEE 802.1AX-2008 [13], OpenFlow [41], [42], IEEE 802.1aq [14], TRILL [18], SPB [43]
Network	Phatak et al. [44], [45], BAG [46], [47], PRISM [48], [49], ETOM [50], MAR [51], INTELiCON [52], MLP [53], SIMA [54], mHIP[55], [56], Sun et al. [57], LISP [17], OSCAR [58], LISP-HA [20]
Transport	BA-SCTP [59], W-SCTP [60], CMT-SCTP [61]–[70], LS-SCTP [71], [72], cmpTCP [73], WiMP-SCTP [74], cmpSCTP [75], mSCTP-CMT [76], FPS-SCTP [77], R-MTP [78], Lee et al. [79], pTCP [80]–[82], R^2CP [83], [84], Cetinkaya et al. [85], mTCP [86], M-TCP [87], M/TCP [88], R-M/TCP [89], cTCP [90], MPLOT [91], [92], JOSCH [93], Super-aggregate [94], BMC [95], MPTCP [96]–[108], Han et al. [109], NC-MPTCP [110], FMTCP[111], [112], QoS-MPTCP [113], CWA-MPTCP [114], Openflow-MPTCP [115], Balia [116], [117], A-MPTCP [118], Yang et al. [119], SC-MPTCP [120], MPTCP-MA [121], EW-MPTCP [122], Yang and Amer [123], DRePaS [124], Coudron et al. [125]
Application	XFTP [126], PSockets [127], GridFTP [128], PA [129], ATLB [130], [131], Tavarua [132], SBAM [133], DMP [134], [135], MultiTCP [136], PATTHEL [137], Kaspar et al. [138], [139], Evensen et al. [140]–[142], Miyazaki et al. [143], DBAS [144], [145], G-DBAS [146], OPERETTA [147], MPTS-AR [19], [148]
Cross-layer	PRISM [48], [49], MPTCP-MA [121], ATLB [130], [131], Tavarua [132], SBAM [133], MultiTCP [136], PATTHEL [137], DBAS [144], [145], G-DBAS [146], OPERETTA [147], A-MPTCP [118], Openflow-MPTCP [115]

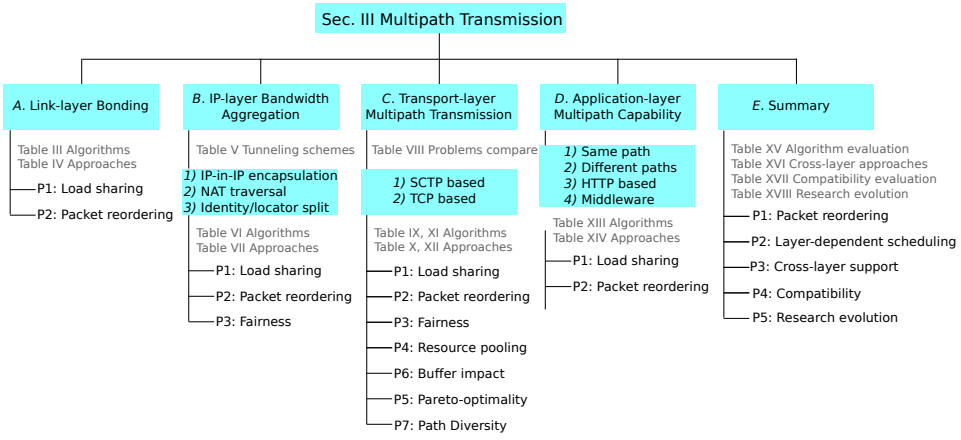

Figure 2.1: Structure of Section 2.1.1 and research problems (P) to address.

key contributions and summarize them as follows: (1) a complete taxonomy regarding multipath transmission is presented, covering various protocol layers including link layer, network layer, transport layer, application layer and cross layer; (2) the state-of-the-art for each layer is surveyed, the problems addressed by layer specific approaches are investigated, and comprehensive comparisons among them are made; (3) the standardization efforts from various parties are summarized, including working groups from IETF and IEEE.

2.1.1 Organization, Structure, and Research Problems

Grouping and discussing multipath transmission approaches according to their stack position are beneficial for researchers and practitioners to understand the benefits and trade-offs from each layer, and make an all-around decision. Therefore, we survey the state-of-the-art multipath transmission from layer-specific perspectives. Table 2.1 shows the classification of the research work according to the stack position.

The structure of the survey is organized as follows. In Section 2.2, various approaches are classified based on their network stack position and cross-layer approaches are discussed separately (see table 2.1). Figure 2.1 illustrates the structure and coverage of Section 2.2. In each discussion of the layer specific approaches, we investigate the problems the approaches on that layer aim to address. Some problems are common to all layers, such as the load sharing and packet reordering problems. Some are addressed only on certain layers. For example, the fairness problem is addressed only on IP and transport layers. Compared with other layers, transport layer approaches have more problems to address including RP, buffer impact, Pareto-optimality and path diversity. The discussion of approaches follows a chronological order except that we group some research work which has similarity or progression. In addition, two tables are used to summarize the key algorithms and approaches respectively. The approach table is connected to the key algorithm table by the means of listing the key algorithms used in each approach as well as the intended network environments of the algorithms. Note that the same algorithms, which are used on different layers, are not repeatedly described in different key algorithm tables. Instead, we only provide explanation in the table when the algorithm is first discussed. Following the discussion of the approaches on specific layers, we

make a summary to present a comprehensive comparison from five perspectives. Finally, we conclude the survey in Section 2.3.

First of all, we investigate multipath transmission in wired and wireless networks In addition, we focus solely on the data plane problem of how to split data on multiple paths and intentionally leave out all work that focuses on multipath routing, i.e. the control plane problem of how to compute and select the routes, as well as security-related issues. Such issues are covered in [6]–[10].

Compared with surveys on multipath routing, surveys on the data plane problems are less popular. There are only a few surveys [10], [31]–[35], [149] that touch on the topic. In a somewhat old but still relevant survey [33], Ramaboli et al. reviewed some bandwidth aggregation approaches in heterogeneous wireless networks which consist of a variety of integrated and jointly managed radio access technologies. They found that packet reordering is the most dominant challenge because it can introduce undesirable delays for real-time applications and unnecessary retransmissions for TCP applications.

We summarize the key algorithms used on all the other layers except the physical layer (we refer the readers to [150], [151] for multipath transmission at the physical layer) and associate the algorithms to the problems they were designed to address. We extract the scheduling algorithms and compare their efficiency in terms of packet reordering and load sharing capabilities without considering their layer dependency. The approaches based on a cross-layer design are summarized and discussed in a separate section. The approaches on different layers are also evaluated from the viewpoint of compatibility capability. We also discuss the evolution of the research questions on multipath transmission and found that only the transport layer approaches following a nice evolution.

2.2 Multipath Transmission

Before we dig into the technical details of multipath transmission, we present the timeline of its milestones in Figure 2.2 in order to give readers a general picture of its development. The first paper on TCP was published in 1974. In the following year, Dr. Maxemchuck proposed Dispersity Routing [152] in his Ph.D. dissertation to concurrently transmit data over multiple paths.

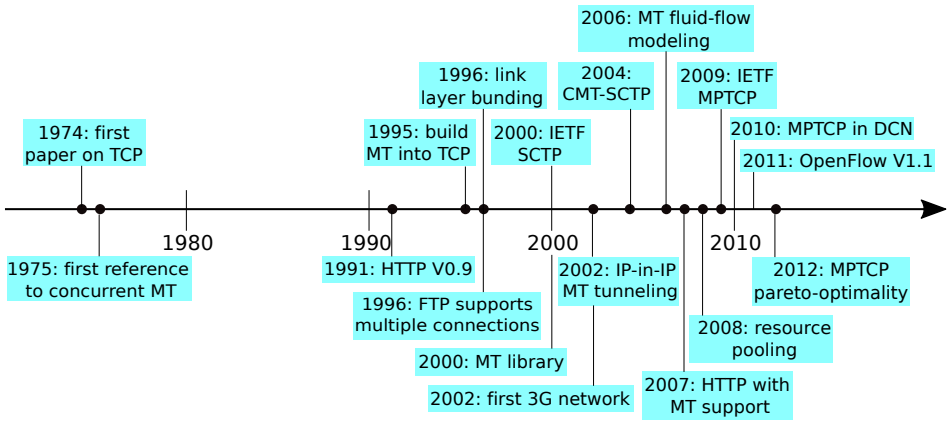

Figure 2.2: Milestones in the evolution of multipath transmission. MT: Multipath Transmission, DCN: Data Center Network, MPTCP: Multipath TCP.

From that point onward, various forms of multipath transmission have been proposed. For example, the idea of building multipath capability into TCP was, to the best of our knowledge, first suggested by Huitema [15] as an Internet draft in IETF in 1995. In 2006, Key et al. [153] used fluid-flow modeling to demonstrate that multipath transport can provide not only robustness but also balanced congestion in a stable manner. In the same year, Shakkottai, Altman, and Kumar [154] used a non-cooperative pricing game to show that multihoming outperforms unihoming in terms of throughput and profit to the Internet Service Providers (ISPs). In 2008, Wischik, Handley, and Braun [155] investigated the RP principle, which makes a collection of resources behave like a single pooled resource. This principle is a significant step towards a practical multipath-aware end system. From 2009, IETF started to define and standardize MPTCP, which employs a coupled congestion control algorithm to achieve RP principle.

In the remainder of this section, the state-of-the-art multipath transmission schemes are classified according to which layer of the protocol stack the proposed approach performs at: link layer, network layer, transport layer and application layer.

Table 2.2: Key algorithms for link layer bonding .

Algorithm	**Problems to address**	**Description**
WRR (Weighted Round Robin)	Load sharing	It is designed to better distribute data onto paths with different capabilities. Each path is assigned a weight which indicates the path's transmission capability in terms of bandwidth, delay and packet loss (or partial of them). Data is distributed over different paths proportionally to their transmission capability.
FLSA (Fair Load-Sharing Algorithm), FQA (Fair-Queuing Algorithm)	Load sharing	A FLSA is obtained by transforming the operations of a Fair-Queuing Algorithm (FQA) in a time reversed manner. FLSA and the corresponding FQA need to run at the sender and the receiver respectively to provide a fair load sharing in the presence of variable sized packets and variable capacity channels.
PCA (Per-Conversation Allocation)	Load sharing, packet reordering	It allocates frames on a per conversation basis. For example, frames belonging to the same conversation is distributed only onto the same path. Multiple different conversations could share the same path.
PFA (Per-Flow Allocation)	Load sharing, packet reordering	It allocates traffic on a flow-by-flow basis. For example, traffic belonging to the same TCP flow is distributed only onto the same path. Multiple different flows could share the same path.
ECT (Equal Cost Tree)	Load sharing	It allows shortest path forwarding in an Ethernet mesh network context utilizing multiple equal cost trees. ECT supports much larger layer-two topologies than per-hop based ECMP.
ECMP (Equal Cost Multipath)	Load sharing	It is a routing strategy where next-hop packet forwarding to a single destination can occur over multiple "best paths". ECMP is a per-hop decision that is limited to a single router.
RR (Round Robin)	Load sharing	This simple scheduling algorithm orders paths and sends each piece of data on the next available/possible path in circular order.

Table 2.3: Summary of Link Layer Bonding Approaches.

Scheme	**Year**	**Algorithm and Protocol**	**Network Environment**	**Re-sequence Header**
MP [36]	1996	WRR	ISDN	Yes
strIPe [38], [39]	1996, 1999	FLSA, FQA	General	Support
LQB [37]	1999	WRR	WWANs	Yes
LACP [13], [156]	2000, 2008	PCA	Ethernet	No
FatVAP [40]	2008	PFA	WAPs	No
SPB IEEE802.1aq [43]	2012	PFA, ECT	Ethernet	No
TRILL [18]	2014	PFA, ECMP	General	No
OpenFlow [41], [42]	2014, 2015	PCA	Ethernet, data center	No

2.2.1 Link Layer Bonding

High end workstations and data centers can easily saturate existing Local Area Networks (LANs). On the link layer, multipath transmission is typically called bonding or link aggregation because multiple physical channels are bundled (or aggregated) into a single logical channel. The primary goal of link layer bundling is to coordinate multiple independent links between a fixed pair of systems, providing a virtual link with a larger bandwidth than what a single link can sustain. Figure 2.3 shows a simplified example of link aggregation between two Ethernet switches (SW_1 and SW_6). These switches can obtain increased throughput by striping data across multiple interfaces.

In the following discussion, we use Table 2.2 and Table 2.3 to summarize the key algorithms and approaches respectively. The approaches in Table 2.3 are sorted in chronological order. The algorithms in Table 2.2 are sorted according to their order mentioned in Table 2.3. Note that the algorithms in Table 2.2 may be not only adopted by approaches on the link layer, but may also be used by those on other layers. In this survey, we will not elaborate the algorithms that have been discussed previously. This same rule is applicable for all other algorithm tables.

Multi-Link PPP (MP) [36], designed for Integrated Services for Digital Network (ISDN), aggregates multiple links using the PPP protocol [157]. In order to detect fragment loss and disorder, MP uses a 4-byte re-sequencing header (RSH) for synchronization and detecting lost fragments at the receiver. Therefore, a reorder buffer is required at the receiver to accommodate the out-of-order fragments caused by link aggregation. MP suggests a Weighted Round Robin (WRR) scheduling scheme so that data can be distributed proportionally to the transmission rates of the links. To achieve this goal, two methods for fragmentation have been proposed. The first one divides packets into segments with sizes proportional to the transmission rates of different paths. The other method divides packets into many small equal sized fragments and distributes the number of the fragments proportionally to the transmission rates of different paths.

Adiseshu et al. [38], [39] added a “strIPe" layer, a virtual IP interface below the IP layer and above the data link layer, to aggregate multiple data links. The stripe layer implements the striping algorithm at the sender and the fair queuing algorithm at the receiver. The authors first showed how

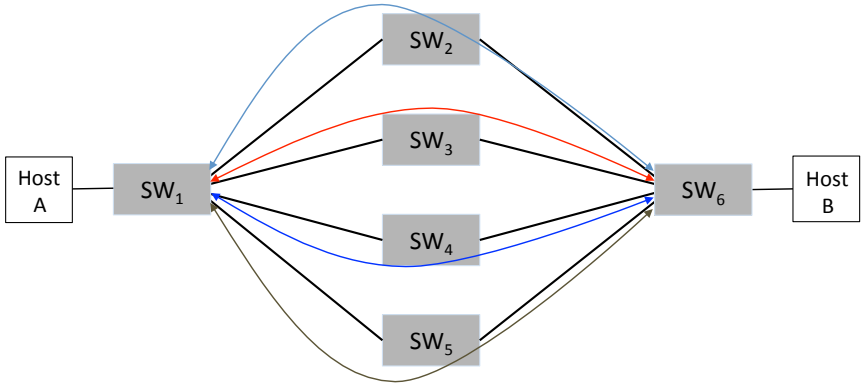

Figure 2.3: Link aggregation between Ethernet switches. SW: Switch.

a fair-queuing algorithm (FQA) can be transformed to a fair load-sharing algorithm (FLSA), and then proposed that the FQA should run in a reversed manner of the load-sharing algorithm in order to solve the load sharing issue with variable packet size. It implies that identical equipment (or of the same vendor) is required on both sides of the aggregation. They also dealt with the FIFO delivery issue for two separate cases. For example, if a RSH can be added to each packet, the issue can be solved by using the additional reordering number; if no header can be added, they proposed a way of synchronization in the event of frame loss to provide quasi-FIFO delivery.

An implementation of MP in Wide Area Networks (WANs) was discussed by Snoeren et al. in [37]. They proposed a Link Quality Balancing (LQB) scheme to bundle multiple channels of the same Wide-area Wireless Access Network (WWAN) technology. In order to adjust traffic striping across bundled links according to their transmission capabilities, LQB adapts the MTU of each link in proportion to its available bandwidth (short-term averages of the observed throughput). A link layer receive buffer is also required to reorder fragments.

FatVAP (an 802.11 driver design) [40] is another work in a wireless environment for link bonding. FatVAP aggregates the bandwidth available at multiple wireless access points (WAPs) that are worth connecting to and balances their loads by scheduling traffic to different APs according to their available bandwidth. In order to continue delivering the sum of the bandwidths available across all APs, FatVAP uses a constant estimation of both end-to-end and wireless bandwidth to react to changes within a few seconds.

Note that FatVAP uses a Per-Flow Allocation (PFA) strategy to distribute traffic over APs. For example, when a new flow arrives, FatVAP determines which AP to assign this flow to and records the mapping in a hash table. Subsequent packets in the flow are simply sent through the AP recorded in the hash table.

Within IEEE specifications, the Link Aggregation Control Protocol (LACP) allows multiple links of Ethernet to be aggregated together to form a Link Aggregation Group (LAG). As such, the Media Access Control (MAC) client can treat the LAG as a single link. LACP allows a network device to negotiate an automatic bundling of links by sending LACP packets to the peer. LACP was initially released as 802.3ad [156] in 2000. Nearly every network vendor quickly adopted this standard over their proprietary standards. In 2008, the protocol was transferred to the 802.1 group with the publication of IEEE 802.1AX-2008 [13]. In LACP, a Frame Collector (FC) at the receiver is responsible for maintaining any frame ordering constraint. In order to avoid frame reordering, the Frame Distributor (FD) at the sender transmits all frames that compose a given conversation[3] only to a single link, which is a Per-Conversation Allocation (PCA) strategy (very similar to PFA strategy). Therefore, no frame reordering scheme or reordering buffer is required at the FC. In addition to the IEEE link aggregation standards, there are a number of proprietary aggregation schemes, including EtherChannel [23] from Cisco, Aggregated Ethernet [24] from Juniper, Multi-Link Trunking [25] from AVAYA. These proprietary aggregation schemes and IEEE 802.3ad standards are very similar and accomplish the same goal.

Since the version 1.1 [158], OpenFlow (OF) has supported multi-link aggregation on layer-2. The specification of OF switch has introduced Link Aggregation (LA) to obtain the ability for one port to point to a group of other ports. Using LACP for exchanging dynamic information between LA-supported devices, the OpenFlow controller has full control over the switches on how frames are distributed and collected on multiple links. Nguyen-Duc et al. [41] investigated the operation of LA in OpenFlow switches and found that LACP on OpenFlow switch provides a slightly lower throughput than the one on a conventional switch. Thus, OpenFlow switches need to be further optimized to achieve equivalent performance. Subedi et al. [42] pre-

[3] A set of frames transmitted from one end station to another, where all of the frames form an ordered sequence, and where the communicating end stations require the ordering to be maintained among the set of frames exchanged.

sented an adaptive multipath forwarding architecture in a layer-2 OpenFlow data center network. In the architecture, all-to-all forwarding paths are set up proactively among the edge nodes. Aggregated bandwidth is achieved by using all the available paths simultaneously. To avoid the out-of-order delivery issue due to using all available paths, a PCA style scheduling algorithm is used in [41], [42]. Specifically, the algorithm excludes the paths whose path length exceeds the shortest path length significantly.

In the last few years, there are notable new protocols designed to support multipath forwarding at link-layer in IEEE and IETF standards, e.g., Shortest Path Bridging (SPB) [43] (specified in the IEEE 802.1aq standard [14]) and IETF Transparent Interconnection of a Lot of Links (TRILL) [18]. SPB and TRILL are potential successors of the Spanning Tree Protocol (STP). SPB now supports multipath forwarding by using Equal Cost Multipath (ECMP) and Equal Cost Tree (ECT) routing strategies. TRILL currently only supports multipath forwarding by using ECMP routing strategy. In both ECMP and ECT strategies, if multiple equal cost paths are present towards a destination, network traffic is distributed over those multiple paths. In order to avoid reordering and path MTU discovery problems, similar to FatVAP, both TRILL and SPB use per-flow multipath forwarding. Specifically, frames belonging to the same data flow take the same path and frames belonging to other flows can take the other paths.

From Table 2.2, we find that the main problems the algorithms trying to address are load sharing and packet reordering. Among all the algorithms, the simplest one is Round Robin (RR) where the sender allocates fragments among all the available links in equal portions and in an ordered fashion. In the long run, RR scheduling provides a fair share of fragments as long as these fragments are of the same size. Nevertheless, the basic RR scheduling strategy is rarely used in practice because it provides no load sharing with either variable sized fragments or different link capacities. In order to solve these issues, the Weighted RR variations are the more widely used scheduling strategies.

The main advantage of link-layer bonding is that the signaling rate of the channel is relatively stable and can be utilized to mitigate reordering. However, link-layer approaches only work on a point-to-point link and even require dedicated Ethernet cards installed on both sides. Thus, they are not applicable in general scenarios of end-to-end communications where the

different domains involved are controlled by different providers.

2.2.2 IP Layer Bandwidth Aggregation

Table 2.4: Schemes used on IP level for bandwidth aggregation.

Scheme	Description	Update
IP-in-IP encapsulation	*No Proxy*: The client and server open a TCP connection with an agreed IP for each other. When a packet is sent through interfaces with IPs other than the agreed one, the packet is encapsulated in another packet with the agreed IP.	Endpoints
	One proxy at the client side: A proxy is required. IP-in-IP encapsulation is running between the proxy and the client to hide the usage of multiple IPs from TCP. The server which is unaware of the client's multiple IPs communicates with the proxy using normal TCP.	Client, network
	Two proxies at the both sides: Two proxies on the client and server sides are required. IP-in-IP encapsulation is running between the proxy client and server. Each endpoint communicates with the proxy (client or server) with normal TCP. The usage of their multiple connections is hidden from both endpoints.	Network
NAT (Network Address Translation)	*No Proxy*: The client and server agree with one IP for each other. The source and destination IPs at the client are replaced with the agreed ones. Upon receiving a packet, the server reverses its source and destination IPs using the agreed ones before forwarding it to TCP.	Endpoints
	One proxy at the client side: One proxy is required. NATing is running between the proxy and the client to hide the usage of multiple IPs from TCP. The server which is unaware of the client's multiple IPs communicates with the NAT box using normal TCP.	Client, network
	Two proxies at the both sides: A proxy client and sever are required. NATing is running between the proxy client and server. Each endpoint communicates with the proxy (client or server) with normal TCP. The usage of multiple connections between proxies is hidden from both endpoints.	Network
Identity/Locator Split	*Host-level:* The identity of a host is separated from its location (i.e., IP address). Each host uses its globally valid identity to shield the presence of its multiple IPs from transport and application layers.	Endpoints
	Network-level: The IP space is separated into two spaces, one for identity of a host and the other for locator of a border router. A mapping system is required to provide mapping between the identity and locator. Multipath transmission could be provided between source and destination border routers for the purpose of traffic engineering.	Network

The IP layer, originally proposed to handle global addressing and routing, is a natural candidate to host the multipath capability to enhance end-to-end communication. A network approach has the advantage of being transparent to transport protocols and applications, making wide spread deployment much easier. In theory, each packet of a TCP flow can be sent over a different path, and the IP protocol ensures that all packets reach their destination. For example, Sun et al. [57] explored the use of multipath routing to reduce the file transmission delay in a wireless network. Specifically, they proposed taking advantage of packet level erasure code (e.g.,

Table 2.5: Key algorithms for IP layer bandwidth aggregation (sorted according to their order mentioned in Table 2.6).

Algorithm	Problems to address	Description
PET (Packet-Pair based EDPF for TCP applications)	Load sharing, packet reordering	It sends TCP packet-pairs on each path periodically to compute inter-arrival time between the hosts, and schedules packets on the path that delivers it the earliest. PET is a variant of EDPF.
BMP (Buffer Management Policy)	Spurious retransmission, packet reordering	It is designed to hide any residual reordering from TCP at the data receiver side so that unnecessary retransmissions are avoided. For instance, the receiver buffers out-of-order data packets at the network layer before passing them to TCP in order.
EDPF (Earliest Delivery Path First)	Load sharing, packet reordering	It estimates the delivery time of the packets on each path, and schedules each packet on the path that delivers it the earliest. This approach is used to minimize reordering and thereby the delay and jitter experienced by the application.
RPC (Reverse Path Controller)	Spurious retransmission, packet reordering	It is designed to handle spurious duplicated ACKs in the data sender side so that unnecessary retransmissions are avoided. For example, RPC exploits TCP's control information carried by ACKs, determine the meaning of duplicated ACKs, corrects them if necessary.
SACK (Selective Acknowledgment)	TCP performance	It is sent from the receiver to the sender informing the sender of the out of order data that has been received. The sender can then retransmit only the missing data segments.
DATA/ACK SEP	TCP performance	It separates the forward (DATA) and the backward (ACK) traffic on different paths.
PBCS (Piggy-Backing for Control Signaling)	Path status	It adds piggy-backing extra information on packets before injecting them into the networking stack for transmission. The information is stripped out at the recipient.
TFCC (TCP-Friendly Congestion Control)	Fairness	It restricts the subflows of one TCP connection to use more bandwidth than normal TCP does at a shared bottleneck.
PRM (Packet Reordering Module)	Spurious retransmission, packet reordering	It runs at both sides of a communication to handle packet reordering issue. Specifically, it delays the data packets at the receiver and their ACKs at the sender before forwarding them to the upper layer. To avoid over-protection, it only delays forwarding them before the timeout.

digital fountain code) to transmit data file with redundancy over a set of paths. They obtained the intuitive understanding of the trade-off between the code rate and delay reduction. Their research was made for a special network environment where a source and destination pair has a rich set of identical and disjoint paths, hence no packet reordering issue introduced. Nevertheless, in most practical network environments, when packets inside one connection taking more than one path, they can experience different propagation delay and arrive out of order. The TCP receiver sends duplicate acknowledgments (ACKs) to the sender, which causes the TCP sender mistakenly interprets packet reordering as packet loss. The results found in

Table 2.6: Summary of IP Level Bandwidth Aggregation Approaches.

Scheme	Year	Tunneling	Proxies or Updated Routers	Algorithm and Protocol	Fairness	Network Environment	Sequence Space
Phatak et al. [44], [45]	2002, 2003	IP-in-IP encapsulation	0	WRR	No	Mobile	Single
MAR [51]	2004	NAT	1 or 2	WRR, PFA	No	Mobile	Single
BAG [47]	2005	IP-in-IP encapsulation	1	PET, BMP	No	Wireless access	Single
PRISM [48], [49]	2005, 2007	IP-in-IP encapsulation	1	EDPF, RPC, SACK	No	Mobile collaborative	Single
BAG [46]	2006	IP-in-IP encapsulation	1	EDPF	No	Wireless access	Single
SIMA [54]	2006	Identity/locator split	0	PFA	No	HIP-enabled	Double
INTELiCON [52]	2008	NAT	0	DATA/ACK SEP, PBCS	No	Wireless access	Single
mHIP [55]	2009	Identity/locator split	0	EDPF	No	HIP-enabled	Double
MLP [53]	2009	NAT	1	WRR	No	Wireless access	Single
mHIP [56]	2011	Identity/locator split	0	TFCC	Yes	HIP-enabled	Double
ETOM [50]	2012	IP-in-IP encapsulation	2	EDPF, BMP	No	Wireless	Double
LISP [17]	2013	Identity/locator split	2	WRR	No	General	Single
OSCAR [58]	2014	NAT	0	PRM, PFA, WRR	No	Mobile collaborative	Single
LISP-HA [20]	2015	Identity/locator split	2	PFA	No	Hybrid access	Single

[48], [159], [160] show that TCP suffers significant performance degradation due to frequent packet reordering. Thus, the use of multiple paths with varying characteristics deteriorates the problem.

In the following discussion, the state-of-the-art is divided into three categories: IP-in-IP encapsulation, NAT traversal, and Identity/locator split. We summarize their features in Table 2.4 and discuss each category according to the order they show in the table. Table 2.5 and Table 2.6 are used to summarize the key algorithms and approaches respectively. The “Proxies or Updated Routers" in Table 2.6 indicates the required number of proxies or updated routers.

IP-in-IP Encapsulation

A widely used IP layer approach for aggregating bandwidth of multiple IP paths is to use tunneling mechanisms which transparently redirect packets between two hosts on routing level. For example, Phatak et al. [44], [45] proposed using IP-in-IP encapsulation [161] to split a data flow across multiple network interfaces. As shown in Figure 2.4, at the source (A), the

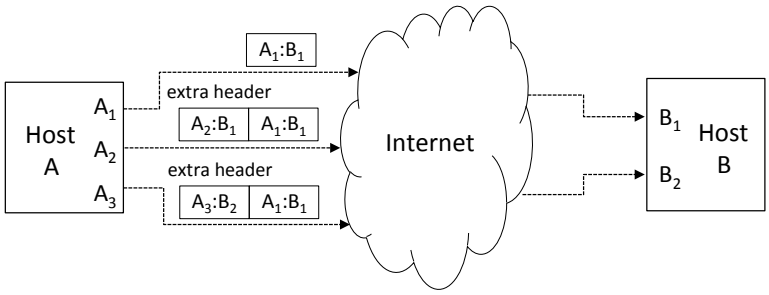

Figure 2.4: IP-in-IP tunneling between two multi-homed hosts.

transport layer assembles all packets as if they were going through A_1 and addressed to B_1. The packets going out on interface A_2 get encapsulated in new IP packets each with an extra header having destination B_1 and source A_2. Likewise, each packet going out on interface A_3 can be encapsulated in a new IP packet having destination B_2 and source A_3. The destination (B) can then recognize IP-in-IP packets and strip the outer header. This leaves the original packets with source A_1 and destination B_1 to be delivered up the network stack to TCP in a transparent manner. The same encapsulation scheme is used for tunneling in the mobile IP standard [162]. In order to avoid fast retransmission, Phatak et al. used a WRR style scheduler, which distributes packets proportionally to the effective rates of the paths.

Chebrolu et al. [46] presented a network layer architecture to aggregate bandwidth on multiple paths for real-time applications. They made the assumption that an infrastructure proxy (like the Home-Agent in Mobile IP [163]) is aware of the multiple interfaces of the client, and tunnels the captured packets to the client using IP-in-IP encapsulation. The advantage of a proxy solution is that it is fully controllable and allows servers to remain unchanged and hide using multiple IPs from TCP. Chebrolu et al. proposed a scheduling algorithm, Earliest Delivery Path First (EDPF), to ensure that packets meet their playback deadlines by scheduling packets based on the estimated delivery time of the packets. To improve the overall performance of IP-in-IP tunneling based bandwidth aggregation by the means of minimizing packet reordering, Chebrolu et al. in [47] proposed a two-pronged approach. Firstly, a scheduling policy Packet-Pair based EDPF for TCP applications (PET) was used to partition traffic onto different paths. The design of PET has the same concept of EDPF but with idealized delay and

bandwidth values replacing the estimates. Secondly, working together with the scheduling policy, a receiver-side Buffer Management Policy (BMP) was used to delay forwarding the out-of-order packets to TCP and to detect losses, so that a variety of adverse effects can be hidden.

Kim et al. [48], [49] introduced PRISM, another proxy based approach that enables TCP to efficiently utilize the WWAN connections from community members. The proxy can be a trusted party or a community member. PRISM uses a cross-layer approach that involves support from both transport and network layer. We classify PRISM as network layer approach because the PRISM proxy, which is the main entity for multipath support, is located on the network layer. PRISM uses a packet-scheduling algorithm, i.e., Adaptive Scheduler (ADAS), to maintain up-to-date path state. Using the up-to-state information, ADAS sends packets according to their expected arrival time (a variant of EDPF algorithm) to reduce packet reordering. ADAS also uses the path state to adjust path weight by using the Additive Increase and Multiplicative Decrease (AIMD) strategy from TCP, so that ADAS can dynamically react to congestion from partial paths and control the amount of traffic to be allocated on those paths. Moreover, PRISM masks the effects of out-of-order delivery by identifying spurious duplicate ACKs and re-sequencing them so that a TCP sender receives correctly sequenced ACKs.

Lan et al. [50] designed a different proxy based multipath network protocol called Enhancements for TCP On a Multi-homed mobile router (ETOM) that runs transparently to both clients and servers. ETOM involves two proxy components instead of one kind: MR (Mobile Router) and HA (Home Agent). The client and the MR (as well as the server and the HA) use normal single-path connection, whereas all packets traveling between MR and HA are IP-in-IP encapsulated. ETOM uses a reordering buffer to eliminate packet reordering. For example, out-of-order packets are buffered at the HA until the missing packets are received, and packets are then sent out in order to the destination. ETOM also uses a variant of EDPF algorithm to further reduce packet reordering. Note that unlike other IP layer approaches, ETOM employs a subflow sequence number in the inner IP header to detect packet loss between MR and HA.

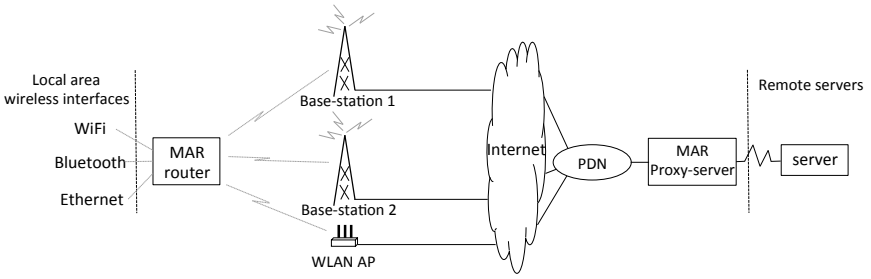

Figure 2.5: MAR [51] system architecture where the MAR router is placed in public mobile vehicles and data traffic is sent from remote servers to local devices. PDN: Public Data Network, AP: Access Point.

Network Address Translation Traversal

Unlike the previous approach that relies on IP encapsulation, there exist several approaches taking advantage of NAT instead of tunneling.

Rodriguez et al. [51] introduced MAR system (see Figure 2.5), a commuter mobile access NAT router that provides a set of local interfaces and a number of wide-area wireless interfaces. The former provides access to local mobile devices and the latter accommodates a variety of wide-area wireless technologies. The MAR router acts as a NAT box that is located in the middle and translates IP addresses and ports of packets for two directions. A MAR router can work alone or cooperate with a MAR proxy-server. With such a proxy-server, the Packet-Oriented Scheduling Mode (POSM) is used where the packets of the same TCP flow can be delivered over multiple paths and a MAR router can implement intelligent optimization including avoiding TCP 3-way handshake, slow-start, spurious timeouts and so on. When the proxy-server is absent, the Flow-Oriented Scheduling Mode (FOSM) is used where a per-flow allocation strategy schedules all packets belonging to the same TCP flow onto the same path. MAR provides an API that can accommodate any custom purpose-built scheduling protocol. But the scheduling protocol itself is not part of the MAR architecture. MAR is also designed to determine the weight that should be assigned to each interface to properly perform load balancing (e.g. dynamically shifts load from poor quality to better quality channels). Note that the MAR router was supposed to be placed in moving vehicles, where users can use their devices for web-browsing and audio/video streaming. Therefore, the traffic load-balanced is

only in one direction (i.e., from remote servers to local devices).

Manousakis et al. [52] proposed INTELiCON to allow devices to exploit wireless access diversity. At the sending side, a Packet Processing module manipulates the content of packets, e.g., modifying IP headers and Piggy-Backing for Control Signaling (PBCS) (e.g., timestamp and customized sequence numbers). At the receiving side, the piggy-backed information can be utilized to smooth out the arrival sequence of incoming packets. The extra information is stripped out at the Packet Processing module before the packets are forwarded to the upper layer. Moreover, INTELiCON uses a DATA/ACK separation (SEP) scheme to reduce contention on shared media. For example, it transmits the DATA and ACK packets on different paths.

Evensen et al. [53] proposed a Multilink Proxy (MLP) that makes use of a NAT proxy to rewrite the default destination IP address and port to the address of the other additional interfaces. The client does the inverse address translation of the packets arriving at non-default interfaces and forwards the packets internally. In order to mitigate packet reordering, Evensen et al. uses a WRR based scheduler in the NAT to distribute packets according to estimated throughput ratio of available paths.

Habak et al. [58] proposed OSCAR architecture that works in a distributed environment. An OSCAR-enabled node can share and use the bandwidth available from its OSCAR-enabled neighbors to connect to both legacy and OSCAR-enabled servers. OSCAR has a NAT module at both sides of a connection. At the sender, the NAT module replaces the source and destination IP addresses with the used IPs for transmission. Upon receiving a packet, the NAT module at the receiver reverses the source and destination IPs by replacing them with the negotiated ones before delivering the packet to TCP. When a connection goes through a shared neighbor to a legacy server, the neighbor also needs to have a NAT module that conducts the address translation operation. OSCAR uses a Packet Reordering Module (PRM) to handle packet reordering issues. Specifically, it delays the packets and their ACKs on both sides respectively before forwarding them to the upper layer. OSCAR has two scheduling modes: POSM and FOSM. FOSM is used if the server is a legacy server, where PFA is used. POSM is used if the server is OSCAR-enabled such that a WRR style scheduler is used.

Some of the IP-in-IP encapsulation and NAT based approaches, e.g., [46]–[51], [53], assume the presence of a proxy infrastructure in the network. Nevertheless, such approaches work only for plain-text TCP communication and fail in the presence of IPsec encryption or authentication mechanisms. When TCP packets are protected with IPsec, the proxy is not able to observe or modify the packet headers. Next we discuss certain bandwidth aggregation approaches that use IPSec encapsulation for tunneling.

Identity/locator Split

Host Identity Protocol (HIP) [164][4] and Site Multihoming by IPv6 Intermediation (SHIM6) [165] have been proposed and implemented to provide multihoming support for failover with the possibility of flow-based load balancing. SHIM6 is architecturally related to HIP in that they both introduce an additional addressing layer to allow changing IP addresses on network interfaces, while keeping constant transport-layer identifiers. These two protocols enable IP packet flows to dynamically change paths in the presence of link failure. Thus, they naturally shield the presence of multiple paths from transport and application layers, presenting only the global identity of the peer host. Nevertheless, HIP and SHIM6 do not support simultaneous multipath transmission without additional extensions.

SIMA [54] is an extension of HIP to use multihoming for assigning separate TCP connections independently to different paths. Like FatVAP [40], TRILL [18], SPB [43], MAR [51] and OSCAR [58], SIMA also uses PFA multipath forwarding strategy where flow bonding rules are created to define the usage of the local interfaces. SIMA does not define any additional sending or receiving policies to mitigate reordering issue, instead it uses the IPsec Encapsulating Security Payload (ESP) packet processing unit built in HIP to handle each data packet, as specified in [166]. Gurtov et al. [55] designed and implemented Multipath HIP (mHIP), a multipath scheduler based on HIP, to distribute traffic over multiple available paths. Utilizing a EDPF scheduling algorithm, they striped packets within a TCP connection to multiple paths to mitigate packet reordering. Nevertheless, they found that EDPF algorithm is only effective against packet reordering with stable paths in terms of bandwidth and delay. In order to react to dynamic

[4]HIP may not be considered as a strict IP layer approach; however, its functions related to multipath transmission are best suited to this layer.

path characteristics, a Marking Technique is used as a part of the multipath congestion avoidance scheme, so that changes of path characteristics can be detected in one RTT.

Polishchuk and Gurtov [56] proposed a TCP-friendly congestion control algorithm for mHIP to prevent stealing bandwidth from legacy TCP flows at the shared bottleneck. Specifically, they proposed a two-level congestion control scheme (removing the Marking Technique): per-path congestion control, and global congestion control on top of it. The global congestion controller coordinates the individual per-path controllers and balances traffic load among the paths based on their available capacity. The per-path controllers are connected so that the aggregated congestion window is the sum of per-flow congestion windows. The goal of this twofold congestion control scheme is to automatically redirect traffic from congested paths to the ones that have available capacity. The concept of joint congestion control algorithm adopted in [56] is also used by certain transport layer approaches (which will be discussed in the next section). Thus, the concern is that the reordering and congestion avoidance algorithms used on the IP layer (or between IP and TCP, like HIP) may need to repeatedly design additional mechanisms that are already existing on the transport layer.

When ESP is used with HIP, a 64-bit sequence number must be used. Therefore, HIP based bandwidth aggregation approaches such as SIMA [54] and mHIP [55], [56] all have a double sequence space design. However, instead of being used for packet reordering, the additional sequence number in HIP is used for the purpose of anti-replay.

Unlike HIP and shim6 which focus on host-level identity and locator separation, LISP [17] is an identity and locator separation protocol working on the network-level to improve the scalability of the routing system. LISP creates two numbering spaces and uses two IP addresses: Endpoint Identifiers (EIDs) (assigned to end-hosts) and Routing LOCators (RLOCs) (typically assigned to border routers). To achieve the separation of identification and localization, LISP follows a map-and-encapsulate scheme. Specifically, upon reception of a packet from the local network to an outer EID, the border router is responsible for looking up and retrieving the mapping (from a mapping system) between EID and RLOC and this process is invisible to the endpoints. Then the router encapsulates the packet with a LISP header and an outer IP header with the destination RLOC as the destination IP

address. When the packet reaches the border router assigned with the destination RLOC, the router decapsulates the outer headers and forwards the inner packet to the destination EID. LISP has two metrics to support multipath transmission: RLOC priority and RLOC weight. If equal priority is sent on the , the RLOC weight could be used for the load-balancing ratio. Under such a setting, an IP-level aggregate flow (e.g., the same destination prefix) would use different paths. For a more detailed presentation of LISP, refer to page 103.

Locator/Identifier Separation Protocol - Hybrid Access (LISP-HA) [20] is a mechanism to provide simultaneous hybrid access (e.g., DSL-line and LTE) based on LISP technology in both upstream and downstream direction. LISP by itself has basic capabilities to support hybrid access with static load balancing. However, static load balancing may lead to statistical variations [167] so that some paths are already overloaded while others are underutilized. Instead, LISP-HA can perform dynamic per-flow load-balancing, which increases the efficiency of hybrid access. The basic idea is to obtain feedback about path-specific packet loss and delay, and leverage this information for improved load balancing. In addition, LISP-HA also supports dynamic per-packet load-balancing. Currently, the challenge is the packet reordering problem in the case that paths have different delay.

As summarized in Table 2.5, packet reordering is one of the main challenges for all IP layer approaches. These approaches use various scheduling algorithms to minimize the reordering effect. In table 2.6, we have several observations. First, most of the approaches were proposed either for mobile networks or wireless networks. Second, there are two primary scheduling schemes: EDPF and WRR. Third, some buffer management strategies are used to compensate for the inefficiency of the scheduling algorithm in the scenario of dynamically changing networks.

2.2.3 Transport Layer Multipath Transmission

Compared with IP layer based approaches, transport layer approaches have certain inherent benefits because congestion control can be used as a mechanism for resource allocation in a network. At this layer, end-systems can easily obtain information about each path: capability, latency, loss rate and congestion state. This information can then be used to react to congestion in the network by moving traffic away from the congested paths. Cur-

rent connection-oriented transport protocols, e.g., TCP, SCTP [168], and Datagram Congestion Control Protocol (DCCP), transmit data only over a single path between a source and a destination at any given time. Numerous attempts have been made to tune these existing transport protocols for multipath capability. Currently, Concurrent Multipath Transfer (CMT) for TCP and SCTP are in the process of IETF standardization.

Like bandwidth aggregation on network layer, concurrent multipath transmission at the transport layer introduces an increase in the occurrence of packet reordering due to different path characteristics, including run-time throughput, RTT, loss, and error. Specifically, if a connection is striped over multiple network paths, the overall throughput may potentially be even worse than the throughput available on any one of the paths [106], [124]. There are two main causes of it. The first comes from the impact of heterogeneous RTT. For example, TCP expects a first-in-first-out delivery of packets through the network. Packet reordering at the receiver results in the reception of duplicate ACKs at the sender. The sender will fast retransmit the "missing" packet that may still be on its way over a high RTT path. Due to the misunderstanding of packet reordering, the overall throughput may degrade significantly. The second cause is the receive buffer blocking due to path heterogeneity or path failing. We provide more detail about the receive buffer blocking problem in later discussion on CMT-SCTP.

In this section, we classify the state-of-the-art according to their base protocols, in the order of QUIC, SCTP and TCP. In each discussion of SCTP and TCP based approaches, we further divide them into two categories: with and without considering fairness. In Table 2.7, we make a comparison of the general problems addressed by approaches in each category. In addition to the fairness issue, we also analyze the buffer impact, Pareto-efficiency, and path diversity of the TCP based approach MPTCP. In the end, we give a comparison between two representative approaches based on SCTP and TCP (i.e., CMT-SCTP and MPTCP).

Tables 2.8 and 2.9 (sorted according to their mentioned order respectively in Table 2.9 and Table 2.11) are used to summarize the key algorithms and approaches of SCTP based multipath transmission respectively. Likewise, Table 2.10 and Table 2.11 are used to summarize the key algorithms and approaches of TCP based multipath transmission respectively.

Table 2.7: Comparison of problems addressed by SCTP and TCP based multipath transmission approaches.

	SCTP based approaches	TCP based approaches
No fairness	• Maximize throughput • Spurious retransmission • Head-of-line Blocking (HLB)	• Maximize throughput • Spurious retransmission • Head-of-line Blocking (HLB)
Fairness	• SCTP-friendly at the same bottleneck • Resource pooling (RP)	• Avoid establishing multiple subflows at the same bottleneck • Maximize throughput at different bottlenecks • TCP-friendly at a shared bottleneck • Resource pooling (RP) • Incast collapse • Quality-of-service for multimedia applications • Trade-off between responsiveness and friendliness

Quick UDP Internet Connections

QUIC is a transport protocol started in 2013 by Google but recently taken over by the IETF working group (started in October 2016 [169]). Because TCP is implemented in operating system kernels, and middlebox firmware, making significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP, it suffers from no such limitations and was devised by Google as a way to quickly experiment different technologies, and potentially encourage their adoption in TCP. QUIC is embedded in the Google Chrome browser and enabled when visiting Google servers with QUIC support.

The first time a QUIC client connects to a server, the client must perform a 1-round-trip handshake in order to acquire the necessary information to complete the handshake. The client sends an empty client hello message, the server sends a rejection message with the information the client needs to make forward progress, including the source address token and the server's certificates. The next time the client sends a hello, it can use the cached credentials from the previous connection to immediately send encrypted

requests to the server.

The protocol supports a set multiplexed connections over UDP, and was designed to provide security protection equivalent to Transport Layer Security (TLS), along with reduced connection and transport latency, i.e., when the server cryptographic details are in the cache, QUIC is capable of sending cyphered data in the first packet. TCP is capable of sending payload in the initial SYN packet with the experimental option Fast Open [170] but it can be replayed and makes SYN flood attacks more efficient.

The multiplexing aspect of QUIC plays an important role in minimizing the latency of HyperText Transfer Protocol 2 (HTTP/2) applications as a loss only impacts one stream, hence one element of the loaded page.

In order to support mobility scenarios, QUIC connections are identified by a 64 bit connection ID, randomly generated by the client. This means that contrary to TCP where a client changing IP addresses (for example, by moving out of Wi-Fi range and switching over to cellular) or ports (if a NAT box loses and rebinds the port association), would lose any active connections, a QUIC client can continue to use the old connection ID from the new IP address without interrupting any in-flight requests. Using the ID rather than the TCP tuple also allows in practice to use different paths as MPTCP would.

QUIC has adopted the TCP Cubic [171] congestion control along with a set of techniques under review to avoid congestion. By comparison, TCP employs a single technique, congestion windows, which are unforgiving to multiplexed connections. Among the techniques being tested are packet pacing and proactive speculative retransmission. The packet pacing is also used by Google in their TCP Bottleneck Bandwidth and RTT (BBR) congestion control which was made public in December 2016 [172]. While not explicit, QUIC may have played a role in testing the congestion control beforehand. A distinctive feature from TCP is that each packet, even retransmitted, carries a new sequence number which allows a QUIC sender to distinguish acknowledgments between the original and the retransmitted packet. Acknowledgments explicitly carry the receiver processing time to allow for accurate RTT computation at the sender. ACK frames also support up to 256 ranges of missing data, to compare with the 4 TCP Selective Acknowledgment (SACK) ranges.

As for proactive speculative retransmission, it amounts to sending du-

plicate copies of the most important packets, such as the initial encryption negotiation packets or lost packets. Losing either of these packet types triggers a snowball effect, so selectively duplicating them can serve as insurance.

Among the innovative features (i.e., deployed at such a scale) is the use of Forward Error Correction (FEC) to recover from lost packets without waiting for a retransmission: QUIC can complement a group of packets with an FEC packet. If one of the packets in the group is lost, the contents of that packet can be recovered from the FEC packet and the remaining packets in the group. The sender may decide whether to send FEC packets to optimize specific scenarios

SCTP based on Multipath Transmission

SCTP standardization started in 2000 [173] as a general-purpose, connection-oriented unicast transport protocol. An *association* denotes a SCTP connection denotes. The user data is segmented into units of so called DATA chunks[5], which are identified by unique Transmission Sequence Number (TSN) [6]. The SACK [174] mechanism is used as default to acknowledge received data chunks and report gaps (i.e., missing data chunks indicated by their TSNs) to the sender. and has similar characteristics and applications as TCP, but includes some important improvements:

1. No head-of-line blocking: TCP imposes a strict data ordering. However, if a user data message is lost during transit, all subsequent messages are delayed until the lost message is received. Depending on the Retransmission Timeout (RTO), this situation can lead to HoL. Some applications do not require strict ordering of messages (bulk transfer), in such cases, this HoL artificially hurts the connection.

2. Embedded message framing: TCP is stream-oriented, meaning that a tcp stack transmits received bytes in-order to the application. While this allows to support all applications, this can unnecessarily degrade the connection to seen in the previous items. It also means that each application has to implement a framing protocol. In comparison, SCTP is message oriented, it preserves message boundaries using its own message framing protocol, application messages (Application

[5]Corresponding to segments in TCP.

[6]TSN serves the same function in SCTP as the sequence number does in TCP.

Table 2.8: Key algorithms for SCTP based CMT.

Algorithm	**Problems to address**	**Description**
WRR-PULL	Scheduling	It is a congestion window based data allocation without estimation of each path's available bandwidth. Data packets are stored in a shared sending buffer and are pulled by sub-flows when the sub-flows have congestion windows space to transmit data.
UCCSB (Unified Congestion Control for flows sharing the Same Bottleneck)	Fairness	First identifies the bottlenecks shared by flows from the same connection, and then performs an unified congestion control for those flows so that they compete for the bottleneck bandwidth fairly with other TCP flows.
SFR-CACC (Split Fast Retransmit Changeover Aware Congestion Control)	Spurious retransmission	Introduces a per destination virtual queue within the senderâĂŹs retransmission queue. The sender uses SACK along with history information in the retransmission queue to deduce missing reports for a segment by inferring cumulative ACK and gap reports per destination.
Cwnd Updates	Cwnd slow growth	First tracks the earliest outstanding Transmission Sequence Number (TSN) per destination and then uses SACKs and history information to deduce missing reports for a segment by inferring cumulative ACK and gap reports per destination. Therefore, the algorithm can update the cwnd even in the absence of new cumulative ACKs.
DAC (Delayed ACK for CMT)	Spurious retransmission	Delays sending an ACK if an out-of-order segment arrives at the receiver.
RTX-LCS (Lossrate, Congestion window and Slow start threshold)	Spurious retransmission	Prioritizes the retransmission through the subflow with the largest cwnd. If subflows have the same cwnd, the retransmission is made through the subflow with the largest ssthresh. If subflows have equal ssthresh, the retransmission is sent through the subflow with the lowest loss rate. Otherwise, a subflow is selected randomly.
FCCS (Flow and Congestion Control Separation)	Load sharing	Separates the association (or connection) flow control from congestion control. The flow control is on association basis. Both endpoints use their association buffer to hold the data chunks from all paths. Congestion control is performed on per path basis. Thus, the sender has a separate congestion control for each path.
CMT-PF (CMT Potentially Failed scheme)	Packet reordering	Marks a path that experiences a single timeout as a "potentially failed" path so that no further data transmission is allowed on that path. To detect its status, the sender sends heartbeat packets to the receiver. Upon heartbeat ACKs, the sender re-enables the path for data transmission again.
Buffer Splitting	Packet reordering	Splits the shared sender buffer size into n (i.e. number of paths) fixed per-path sections. A new chunk on a path can only be sent if its own buffer share has available space.
Chunk Rescheduling	Packet reordering	For each retransmission, searches the first chunk blocking the removal of chunks on the path from the sender buffer. That chunk is rescheduled on the path immediately when the congestion window has available space. Chunk Rescheduling is triggered when the path blocks more than half of the path's buffer share.
Smart Fast Retransmission	Spurious retransmission	Does not consider chunks that are moved from a path in the decision about fast retransmissions on a new path.
Multi-streaming	Packet reordering	Assigns each message a stream identifier. Each stream is sent over a certain path. It only needs to restore the sequence of streams belonging together. Hence, after a packet loss only messages of the affected streams have to be delayed to restore the sequence.
CMT/RP (CMT/Resource Pooling)	RP	Takes the interaction of the congestion controls on different subflows into account instead of handling them independently. One of its key operations, for example, is that it incorporates the possibility of shared bottlenecks by trying to halve the overall congestion window on the lossy path. Valid only if paths have similar characteristics.
FPS (Forward Prediction Scheduling)	Packet reordering	Takes account of the transmission delay of each path and schedules the specific data unit accordingly on each path so that the data arrive at the receiver in order.
Buffer Splittingv2	Packet reordering	Splits the shared sender buffer space dynamically. For instance, it grants more buffer space to faster paths so that they become more likely to send data.
CMT/RPv2 (CMT/Resource Pooling Version 2)	RP	Overcome CMT/RP-SCTP limitations by considering different path characteristics.
BERP (Bandwidth Estimation Based Resource Pooling)	RP	Applies the RP principle based upon the bandwidth estimates obtained by observing the data flow on the paths.

Table 2.9: Concurrent Multipath Transfer Protocols based on SCTP.

Scheme	Year	Algorithm and Protocol	Paths	Receive Buffer	RP & Fairness	Sequence Space	Network Environment
BA-SCTP [59]	2003	WRR-PULL, UCCSB, SACK	General	Constrained	Fairness	Single	General
W-SCTP [60]	2004	EDPF, SACK, Westwood	Disjoint	Not specified	No	Single	General
CMT-SCTP [61]	2004	SFR-CACC, Cwnd Updates, DAC, SACK	Independent	Infinite	No	Single	General
CMT-SCTP [62]	2004	Retransmission policies, SACK	Independent	Infinite	No	Single	General
LS-SCTP [71], [72]	2004	FCCS, WRR, SACK	General	Constrained	No	Double	
CMT-SCTP [63]	2006	SFR-CACC, Cwnd Updates, DAC, Retransmission policies, SACK	Independent	Constrained	No	Single	General
cmpTCP [73]	2006	WRR, SACK	Independent	Infinite	No	Single	General
WiMP-SCTP [74]	2007	WRR, SACK	Independent	Not specified	No	Single	Wireless
CMT-SCTP [64]	2008	RTX-LCS, SACK	General	Constrained	No	Single	General
cmpSCTP [75]	2008	FCCS, SACK, WRR	General	Constrained	No	Double	General
mSCTP-CMT [76]	2009	SACK, CMT-PF	Disjoint	Constrained	No	Single	Wireless
CMT-SCTP [65]	2010	Buffer Splitting, Chunk Rescheduling, Smart Fast Retransmission, SACK	General	Constrained	No	Single	General
CMT-SCTP [66]	2010	Multi-streaming, SACK	General	Constrained	No	Single	General
CMT-SCTP [67]	2010	CMT/RP, SACK	Similar paths	Not specified	Yes	Single	General
FPS-SCTP [77]	2010	FPS, SACK	Disjoint	Constrained	No	Single	Mobile
CMT-SCTP [68]	2011	Buffer Splittingv2, SACK	General	Constrained	No	Single	General
CMT-SCTP [69]	2011	CMT/RPv2, SACK	General	Constrained	Yes	Single	General
CMT-SCTP [70]	2011	BERP, SACK	General	Not specified	Yes	Single	Wireless

Protocol Data Unit (APDU)) are “bundled”in data “chunks ”along with SCTP control (“chunks ”). A control chunk can be a cumulative acknowledgment for instance. Contrary to TCP or UDP packets with control information in the header and then an optional data field, SCTP packets consist in a simple header (Source and destination ports, verification tag and checksum) followed by one or more chunks of either control or data information.

3. multihoming. High-availability application looking at increasing their availability may want to open several connections in parallel, preferably with different ips as most likely IPs are Network Interface Controller (NIC)-bound and different interfaces are also more likely to increase disjointness and path diversity.

SCTP embeds many features that previously had to be implemented by applications. Obviously this does not come for free and require existing applications to be upgraded to use the new Application Programming Interface (API). This drawback along with middlebox interference have inhibited its success, yet it allowed its successors to learn from this and influenced the design of its successors QUIC and MPTCP. Vanilla SCTP uses multihoming for redundancy purpose (network fault tolerance) but CMT extensions exist.

In this section, we divide the approaches into two groups differentiated by whether they have considered fairness.

Multipath Transmission without Considering Fairness:

Unlike TCP, SCTP was designed with multihoming in mind that an SCTP association allows multi-homed source and destination endpoints. Nevertheless, SCTP uses only one primary path and switches to another path for retransmission of lost packets, or as a backup in the case of failure from the primary path. Note that SCTP uses a single buffer structure on both endpoints, but maintains several states per destination: separate congestion window (cwnd), slow start threshold (ssthresh), retransmission timer, and RTT estimate. Several SCTP extensions, such as BA-SCTP [59], W-SCTP [60], CMT-SCTP [21], [61]–[64], LS-SCTP [71], [72], WiMP-SCTP [74], cmpSCTP [75], mSCTP-CMT [76] and FPS-SCTP[77], enable SCTP to transmit data over multiple paths simultaneously.

Argyriou et al. [59] proposed BA-SCTP, a bandwidth aggregation protocol based on SCTP. BA-SCTP implements a mechanism for identifying bottlenecks that are shared by flows from the same aggregate connection. Based on this mechanism, BA-SCTP performs a unified congestion control algorithm for the flows that share the same bottleneck (we name this algorithm UCCSB) instead of applying congestion control for each flow separately. This design guarantees that BA-SCTP flows are fair with the other TCP flows sharing the same bottleneck. BA-SCTP employs a WRR style scheduling strategy, a congestion window based data allocation strategy where each subflow pulls data from the shared sending buffer whenever the subflow has congestion window space to send data. This strategy assumes the congestion window to be a true representative of the bandwidth-delay product of the subflow compared to an estimated product. In the rest of the thesis, we use

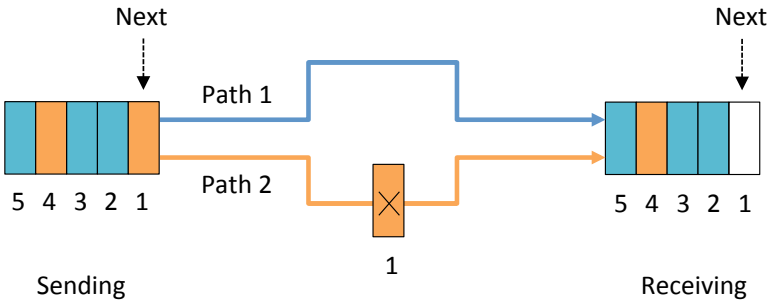

Figure 2.6: Head-of-Line blocking (HoL): the receive buffer cannot accommodate other chunks any more before the arrival of the head-of-line chunk (chunk 1).

WRR-PULL to denote congestion window based data allocation strategy.

Casetti et al. [60] proposed W-SCTP, a Westwood [175] flavored SCTP to exploit bandwidth aggregation. The authors believed that Westwood style congestion control could fully exploit the advantages of bandwidth estimation which could be utilized for traffic allocation among multiple flows. W-SCTP uses a EDPF style scheduler that chooses the path for next packet by predicting whether it can deliver the packet the fastest to the destination.

Iyengar et al. [61], [63] proposed integrating CMT capability into SCTP, namely CMT-SCTP. CMT-SCTP utilizes the multihoming feature from SCTP to correctly transfer data between multi-homed end hosts. They identified three negative side-effects of CMT, and proposed algorithms to solve them accordingly. First, they proposed a Split Fast Retransmit Changeover Aware Congestion Control (SFR-CACC) algorithm to eliminate the unnecessary fast retransmissions by using a different interpretation of SACK information. Second, they used a congestion window (cwnd) growth algorithm to track the earliest outstanding TSN per destination and update the cwnd, even in the absence of new cum ACKs. Third, they proposed a new Delayed ACK algorithm for CMT-SCTP, namely Delayed ACK for CMT (DAC). The algorithm allows the receiver to delay sending ACK of an out-of-order segment. In [62], [63], they proposed five retransmission policies for CMT. They demonstrated the occurrence of spurious retransmissions with all of those policies, and proposed amendment algorithms to avoid them.

There has been a considerable amount of work on the core CMT-SCTP [63] to overcome its defect and improve performance. In [64], Liu et al. found

that all of the five retransmission policies may cause throughput degradation due to receive buffer blocking. This blocking problem is also named HoL. Figure 2.6 illustrates a simplified example of it. As shown in the figure, a CMT-SCTP receiver maintains a single receive buffer which is shared across two sub-association flows in an association. The C_1 (chunk 1) is transmitted through the path 2 and is lost due to traffic congestion or path failure. During the time period of C_1's retransmission, the receive buffer cannot accommodate any other packets due to flow control so that the overall throughput degrades. To solve the problem, Liu et al. proposed a compound parameter retransmission policy named RTX-LCS. It limits the retransmission path selection by considering three common conditions: cwnd, ssthresh, and loss rate. Dreibholz and Adhari et al. [65], [66], [68] examined the challenges of CMT-SCTP over dissimilar paths. They identified the issues of sender and receiver queue blocking, which may lead to poor overall performance. In order to improve performance, Dreibholz [65] proposed multiple mechanisms accordingly, including Buffer Splitting, Chunk Rescheduling, and Smart Fast Retransmission. Buffer Splitting is used to avoid one path occupying too much buffer space, which prevents other paths from sending out new chunks. Chunk Rescheduling copes with the problem of certain delayed or lost chunks stalling the whole transmission. Smart Fast Retransmission deals with spurious fast retransmission bursts. For example, it does not consider chunks being moved from another path in the decision about fast retransmissions on the new path. In [68], they presented an optimized buffer handling technique to further improve performance. Specifically, they proposed to use the shared buffer space dynamically so that a faster path can have the possibility to send more data by granting it more buffer space. We use Buffer Splittingv2 to denote this updated version of Buffer Splitting mechanism. Moreover, in [66], they proposed using the Multi-streaming feature of SCTP to mitigate the HoL problem. Specifically, each message is assigned an identifier to indicate a stream. With this identifier, the protocol only needs to restore the sequence of messages belonging together. Hence, after a packet loss, only the messages of the affected streams have to be delayed to restore the sequence. The other messages can be processed immediately without delay.

The authors in LS-SCTP [71], [72] and cmpSCTP [75] proposed separating the association flow control from per path congestion control (denoted

as FCCS). The congestion control is performed per path, whereas the flow control is performed per association. In order to achieve this goal, LS-SCTP uses two different sequence numbers. The first one is the Association Sequence Number (ASN) that is used to reorder the received data at the receiver association buffer. The second one is the Path Sequence Number (PSN) that is used for reliability and congestion control on each path. The scheduling module of LS-SCTP uses the current congestion window (cwnd) of each path as an estimate of its current bandwidth-delay product. For example, it assigns data to the paths according to the cwnd/RTT of each path. cmpSCTP distributes data over available paths based on real-time bandwidth estimation of each path. Similar to LS-SCTP, cmpSCTP also distributes the data on the available paths based on the estimation of the available bandwidth of each path. Thus, they all use a WRR style data scheduler.

Sarkar [73] proposed Concurrent Multipath TCP (cmparallel TCP (pTCP)), an extension of SCTP. cmpTCP splits packets concurrently over all available paths from a shared sending buffer. cmpTCP maintains a virtual retransmission queue (RTxQ) on each path to control the number of outstanding bytes on the path. The receiver sends back ACKs on the same path on which the packets are received. These two designs may help to ignore spurious gap reports and eliminate unnecessary packet retransmissions. Sarkar also developed a Markov model in cmpTCP to estimate the data transport rate on each path when the transmission has reached a steady state. cmpTCP uses a WRR style scheduler by considering the number of outstanding bytes and congestion window size.

Huang et al. [74] proposed Wireless Multi-Path SCTP (WiMP-SCTP), which devised two data transmission modes, i.e., Data-striping Mode and Data-duplicating Mode, for multipath transmission in multiple wireless access networks. When the network status is good, the Data-striping Mode is selected to aggregate bandwidth. On the other hand, when the network status becomes bad, the Data-duplicating Mode is selected to increase destination reachability. To switch between the two modes, a mode selection scheme that determines the status of these multiple paths was proposed. Specifically, it designed a HEARTBEAT scheme where heartbeat chunks are sent periodically on the paths. The transmission error counter increases when a heartbeat is not acknowledged within one retransmission timeout

interval. If the number of transmission error counter plus the number of consecutive retransmissions exceeds a certain threshold, the Data-duplicating Mode is switched on. Otherwise, the Data-striping mode is used. Nevertheless, Huang et al. did not present a complete explanation of the scheduling algorithm used by WiMP-SCTP. They only mentioned that the sender transmits data as soon as the corresponding receive window at the receiver side allows data to be sent. We speculate that this is a variant of the WRR style scheduling algorithm.

Budzisz et al. [76] proposed an mSCTP-CMT protocol to investigate the applicability of using CMT-SCTP to distribute data between two paths during the handover transition process. They emphasized the consequence of a sender-introduced reordering and its effect on congestion control. The authors found that in CMT-SCTP the receive buffer may be filled with out-of-order data caused by complete or short-term failures during handover. Although handover is out of the scope of this survey, the handover scenario is very similar to the worst case in CMT where a path experiences a long delay suddenly. To solve this problem, they proposed using Potentially Failed (CMT-PF) scheme that a path experiencing a single timeout is marked as "potentially failed" and no further data transmission is allowed on that path. They utilized a heartbeat scheme to probe whether the potentially-failed path has got back to a positive state in the case of successful heartbeat acknowledgment.

Mirani et al. [77] proposed a multipath Forward Prediction Scheduling (FPS) for SCTP, namely FPS-SCTP. In order to reduce the number of out-of-order packets, it estimates the arrival time of each packet in advance and decides which packet is to be sent through a certain path so that the packets can arrive at the destination in order. They used roughly a half of the RTT on a subflow to estimate the one trip time from data leaving the send buffer to being received at the receiver. This prediction is an approximation considered as too coarse because previous studies have shown that a majority of Internet connections experience latency asymmetry [176], [177].

Multipath Transmission Considering Fairness:

The work discussed previously on CMT-SCTP performs independent congestion control on each path, and considers little about the fairness against other single-path flows. For example, in the case of n CMT-SCTP

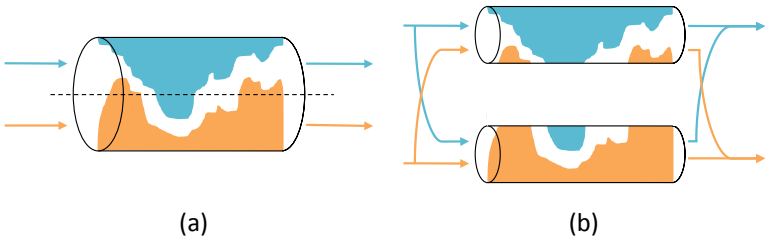

Figure 2.7: RP (a) a single path shares its resource fairly to competing flows (b) multiple paths are treated as a single pooled resource. RP: Resource Pooling.

paths, the association will get n times the bandwidth share of a competing non-CMT SCTP or TCP flow over the same bottleneck.

The concept of RP [155] is a milestone for multipath transmission aggressiveness control. In the context of multipath transmission, it makes a collection of resources behave like a single resource by balancing traffic across multiple paths. As shown in Figure 2.7, when several TCP flows compete for a single path, TCP can share the path's capacity fairly among them. When the flows go through more than one path, the paths are treated as a single pooled resource. With appropriate coordination, traffic can move away from more congested paths to less congested ones, and larger bursts can be accommodated.

Dreibholz et al. [67] proposed a RP congestion control for CMT-SCTP denoted as CMT/RP-SCTP by combining CMT-SCTP with the concept of RP. The goal of CMT/RP-SCTP is to improve the data throughput while still remaining fair to concurrent single-path flows on the shared bottleneck. For example, when two paths are used concurrently for data transmission and they share a bottleneck link, the overall throughput obtained by a CMT/RP-SCTP association should be similar as that of a standard SCTP. However, CMT/RP-SCTP assumes similar paths, i.e., paths having very similar characteristics in terms of bandwidth, delay and loss rate. Dreibholz et al. [69] proposed an updated version of CMT/RP-SCTP (denoted as CMT/RPv2-SCTP) to overcome the limitations of CMT/RP-SCTP by considering path bandwidths. They studied the behavior of CMT/RPv2-SCTP on dissimilar paths and found that CMT/RPv2-SCTP achieves the goals of RP. Furthermore, they observed that compared with MPTCP (which will be

discussed later), CMT/RPv2-SCTP distributes bandwidth to flows equally when possible regardless of the number of paths used for transport.

Note that both CMT/RP-SCTP and CMT/RPv2-SCTP apply RP principle only during the congestion control phase. Shailendra et al. [70] argued that this strategy is not optimum on heterogeneous networks with wireless links because losses in wireless links may happen because of reasons other than congestion. Hence they considered the resources to be as a single pool of resources during the congestion detection phase as well. To achieve this goal, they proposed a Bandwidth Estimation Based Resource Pooling (BERP) algorithm which applies the RP principle based upon the bandwidth estimates obtained by observing the data flow on the paths.

TCP based Multipath Transmission

In contrast to SCTP, which was designed with multihoming support in nature, TCP is unaware of multiple interfaces and allows only a single IP address per endpoint. Nevertheless, TCP has dominated the Internet traffic and has sparked a lot of interests in enabling TCP to support simultaneous multipath transmission. In this section, we divide the approaches in four groups. The grouping principle is influenced by the research issues the approaches aim to address, such as fairness, buffer impact on performance, Pareto-optimality and path diversity. Although some approaches may cover more than one research issue, we only discuss them in the group which we believe the approaches are best fit into. In our study, we found that SCTP and MPTCP share many similar issues and certain algorithms. At the end of this section, we summarize their common features as well as their differences.

Multipath Transmission without Considering Fairness:

Magalhaes et al. [78] proposed Reliable Multiplexing Transport Protocol (R-MTP), which is a rate-based reliable transport protocol multiplexing data across multiple network interfaces (i.e., a WRR style scheduler). It relies on explicit bandwidth probing via the packet pair method [178] to estimate bandwidth in order to adjust the rate on the available paths accordingly. For example, it measures packet inter-arrival times and jitter to sense bandwidth scarcity. The probing period should occur on a fine time-scale to reflect the fluctuation of the available bandwidth.

Table 2.10: Key algorithms for TCP based CMT.

Scheme	Year	Algorithm and Protocol	Paths	Receive Buffer	Fairness	Sequence Space	Network Environment
R-MTP [78]	2001	SACK, WRR	Disjoint	Not specified	No	Single	Mobile
Lee et al. [79]	2002	IFRT, DACv2, PFA	General	Not specified	No	Single	General
pTCP [80]–[82]	2002, 2005	WRR-PULL, Delayed Binding, Packet Restriping, Redundant Striping, SACK	Independent	Constrained	No	Double	Mobile
R^2CP [83], [84]	2003, 2005	EDPF, Packet Restriping	General	Constrained	No	Double	Wireless
Cetinkaya et al. [85]	2004	OMS	Independent	Constrained	No	Single	General
mTCP [86]	2004	WRR, Shared Congestion Detection	Disjoint	Constrained	Yes	Single	General
M-TCP [87]	2004	Duplicate Transmission	Disjoint	Not specified	No	Single	Ad hoc
M/TCP [88]	2004	OWTT, WRR, Duplicate Transmission, Duplicated ACK, Duplicated & Delayed ACK	Independent	Not specified	No	Single	General
R-M/TCP [89]	2005	OWTT, WRR, Duplicate Transmission, RCC	Independent	Constrained	No	Single	General
cTCP [90]	2007	WRR, Duplicated ACK classifier	Independent	Constrained	No	Single	General
MPLOT [91], [92]	2008, 2012	Packet coding, ECN, EDPF	General	Not specified	No	Single	Lossy
JOSCH [93]	2009	WRR	Independent	Constrained	No	Single	Wireless
Super-aggregate [94]	2009	Selective Offloading, IP-in-IP encapsulation, Duplicate Transmission	Independent	Constrained	No	Single	Mobile
BMC [95]	2009	WCC	General	Not specified	Yes	Single	General
MPTCP [96]–[98]	2009	LIA, WRR	General	Constrained	Yes	Double	General
MPTCP [99]	2010	LIA, WRR	General	Constrained	Yes	Double	Data center
MPTCP [100]	2011	LIA, WRR-PULL	General	Constrained	Yes	Double	General
Hassayoun et al. [101]	2011	DWC, WRR-PULL	General	Constrained	Yes	Double	General
MPTCP [103], [104]	2012, 2013	OLIA, WRR-PULL	General	Constrained	Yes	Double	General
Han et al. [109]	2012	EDPF	Independent	Constrained	Yes	Double	General
NC-MPTCP [110]	2012	Packet coding, FPS	Independent	Constrained	Yes	Double	General
FMTCP[111], [112]	2012, 2014	Packet coding, FPS	Independent	Constrained	Yes	Double	General
MPTCP [102]	2012	Opportunistic Retransmission (OR), Penalizing slow subflows	General	Constrained	Yes	Double	General
QoS-MPTCP [113]	2012	Partial Reliability	Independent	Infinite	Yes	Double	General
Peng et al. [117]	2013	Balia	General	General	Yes	Double	General
MPTCP/OF [115]	2013	OF	Independent	General	Yes	Double	General
MPTCP [105]	2013	New Delayed ACK, packet coding	Independent	Constrained	Yes	Double	General
MPTCP [106]	2013	Penalizing slow subflows (Improved)	General	Constrained	Yes	Double	General
CWA-MPTCP [114]	2013	CWA, FPS	General	Constrained	Yes	Double	General
Singh et al. [107]	2013	AOLIA, EDWC, PSPLH	General	Constrained	Yes	Double	General
Yang et al. [119]	2013	NR-SACKs	General	Constrained (Send buffer)	Yes	Double	General
Lim et al. [121]	2014	Detect MAC-Layer path status	General	General	Yes	Double	Wireless
Ferlin et al. [124]	2014	DRePaS	General	General	Yes	Double	Wireless
SC-MPTCP [120]	2014	Packet coding, FPS	Independent	Constrained	Yes	Double	General
EW-MPTCP [122]	2014	WCC	Independent	Constrained	Yes	Double	General
Yang and Amer [123]	2014	FPS	Independent	Infinite	Yes	Double	General
Le et al. [108]	2015	FPS	Independent	Infinite	Yes	Double	General

Table 2.11: Concurrent Multipath Transfer Protocols based on TCP.

Scheme	Year	Algorithm and Protocol	Paths	Receive Buffer	Fairness	Sequence Space	Network Environment
R-MTP [78]	2001	SACK, WRR	Disjoint	Not specified	No	Single	Mobile
Lee et al. [79]	2002	IFRT, DACv2, PFA	General	Not specified	No	Single	General
pTCP [80]–[82]	2002, 2005	WRR-PULL, Delayed Binding, Packet Re-striping, Redundant Striping, SACK	Independent	Constrained	No	Double	Mobile
R^2CP [83], [84]	2003, 2005	EDPF, Packet Re-striping	General	Constrained	No	Double	Wireless
Cetinkaya et al. [85]	2004	OMS	Independent	Constrained	No	Single	General
mTCP [86]	2004	WRR, Shared Congestion Detection	Disjoint	Constrained	Yes	Single	General
M-TCP [87]	2004	Duplicate Transmission	Disjoint	Not specified	No	Single	Ad hoc
M/TCP [88]	2004	OWTT, WRR, Duplicate Transmission, Duplicated ACK, Duplicated & Delayed ACK	Independent	Not specified	No	Single	General
R-M/TCP [89]	2005	OWTT, WRR, Duplicate Transmission, RCC	Independent	Constrained	No	Single	General
cTCP [90]	2007	WRR, Duplicated ACK classifier	Independent	Constrained	No	Single	General
MPLOT [91], [92]	2008, 2012	Packet coding, ECN, EDPF	General	Not specified	No	Single	Lossy
JOSCH [93]	2009	WRR	Independent	Constrained	No	Single	Wireless
Super-aggregate [94]	2009	Selective Offloading, IP-in-IP encapsulation, Duplicate Transmission	Independent	Constrained	No	Single	Mobile
BMC [95]	2009	WCC	General	Not specified	Yes	Single	General
MPTCP [96]–[98]	2009	LIA, WRR	General	Constrained	Yes	Double	General
MPTCP [99]	2010	LIA, WRR	General	Constrained	Yes	Double	Data center
MPTCP [100]	2011	LIA, WRR-PULL	General	Constrained	Yes	Double	General
Hassayoun et al. [101]	2011	DWC, WRR-PULL	General	Constrained	Yes	Double	General
MPTCP [103], [104]	2012, 2013	OLIA, WRR-PULL	General	Constrained	Yes	Double	General
Han et al. [109]	2012	EDPF	Independent	Constrained	Yes	Double	General
NC-MPTCP [110]	2012	Packet coding, FPS	Independent	Constrained	Yes	Double	General
FMTCP[111], [112]	2012, 2014	Packet coding, FPS	Independent	Constrained	Yes	Double	General
MPTCP [102]	2012	OR, Penalizing slow subflows	General	Constrained	Yes	Double	General
QoS-MPTCP [113]	2012	Partial Reliability	Independent	Infinite	Yes	Double	General
Peng et al. [117]	2013	Balia	General	General	Yes	Double	General
MPTCP/OF [115]	2013	OF	Independent	General	Yes	Double	General
MPTCP [105]	2013	New Delayed ACK, packet coding	Independent	Constrained	Yes	Double	General
MPTCP [106]	2013	Penalizing slow subflows (Improved)	General	Constrained	Yes	Double	General
CWA-MPTCP [114]	2013	CWA, FPS	General	Constrained	Yes	Double	General
Singh et al. [107]	2013	AOLIA, EDWC, PSPLH	General	Constrained	Yes	Double	General
Yang et al. [119]	2013	NR-SACKs	General	Constrained (Send buffer)	Yes	Double	General
Lim et al. [121]	2014	Detect MAC-Layer path status	General	General	Yes	Double	Wireless
Ferlin et al. [124]	2014	DRePaS	General	General	Yes	Double	Wireless
SC-MPTCP [120]	2014	Packet coding, FPS	Independent	Constrained	Yes	Double	General
EW-MPTCP [122]	2014	WCC	Independent	Constrained	Yes	Double	General
Yang and Amer [123]	2014	FPS	Independent	Infinite	Yes	Double	General
Le et al. [108]	2015	FPS	Independent	Infinite	Yes	Double	General

Lee et al. supported two transmission modes in their work [79]: FOSM and POSM. In POSM, they investigated multiple schemes to address the spurious retransmissions by modifying two TCP operations: 1) Increasing the Fast Retransmit Threshold (IFRT) and 2) enabling Delayed ACKs for out-of-order packets as well as sending immediately ACKs for retransmitted packets. The second modified operation is like an advanced version of DAC used in [61], [63], thus, we name it DACv2. IFRT makes the TCP sender wait for more than triple duplicate ACKs, which reduces the number of the fast retransmission and the fast recovery events. DACv2 enhances performance because when ACKs are being delayed, new packets may fill the gap and change the out of order packets in order.

PTCP [80]–[82] functions as a wrapper around a modified version of TCP. It opens multiple TCP flows, one for each interface in use. pTCP performs data-striping across multiple micro-flows (TCP flows) by considering their bandwidth difference. Specifically, pTCP uses cwnd/RTT ratio, a WRR-PULL scheduler, to allocate traffic proportionally to path capacity. In addition, pTCP has several other strategies addressing specific problems. For example, the congestion window could be an over-estimate especially just before congestion occurs. This can result in an undesirable hold up of data in subflows. Instead of reassigning data to other subflows later on, pTCP uses a Delayed Binding strategy to adapt to instantaneous changes in path capacity. Specifically, it pulls data from the shared sending buffer only when the data is scheduled to send out immediately through a subflow. In order to avoid an overflow of the receive buffer, pTCP uses a Packet Re-striping strategy to retransmit a packet through a different subflow instead of the subflow which transmitted that packet earlier, and uses a Redundant Striping strategy to send a duplicated packet on one subflow to another. Moreover, pTCP uses SACK feedback mechanism to recover a lost packet in a much shorter time period.

Reception Control Protocol (RCP) [83], [84] is a receiver-centric transport protocol with a minimized sender design. The receiver controls all the key functions in RCP. To support CMT, a multi-state extension of RCP, i.e., Radial RCP (R^2CP), was proposed. R^2CP maintains one RCP pipe (the same as a TCP flow) per end-to-end path with congestion control being handled by individual RCP pipes. Traffic is scheduled to each RCP pipe based on the (estimated) time the requested segment will arrive through the

concerned pipe (a variant of EDPF). Note that each RCP pipe maintains a local sequence number space internally to facilitate loss detection and recovery. The local sequence number can be converted to the global sequence number, and vice versa.

Chen et al. [87] proposed M-TCP that uses a Duplicate Transmission mode for the lossy wireless environment with high interference. In this transmission mode, multiple copies of the same packet are sent on different paths so that the chance that all copies are lost is much reduced. Unfortunately, they only present sending-side modification without addressing duplicate ACKs due to multiple copies of the same packet.

Rojviboonchai and Hitoshi [88] proposed a multipath Transmission Control Protocol (M/TCP). M/TCP uses OWD [179], a similar method as FPS, at the sender to estimate the delay time of the forward path and reverse path separately in order to calculate per path RTO timer. In addition, M/TCP employs two mechanisms to deal with packet loss. In the case of fast retransmission, M/TCP uses Duplicate Transmission policy, i.e., duplicating the missing segment and sending each copy through all paths so that a quick and reliable retransmission can be desirable; in the case of timeout retransmission, the missing segment on a flow is sent through the other flows. Moreover, M/TCP uses two algorithms for the receiver to transmit an ACK in the case of CMT. Namely, using Duplicated ACK algorithm, an M/TCP receiver sends an ACK immediately upon the receipt of a data segment to more than one path; using Duplicated & Delayed ACK, the receiver transmits an ACK for every other data segments through more than one path. Rojviboonchai et al. proposed R-M/TCP [89] as an extension of their previous work M/TCP. R-M/TCP is a rate-based M/TCP that performs congestion control in a rate-based and loss avoidance manner (we use RCC to denote it) to avoid packet loss by adjusting the congestion window before buffer overflows. Specifically, R-M/TCP schedules data packets in a WRR manner while it estimates the queue length at the bottleneck link. If the queue length grows beyond a predefined threshold, the sender recalculates a new congestion window to achieve a fair share at the bottleneck.

Cetinkaya and Knightly [85] proposed an Opportunistic Multipath Scheduler (OMS) that follows a traffic splitting policy that favors low-delay high-throughput paths opportunistically for a short term variations in path quality. To avoid violating path weights of the routing protocol and potentially

leading to oscillation, OMS ensures that over longer time scales traffic is split according to the ratios determined by the routing protocol.

Dong, Pissinou, and Wang [90] proposed that uses a single congestion window to control the global throughput and a single sending buffer to be shared among all paths. It uses Credit-Weighted Round-Robin (a variant of WRR) as the scheduling algorithm. Each time an ACK comes back to the sender, the capacity estimation of that path is updated, and a new sending credit (similar to the congestion window size) is added to the sender. The new credit is for all the paths combined, and it is further divided into each path. cTCP uses the path credit (similar to the path capacity) to distribute data among the available paths. Furthermore, cTCP adopts a duplicated ACK classifier that handles packet reordering by differentiating whether a duplicated ACK is likely caused by CMT or a real duplicated ACK.

Sharma et al. [91], [92] proposed Multi-Path LOss-Tolerant protocol (MPLOT) to provide multipath transmission on multiple heterogeneous, highly lossy paths. MPLOT uses erasure based FEC packet coding. The major benefit of packet coding stems from its ability to compensate for missing packets from redundancy. This makes data transmission over lossy networks robust and efficient. To counter against packet reordering, MPLOT estimates path parameters (i.e., loss rate, capacity, and RTT) continuously to provide adaptive FEC coding. In particular, MPLOT performs latency-aware packet mapping, a variant of EDPF. For example, it maps packets that are not required immediately to paths with long delays, while mapping the more immediately useful packets to paths with short delays. MPLOT uses Explicit Congestion Notification (ECN) [180] to distinguish congestion losses from those due to faulty/lossy links.

Wang et al. [93] proposed a segment-based adaptive Joint Session Scheduling (JOSCH) mechanism. The main goal is to restrain the delay difference among multiple Radio Access Networks (RANs)s (RANs) by means of allocating the traffic to different RANs dynamically with reasonable ratios. Specifically, JOSCH obtains network conditions by a segment-based feedback approach, where a "segment" is defined as a predetermined size of data block. Its size is configurable according to the delay sensitivities of different services. After each segment transmission, the receiver sends feedback to the sender. According to the feedback, the sender adjusts traffic allocation dynamically according to the estimated transmission rates and

delays.

Tsao and Sivakumar [94] argued that aggregated bandwidth of two wireless interfaces (3G and WiFi) is a Simple Aggregation due to path heterogeneity. For example, the low bandwidth interface (e.g., 3G with 100-500Kbps) can only achieve negligible bandwidth compared to the high bandwidth interface (e.g., WiFi with 2-54Mbps). They proposed a super-aggregation to achieve performance that is better than the sum of throughput achievable through each of the interfaces individually by the means of three mechanisms: Selective Offloading, Proxying, and Mirroring. In spite of the fact that an interface may have a relatively small amount of bandwidth, these mechanisms can provide considerable performance improvement. Specifically, the Selective Offloading mechanism is to receive TCP data segments over a comparably high-speed WiFi and return ACKs over a low-speed 3G path to address self-contention in WiFi networks. The Proxying mechanism, following IP-in-IP encapsulation, allows a 3G path to notify the TCP sender about blackout events on the WiFi path. The Mirroring mechanism (i.e., Duplicate Transmission) establishes an addition TCP connection through 3G to fetch the missing segments due to random loss on the WiFi path.

Multipath Transmission Considering Fairness:

Similar to the early development of SCTP based CMT, at the early stage, TCP based multipath transmission was only used for utilizing multiple TCP flows with intelligent scheduling algorithms to mitigate packet reordering. Nevertheless, it was found that simply utilizing multiple TCP flows concurrently at a bottleneck would result in a fairness issue, i.e., an unfair share of the bandwidth at a bottleneck link. For example, NewReno [181] is the most common TCP congestion control variant as it yields an equal share of the congested link. This equal share outcome of NewReno results in an unfair share of the bandwidth if more than one TCP flow is active for a single multipath transmission connection at the bottleneck link. From the literature review we have made, we find that multipath transmission approaches based on TCP in recent years have also started to make fairness a necessary feature.

To the best of our knowledge, mTCP [86] proposed by Zhang et al. is among the earliest proposals that have taken fairness issue into consideration for TCP based multipath transmission. To address the fairness issue, they

proposed not establishing multiple flows through the same bottleneck. For example, to alleviate the aggressiveness problem, Zhang et al. integrated a shared congestion detection mechanism into mTCP so as to identify and suppress subflows that traverse the same set of congested links. For example, mTCP detects shared congestion by examining the correlations among the fast retransmit times of the subflows. mTCP also uses a scheduler in a WRR manner. For example, it maintains a counter, i.e., $pipe_i$, to represent the number of outstanding packets on the i_{th} path. $pipe_i$ is incremented by 1 when the sender either sends or retransmits a packet over the i_{th} path, and is decremented by 1 when an incoming ACK indicates that a packet previously sent has been received. The sender associates a score, i.e., $pipe_i/cwnd_i$, for each path. The path with the minimum score has priority to send the next packet.

Instead of avoiding shared bottlenecks, Honda et al. [95] proposed Bidimensional-Probe Multipath Congestion Control (BMC) to address the fairness issue. Specifically, BMC uses a Weighted Congestion Control (WCC) approach that applies the weight to each subflow so that the throughput of each subflow is in proportion to its weight. In addition, WCC maintains the sum of the weight so that a bundle of subflows in the multipath connection is kept as aggressive as one TCP flow. WCC can achieve not only fair resource allocation at the shared bottleneck, but also RP [155] along the disjoint bottlenecks. For example, multiple different connections can obtain fair resource allocation across distinct bottlenecks.

Approximately in 2009, MPTCP [96]–[98] was proposed with the fairness property as well as RP feature in mind. Specifically, MPTCP, under discussion of IETF, has the following set of goals to achieve:

- Improve throughput: MPTCP should perform at least as a single TCP flow running on the best path.
- Do no harm: MPTCP subflows should not take more capacity than a single TCP flow would get at a shared bottleneck.
- Balance congestion: MPTCP should utilize the least congested path the most.

Figure 2.8 shows the architecture of MPTCP. It is a major extension to TCP and allows a pair of hosts to use several paths to exchange the segments

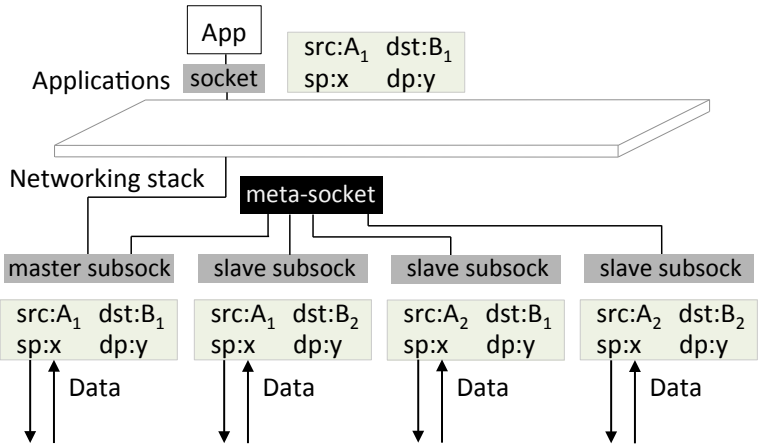

Figure 2.8: The architecture of MPTCP, which has the application compatibility by keeping the standard socket API to legacy applications. MPTCP: Multipath TCP, src: Source IP address, dst: Destination IP address, sp: Source Port, dp: Destination Port, APP: Application.

that carry the data from a single connection. MPTCP presents a standard TCP socket API to the upper layer so that legacy applications can run upon MPTCP transparently. A Coupled Congestion Control (CCC) algorithm, Linked Increase Algorithm (LIA) [182], is used to guarantee fair resource allocation on multiple paths and provide RP feature among them. Its RP feature can shift traffic away from more congested paths to less congested ones. In addition to the joint congestion control algorithm, MPTCP also has a few other design features. For example, MPTCP adds connection-level sequence numbers in order to reassemble the data stream in-order from multiple subflows. A Data Sequence Signal (DSS) option [183] specifies a full mapping from the connection-level sequence number to the subflow sequence number. In the early stage of MPTCP, a PUSH style WRR scheduling strategy is used where the scheduler tries to fill all subflows when there is data coming from the application. Later, MPTCP adopts a WRR-PULL manner scheduler [100], a similar design adopted by pTCP [80]–[82] where the application data stored in a shared connection-level sending buffer is pulled by subflows whenever they have space in their congestion window. Both PUSH and WRR-PULL scheduling strategies are variants of WRR. Their difference lies in the fact that WRR-PULL strategy uses less time waiting

in a subflow queue before its actual transmission on the wire. Although this time period seems minor, the path properties may change during that time. For the latest development of MPTCP in IETF, we refer readers to RFCs such as [16], [182], [183].

Raiciu et al. [99] were the first proposing a natural evolution of data center transport from TCP to MPTCP. They demonstrated that MPTCP could efficiently and seamlessly use available bandwidth, provide improved throughput and better fairness compared to single path TCP. The same authors further investigated what caused these benefits in [184]. They found that using MPTCP allows to rethink data center networks and approach them with a different mindset as to the relationship between transport protocols, routing, and topology. One of the challenges of deploying MPTCP in data centers is the Incast effect, a behavior of MPTCP as well as TCP that results in the gross under-utilization of link capacity in certain many-to-one traffic patterns [185]–[188]. Incast collapse is not specific to MPTCP, but is inherited from TCP. Li et al. [122] investigated how to share network resources among different MPTCP flows by performing an additional weighted congestion control based on the coupled congestion control mechanism.

Although LIA ensures MPTCP subflows to be no more aggressive than a competing TCP flow, LIA is not able to differentiate whether the subflows share the same bottleneck or different ones. This may cause sub-optimal performance of MPTCP. Hassayoun, Iyengar, and Ros [101] proposed a Dynamic Window Coupling (DWC) algorithm to address it by detecting distinct bottlenecks and only coupling those that share a common bottleneck, while allowing other subflows to use separate congestion control. For example, DWC uses a loss congestion event to trigger an alert while using either the Delay or Loss congestion event to group subflows. Singh et al. [107] proposed an extension of the DWC algorithm, denoted as EDWC. This extension uses a delay congestion to trigger an alert while either delay or loss congestion is used to group and couple subflows. The concept behind DWC and EDWC algorithms is similar to that of UCCSB algorithm used in BA-SCTP [59] because they both identify the shared bottleneck and guarantee that the aggregated flows on the bottleneck is TCP-friendly.

Diop et al. [113] proposed QoS-oriented MPTCP that takes advantage of the two sub-layers architecture of MPTCP to use Quality of Service (QoS) techniques for multimedia applications over multiple paths. MPTCP orig-

inally offers a fully reliable and fully ordered service. Nevertheless, full reliability may not be required by certain multimedia applications. Diop et al. investigated the QoS benefits induced by the implementation of the Partial Reliability [189] feature in MPTCP for interactive video applications. Partial Reliability is an important concept for multimedia transmission over IP networks and is defined as the possibility to not recover losses under a threshold in order to improve QoS.

Peng et al. [116], [117] presented a fluid model to investigate a few existing congestion control algorithms designed for MPTCP, and identified design criteria that guarantees the existence, uniqueness, and stability of system equilibrium. They characterized algorithm parameters for TCP friendliness and proved that there is an inevitable trade-off between responsiveness and friendliness. Based on the study, in [117] they proposed a new congestion control algorithm, Balia (balanced linked adaptation). This algorithm generalizes existing algorithms and strikes a good balance among TCP-friendliness, responsiveness, and window oscillation.

Lim et al. proposed MPTCP-MA [121] to improve MPTCP performance during intermittent path connectivity in wireless environment. MPTCP-MA exploits MAC-Layer information to estimate path status, and suspends/releases a path based on the estimation. By quickly detecting path failure and recovery, MPTCP-MA can avoid unnecessary losses and utilize recovered paths more quickly.

Buffer Impact on MPTCP:

Although MPTCP was designed with several merits such as fairness, RP, and Pareto-optimality in mind, buffer size has significant impact on MPTCP performance. This problem stems from the packet reordering issue due to heterogeneous path characteristics. We now discuss the related work which has explicitly examined the impact of buffer size on MPTCP. Note that the impact of buffer size is not limited to MPTCP but on all other approaches using POSM.

Barré, Paasch, and Bonaventure [100] evaluated the impact of heterogeneous paths on the receive buffer and aggregated throughput. The experiment result shows that losses on one subflow have a limited impact on the performance of the other subflows. Nevertheless, this observation is based on the assumption that the reordering buffer is big enough to accommodate all

the out-of-order data. Nguyen et al. [190], [191] evaluated the performance of MPTCP in terms of load sharing and throughput optimization with and without LIA respectively. The results show that the context of mismatched path characteristics has a great impact on the performance of MPTCP with constrained receive buffers. Han et al. [109] proposed a reordering scheme that considers packet scheduling algorithm at the sender to reduce the receive buffer. The main idea is to estimate packet arrival time and schedule packets accordingly. The sender maintains a per path time table including calculated values of receiving time at the receiver from now for each packet. When the sender has opportunity to send a new packet, it chooses the path that can deliver the packet faster than others. Thus, it is a EDPF style scheduling algorithm.

The impact of buffer size on MPTCP performance has also been observed in [105], [110]–[112], [120], which proposed packet coding based approaches to address it. For example, in [110] Li et al. proposed NC-MPTCP that introduces packet coding to some but not all subflows. The regular subflows deliver original packets while the coding subflows deliver linear coded packets. The coded packets are used to compensate for the lost and much delayed packets in order to avoid receive buffer blocking. They used an out-of-order scheduler that calculates the expected packet arriving time by taking RTT, throughput, and loss ratio into account. Thus, the packets that are sent out of order are expected to arrive at the receiver in order. This scheduling algorithm is the same as the FPS algorithm used in [77]. In [120], Li et al. proposed SC-MPTCP to mitigate the packet reordering issue with constrained receive buffer. In SC-MPTCP, they proposed to make use of coded packets only as redundancy to compensate for expensive retransmissions while minimizing the encoding/decoding operations. The redundancy is provisioned in both proactive and reactive manners. Specifically, SC-MPTCP transmits proactive redundancy first and then delivers the original packets. The proactive redundancy is continuously updated according to the estimated aggregate retransmission ratio. In order to avoid the proactive redundancy being underestimated, a pre-blocking warning mechanism is utilized to retrieve the reactive redundancy from the sender. Cui et al. [111], [112] proposed applying the fountain code for multipath scheduling to mitigate the impact of path heterogeneity. They also designed a data allocation algorithm based on the expected packet arriving time and decoding demand

to coordinate the transmissions of different subflows. Li et al. in [105] dealt with the packet reordering issue from a different perspective. They demonstrated that the traditional Delayed ACK mechanism can lead to significant performance degradation in the presence of timeouts. Thus, they proposed a New Delayed ACK (NDA) for MPTCP aiming to remove the Minimum RTO constraint at the sender while to reserve the Delayed ACK function at the receiver.

Raiciu et al. [102] proposed schemes of opportunistic retransmission and penalizing slow subflows to avoid the reordering problem. If a subflow holds up a packet at the trailing edge of the receive window, the opportunistic retransmission allows the sender to resend the packet that is previously sent on another subflow. This scheme is similar to the Packet Re-striping used in [80]–[84] and Pre-blocking warning used in [120]. These three similar schemes are used in different scenarios but for the same purpose. OR is used only if the connection is receive-window limited. Packet Re-striping is employed in the case of path capacity fluctuations. Pre-blocking warning is triggered if the proactive redundancy is underestimated. The penalizing scheme of [102] is used to slow down the slow subflows. For example, if a subflow has caused too many out-of-order packets in the reordering buffer, the congestion window of that subflow is reduced by half and its slow-start threshold is set to the current congestion window. But if that subflow has been in the slow-start phase already, the reordering problem may become worse because the penalization mechanism will set its slow-start threshold to a smaller value. Paasch, Khalili, and Bonaventure [106] proposed improving the penalization mechanism by adjusting the slow-start threshold only when a subflow is not in its slow-start phase. However, they also identified that the penalization mechanism is far from perfect because a subflow at full sending speed may still overflow the receive buffer while another subflows is in slow-start.

Ferlin et al. [124] argued that the opportunistic retransmission scheme does not reduce the effect of extreme RTT heterogeneity. Instead, they proposed a Dynamic Relative Path Scoring (DRePaS) algorithm to dynamically score the paths relative to the best path and adapt the scheduling accordingly. Specifically, when the score of a path is less than a threshold, no payload is scheduled over that path until its score is larger than the threshold measured by probing traffic. Note that the standard MPTCP uses the

congestion window as an estimation of path capacity. In contrast, Ferlin et al. believed that the smoothed in-flight data on each path reflects the behavior of the connection more dynamically than the congestion window.

Chen et al. [192] explored the performance of MPTCP over wireless networks. In order to avoid performance degradation, they set the receive buffer up to 8 MB, which is not feasible in practice for many devices. Shamszaman et al. [193] analyzed the feasibility of MPTCP for big data applications. They found that constrained receive buffer leads to poor performance of MPTCP. Zhou et al. [114] proposed CWA-MPTCP that examines the goodput of MPTCP with bounded receive buffers. They found that if the paths have similar end-to-end delays, the MPTCP goodput is near optimal, otherwise the goodput will be degraded significantly. For a wireless environment, they proposed a Congestion Window Adaptation (CWA) algorithm that can adjust the congestion window dynamically for each TCP subflow so as to mitigate the variation of end-to-end path delay, maintaining similar end-to-end delays over multiple paths. The primary idea behind CWA is that a large delay ratio indicates that the high-delay path is overloaded. Its congestion window needs to be decreased to relieve traffic and reduce path delay. For wired environment with stable end-to-end delay they proposed using a delay-aware scheduling algorithm to predict the receiving sequence, i.e., a FPS manner scheduler, so that packets can arrive at the receiver in order.

Paasch, Ferlin, Alay, *et al.* [194] designed and implemented a generic modular scheduler framework that enables testing of different schedulers for MPTCP. Using this framework, they evaluated different schedulers for MPTCP and provided an in-depth performance analysis. They identified the impact of scheduling decisions on the performance of MPTCP and illustrated the underlying root cause for the observed behavior. For example, they discovered that a bad scheduling decision triggers two packet reordering effects. First, EDPF based scheduler is more efficient than simple RR in terms of avoiding the HoL problem. Second, receive-window limitation may prevent the subflows from being fully utilized.

In the last few years, several articles visited how to use delay-aware scheduling algorithms in MPTCP to improve the receive buffer utilization. Yang and Amer [123] used one-way communication delay of a TCP connection to design an MPTCP scheduler that transmits data out-of-order over

multiple paths such that their arrival is in-order. Le et al. [108] dealt with the packet reordering problem of MPTCP using a Forward-Delay-based packet scheduling algorithm. Its main idea is that the sender distributes packets via multiple paths according to their estimated forward delay and throughput difference. This scheduling algorithm is an advanced version of FPS because it took throughput difference into consideration.

The MPTCP performance is not only impacted by the receive buffer but also by the send buffer. [119] Yang et al. found that in an MPTCP connection with several high-BDP subflows, send buffer blocking can occur and seriously decrease the overall throughput. They introduced Non-Renegotiable Selective Acknowledgments (NR-SACK) too MPTCP. The idea is that once a data packet has been sacked, it can't be removed from the receive buffer. Thus, the sender can free the sacked data sooner than the advance of the MPTCP level cumulative acknowledgment. Arzani et al. [195] found that the send buffer size has significant impact on the performance of MPTCP. For example, MPTCP provides higher performance gains with a larger send buffer. However, they did not propose any method to address the problem.

Pareto-efficient MPTCP:

Khalili et al. [4], [103] demonstrated that MPTCP is not Pareto-optimal because they found that MPTCP users can be excessively aggressive toward TCP users over congested paths even without any benefit to the MPTCP users. They attributed the problem to the LIA of MPTCP. To deal with the problem, they proposed an Opportunistic Linked Increase Algorithm (OLIA) as an alternative for LIA and proved that OLIA is Pareto-optimal and satisfies the three design goals of MPTCP. Like LIA, OLIA is a window-based congestion-control algorithm that couples the increase of congestion windows and uses unmodified TCP behavior in the case of loss. The increase part of OLIA has two terms. The first term provides the Pareto optimality. The second term guarantees non-flappiness[7] as MPTCP with LIA and responsiveness (i.e., the rate of algorithm convergence). OLIA also compensates for different RTTs by adapting the window increases as a function of RTTs. However, Singh et al. [107] found that OLIA of MPTCP still has performance issues. They presented Adapted Opportunistic Linked Increases

[7]Flappiness means that MPTCP would use one path almost exclusively for a while, then flip to another path, and then repeat.

Algorithm (AOLIA) to ensure controlled aggressiveness of the MPTCP subflows. In order to minimize the packet reordering delay, they also proposed a Push-Pull-Hybrid (PSPLH) scheduler where Pull strategy is used to allocate data segments to multiple flows, and Push strategy is used to tune the size of the segments dynamically.

Path Diversity for MPTCP:

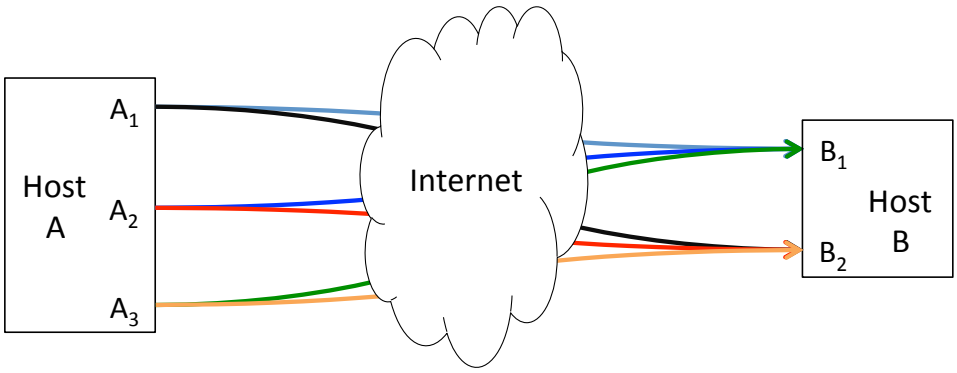

Figure 2.9: Static full-mesh of possible network paths between two MPTCP enabled hosts: A_1:B_1, A_1:B_2, A_2:B_1, A_2:B_2, A_3:B_1, A_3:B_2.

By default, MPTCP uses a *fullmesh* manager to create static full-mesh of possible network paths among the available IP addresses (see Figure 2.9). This path management may not only lead to a large number of subflows being established but also ignore the benefits the path diversity could offer. van der Pol et al. [115] combined MPTCP and OF to dynamically exploit path diversity (choose disjoint paths) between two endpoints to improve stability (in the case of partial path failure) and obtain higher throughput. Specifically, in [115] OF is used to discover the topology of the network, calculate multiple disjoint paths and configure these paths. MPTCP is used to distribute the traffic across the selected paths.

Comparison between CMT-SCTP and MPTCP

SCTP and TCP are the main base protocols widely used to support multipath transportation. Currently, CMT-SCTP and MPTCP are in the focus of the IETF and academia. In the following discussion, we make a comparison of them.

Although CMT enabled SCTP shares the same issues with MPTCP in terms of fairness, reordering, and retransmission policies, moving legacy applications from TCP to SCTP involves a number of challenges such as making SCTP work through NATs, the need to modify applications, and the lack of an easy way to negotiate SCTP versus TCP between a client and a server. None of the issues are insurmountable, but together they make adoption of SCTP as a TCP alternative a challenge. From the previous discussion, we found that MPTCP instead of CMT-SCTP has become the main stream of multipath transportation solution. In this section, we discuss the reason behind it.

By comparing the key algorithms used by CMT-SCTP and MPTCP (see Table 2.9 and Table 2.11), we found that the two protocols have shared the same or similar key algorithms. Therefore, algorithm is not the primary factor that determines which protocol could become the mainstream because the algorithms are not specific to any base protocols but could be used on any of them with minor adaptation. We believe that the reason mainly comes from their difference in terms of backward compatibility and sequence number design. For example, unlike SCTP which modifies the interfaces of legacy TCP to applications, MPTCP presents a single TCP interface to the application layer (see fig. 2.8). This seemingly minor difference makes MPTCP compatible with all legacy applications. As such, the implementation of multipath in TCP, which dominates Internet traffic, is a much more attractive deployment strategy.

The sequence space design is another primary difference. To make a fair comparison, we assume that SCTP has been widely deployed as TCP has, otherwise CMT-SCTP packets would be dropped by legacy middleboxes. As summarized in Table 2.9, CMT-SCTP keeps using one single sequence space as SCTP does. Therefore, a new SACK mechanism and its interpretation accordingly are required to avoid spurious retransmissions. Moreover, single sequence space makes bytestream discontinuous on multiple paths so that certain middleboxes may break such associations [102]. In contrast, in MPTCP the sequence numbers carried in the TCP headers are separate on each path so that the interpretation of out-of-order packets and ACKs remain the same as before. Hesmans et al. [196] and Honda et al. [197] found that MPTCP could traverse most of the middleboxes because of its double sequence space design. Furthermore, with checksums MPTCP can detect

middleboxes interference and fallback to legacy TCP. To allow clients to benefit from MPTCP in its early deployment (e.g., servers have not upgraded to support MPTCP), Detal et al. [198] proposed a protocol converter, MIMBox, to translate MPTCP to TCP and vice versa. MPTCP is not only little influenced by middleboxes but is even extended to explicitly add specified middleboxes in the middle of an ongoing communication. For example, the proposed solution in [199] used MPTCP to implement connection acrobatics (i.e., the ability to explicitly redirect connections via a middle point and the ability to migrate the endpoint of a connection). Therefore, MPTCP connections can be redirected to middleboxes located anywhere in the Internet to improve services like load balancing, DDoS filtering and anycast.

One more difference between CMT-SCTP and MPTCP lies in the path management strategy. By default, MPTCP creates a full-mesh of possible network paths among the available IP addresses, whereas CMT-SCTP only uses pairs of addresses to set up communication paths (creating only one additional path per additional source address). Becke et al. [200] found that MPTCP's path management strategy performs significantly better than that of CMT-SCTP in the case of asymmetric paths.

2.2.4 Application Layer Multipath Capability

Table 2.12: Key algorithms for application layer multipath capability (sorted according to their order mentioned in Table 2.13).

Algorithm	Problems to address	Description
HTTP Range Retrieval Request (HTTP-RRR)	Request segments over multiple paths	It commonly makes a halted download to proceed with the outstanding portion at a later time. In multipath transmission, it could be used to download unique segments of a file at the server. Note that each request must be sequentially handled before the next request could be sent out.
HTTP Request Pipelining (HTTP-RP)	Concurrent HTTP-RRR	It generates multiple requests from the client without waiting for each response from the server.

Provisioning multipath capability at the application layer has received a lot of attention because the approaches are almost independent of the underlying access technologies and network-layer routing. It is a common practice that an application establishes multiple transport connections, binds them to different IP addresses, and distributes the data in proportion to the available path capacity over these connections (see Figure 2.10). To reassemble the data delivered over different connections, each packet or a group of pack-

Table 2.13: Concurrent Multipath Transfer Applications.

Scheme	Year	Algorithm and Protocol	Path(s)	Network Environment
XFTP [126]	1996	WRR	Same	Satellite channel
PSockets [127]	2000	Middleware, RR	Same	General
GridFTP [128]	2001	Fair share (RR)	Same	Bottleneck link
PA [129]	2002	WRR	Different	Many senders and one receiver
ATLB [130], [131]	2005, 2007	Middleware, WRR, FPS	Different	General
Tavarua [132]	2006	Middleware, WRR	Different	Cellular uplink channel
SBAM [133]	2006	Middleware, WRR	Different	Wireless access
DMP [134], [135]	2007, 2009	WRR-PULL	Different	General
MultiTCP [136]	2008	Receiver-driven rate control	Different	Insufficient bandwidth due to traffic bursts
PATTHEL [137]	2009	Middleware, WRR	General	General
Kaspar et al. [138]	2010	WRR, HTTP-RRR	Different	Wireless
Kaspar et al. [139]	2010	WRR, HTTP-RRR, HTTP-RP	Different	Wireless
Evensen et al. [140]	2010	HTTP-RRR, HTTP-RP, Request scheduler, WRR	Different	Wireless
Evensen et al. [141], [142]	2011, 2012	HTTP-RRR and HTTP-RP, Improved Request scheduler, WRR	Different	Wireless
Miyazaki et al. [143]	2012	Middleware, EDPF	Different	Wireless
DBAS [144], [145]	2012, 2013	Middleware, WRR, PFA	Different	Wireless
G-DBAS [146]	2012	Middleware, Energy-awareness Scheduling, WRR, PFA	Different	Wireless
OPERETTA [147]	2012	Middleware, Energy-awareness Scheduling, WRR	Different	Wireless
MPTS-AR [19], [148]	2014, 2015	OpenPath, MPTP	Different	General

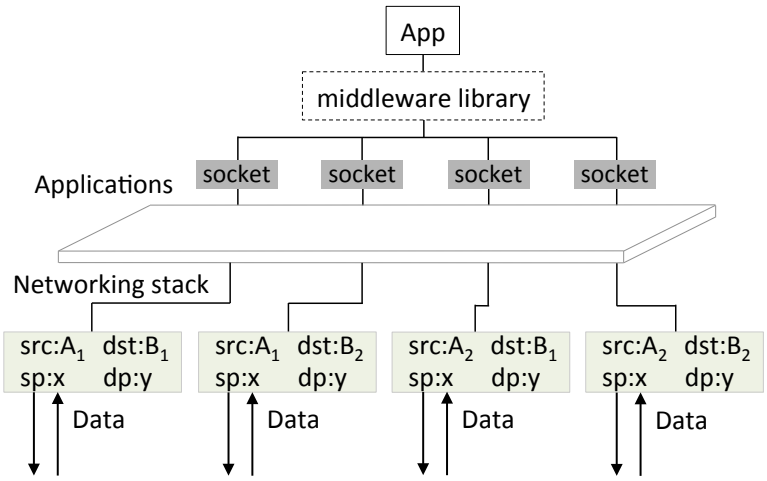

Figure 2.10: Multipath transmission at the application layer. It is assumed that each host has two interfaces. src: Source IP address, dst: Destination IP address, sp: Source Port, dp: Destination Port, APP: Application.

ets are usually assigned additional sequence numbers.

In the rest of this section, we also divide the approaches into four groups. The first group discusses approaches using the same path. The second group discusses approaches using different paths. The third group investigates approaches based on HTTP in order to highlight the importance of the combination of HTTP and multipath transmission. The fourth group exploits middleware approaches. Table 2.12 and Table 2.13 are used to summarize the key algorithms and approaches respectively.

Multiple Connections over the Same Path

In the early stage of research work on application layer multipath transmission, the focus was on bandwidth aggregation using multiple TCP connections over the same physical path. For instance, Allman et al. [126] developed a new application called XFTP, a modified version of FTP [201], to improve the poor performance of TCP over long-fat satellite channels. XFTP creates multiple TCP connections. To send a file, XFTP divides the file into records, reads the file from local storage one record at a time, and sends each record over whichever connection has resource available for transmission. To reassemble the records into the correct order, XFTP uses an additional 4-byte sequence number to each record. GridFTP [128] is another extension of the FTP protocol implemented for bulk data transfer, where parallel TCP connections are created to increase the throughput in a bottleneck link. Specifically, GridFTP divides the data to be transferred into multiple portions and transfers each portion with a separate TCP connection. When competing with non-GridFTP connections over a bottleneck link, the GridFTP connections will be less likely to be selected to drop their packets. Hacker et al. [202] examined the effects of using parallel TCP flows to improve end-to-end network performance. They found that in the absence of congestion, the use of parallel TCP connections is equivalent to using a large Maximum Segment Size (MSS) on a single connection. In addition, they addressed the question of how to select the maximum number of connections to maximize the overall throughput while avoiding congestion. For example, if the selected value is too large, the aggregate flow may cause network congestion and throughput will not be optimized.

Multiple Connections over Different Paths

The approaches mentioned above aim to increase application throughput by using multiple TCP connections through the same physical path. Nevertheless, if they are used for striping data over different physical paths, the reordering issue at the receiver would render them inefficient because they do not take into consideration the reordering issue caused by heterogeneous paths. In the 2000s, researchers started to seek solutions to provide bandwidth aggregation over different physical paths. A simple approach to achieve this goal is to directly add support for multiple interfaces to a given application by opening multiple TCP sockets (one for each active interface), and performing striping of data across different sockets. If the interfaces of a client are connected to independent networks, the simultaneous use of multiple paths can achieve a total throughput close to the sum of all the throughput from individual interfaces.

Given that popular files are often replicated on multiple servers, it becomes natural for clients to connect in parallel to several mirror servers to retrieve a file (i.e., many-to-one fashion). Golubchik et al. in [203] investigated the potential benefits of an application layer multipath streaming approach between a set of senders and a receiver. They found that multipath streaming exhibits better loss characteristics than single path streaming. Rodriguez and Biersack in [129] described a parallel-access (PA) scheme that allows users to fetch different portions of a file from multiple servers at the same time and reassemble the file locally. The PA scheme allows dynamic load sharing among all servers so that faster servers will deliver bigger portions of a file while slower ones will deliver smaller portions.

Shiwen et al. proposed Multiflow Real-time Transport Protocol (MRTP) [204] which is a multipath transmission extension to Realtime Transport Protocol (RTP) [205] for real-time applications. MRTP supports multimedia services by exploring multipath transport in mobile ad hoc networks, where link bandwidth may fluctuate and paths are unreliable. The authors studied the impact of traffic partitioning on congestion at bottleneck links and found that the bandwidth utilization of a bottleneck node could be much improved by two strategies (thinning and striping [206]). Furthermore, they showed analytically that most of the performance improvement can be achieved with a few paths (e.g., two or three paths), while only marginal improvement is gained by further increase in the number of paths.

Wang et al. [134], [135] proposed Dynamic MPath-Streaming (DMP), a scheme for live streaming over multiple TCP connections. DMP allocates packets over multiple paths according to their current throughput. DMP does not use an explicit probing scheme for bandwidth estimation on each path. Instead, it uses the WRR-PULL scheduler to allow each connection to pull data from a shared queue whenever it has opportunity to send data. Thus, the paths with higher throughput deliver more packets than others.

Tullimas et al. [136] proposed MultiTCP for multimedia streaming. It aims at providing resilience against short term insufficient bandwidth due to traffic bursts by using multiple TCP connections for the same application. MultiTCP is a "smart" application that allows the application to control the desired sending rate during congestion periods. MultiTCP achieves rate control by the means of adjusting the receiver window.

Zhang et al. [148] proposed a general framework of multipath transport system based on application-level relay (MPTS-AR), currently under the standardization within the IETF [19]. This framework defines three logical entities and two protocols. The entities include user agent, relay server, and relay controller. The protocols are OpenPath and MPTP (Multipath Transport Protocol), which are used in control plane to manage relay paths and in data plane to facilitate multipath data transport respectively. However, they left a few key functions we concern the most out of the scope. For example, how to split the original data stream into several substreams, how to mitigate the reordering issue at the receiving side, how to provide path diversity among all available paths, and how to obtain the performance of overlay paths are all out of the scope.

HTTP based Multipath Media Streaming

In addition to multipath approaches for specific applications, HTTP [207] with multipath capability is currently one of the most common protocols for streaming video on the Internet through multiple paths. Kaspar et al. [138], [139] and Evensen et al. [140]–[142] presented HTTP-based methods for downloading multimedia content simultaneously over multiple network interfaces.

Kaspar et al. [138] proposed an HTTP-based on-demand streaming service over multiple wireless access networks. They presented a proof-of-concept implementation of a progressive download service, which uses HTTP-

RRR capability [207] to download specific segments of a file from a media server. The drawback of the range retrieval request is that each request must be handled sequentially by the server before the client can send the next request. Thus, an average time overhead of one RTT is introduced for each request. In order to avoid waiting for each response, in [139] they presented an improved version of their work by using an additional HTTP capability, i.e., HTTP-RP [207]. The request pipelining function of HTTP allows a client to make multiple requests simultaneously. In [138], they studied the effect of file segmentation on the buffer requirements and found that there exists an optimal segment size for which the aggregation efficiency is maximized. In [139], they found that due to the use of request pipelining, very small segments can provide efficient throughput aggregation.

Evensen et al. [140] introduced an adaptive, WRR-PULL based scheduler that achieves smooth playback by scheduling requests for video segments of different quality levels over multiple interfaces simultaneously. Like their previous work in [138], they still utilized HTTP-RRR and HTTP-RP functions to support multipath transmission. In order to avoid video deadline misses, they proposed an additional request scheduler in [140]. The scheduler is mainly used for estimation of the aggregated throughput for chosen video quality level and request of segments over the available interfaces. However, the weakness of the request scheduler is that the segments are divided into fixed-sized subsegments, which may lead to low performance with constrained receive buffer. Evensen et al. [141], [142] proposed improving the request scheduler by loosing the segment size constraint. For example, the segment sizes are dynamically calculated on the fly based on the capability of each path.

Session layer Multipath Capability

Unlike application layer approaches discussed previously, some approaches open multiple TCP flows without any change to existing applications by providing specialized middleware or virtual sockets at the session layer between the application and transport layer.

Sivakumar et al. proposed PSockets (Parallel Sockets) in [127]. PSockets is a library that transparently partitions upper layer data into multiple transport streams through the same physical path. The principal idea is to split data equally across several open sockets without application upgrade.

PSockets has the same Application Programming Interfaces (APIs) as those of a regular socket. Note that this work was published in the year 2000. Back then, the TCP window size had to be tuned manually for high-speed networks at both the source and the destination in order to achieve the maximum throughput. Differently, PSockets allowed applications to achieve the best performance without tuning the window size.

Yohei et al. [130], [131] proposed an Arrival-Time matching Load-Balancing (ATLB) layer between the application layer and TCP layer. ATLB consists of a distributed data transfer method and a path-failure detection/recovery method. In order to mitigate the reordering effect at the receiver, ATLB calculates the data arrival time for each path. It considers the time that data segments spend in the TCP queue at a sender and the time needed for data segments to pass through the network.

Qureshi et al. presented a prototype system Tavarua in [132]. Tavarua is a middleware for providing network striping capability to applications with high demands on uplink throughput. Note that Tavarua runs upon UDP. In an effort to mitigate the impact of variations in bandwidth, an application can use feedback information to estimate the maximum available data transmission rate. Then the bit-rate at which the video is encoded is adapted dynamically. The middleware also handles low-level issues related to the network interfaces (e.g., congestion control, disconnections, and reconnections).

Sakakibara et al. [133] proposed a Socket-level Bandwidth Aggregation Mechanism (SBAM) to offer aggregated bandwidth. SBAM is located at the socket layer (close to TCP) so that it can collect system resources efficiently. For example, it has a network monitoring function to collect delay and available bandwidth of each path. Using this information, the traffic scheduler decides the amount of data to fill the bandwidth-delay product of each path.

Like the other middleware approaches, Parallel TCP Transfers Helper (PATTHEL) [137] also provides APIs for applications. The difference of PATTHEL lies in two facts. First, PATTHEL incorporates a separate data connection and control connection, where the control connection is created first to manage the other data connections for the entire communication period. Second, PATTHEL is a cross-layer protocol because it adds an entrance to the routing table in order to deliver data over a certain channel.

Miyazaki et al. [143] examined how much receive buffer is needed in various scenarios and found that the buffer size is proportional to the ratio of the bandwidth of the two interfaces. A larger bandwidth difference leads to a bigger receive buffer and vice versa. The scheduling algorithm used in [143] is EDPF.

Habak et al. [144], [145] proposed a Deplorable Bandwidth Aggregation System (DBAS) middleware architecture for multi-interface enabled devices. Like the work in [51], [58], [79], DBAS also supports both FOSM and POSM. In FOSM where the server is not DBAS enabled, DBAS schedules different connections to the interfaces such that a connection can be assigned to only one of the available interfaces. If both sides are DBAS enabled, POSM is used so that each packet can be scheduled independently on a different interface. To make better scheduling decisions, DBAS estimates the characteristics of the applications dynamically based on their behavior and stores them in a database for history track. DBAS focuses on the actual implementation of the basic core system. The authors presented an extended work based on DBAS, a Green DBAS (G-DBAS) [146] to balance overall throughput with energy consumption. For example, they introduced a new utility based scheduler that takes energy consumption of each interface into account in order to balance the trade-off between maximizing throughput and minimizing power consumption. Note that G-DBAS only works in FOSM. OPERETTA [147] is an extension of G-DBAS to support POSM.

Application layer approaches split a single file or byte stream into segments that are transmitted concurrently over different paths. These kind of approaches seem to be simple in the sense that the applications are in full control of the striping decisions. Thus, it does not require any protocol change at lower layers so that clients and servers can find an optimal way to collaborate. Nevertheless, the complexity and overhead at the application layer are considerable. For example, an application-level sequence number has to be included in each of the application defined headers. Meanwhile, the application has to explicitly ensure that the application layer data units, which carry unique application-level sequence numbers, do not get fragmented during transmission. Moreover, a dedicated resequencing mechanism is required to reassemble the data at the receiver. In practice, different paths may have diverse characteristics, and the striping ratio may not exactly match the ratio of data rate from different paths. A large receive buffer

(on the application level) is required to accommodate the out-of-order data. Finally, in order to split intelligently, the application has to implement a bandwidth estimation mechanism redundantly despite the same mechanism has been employed by TCP through its congestion control mechanism.

The middleware approaches are very similar to application-layer approaches. They also face the same challenges, for example, the reordering issue. The advantage of middleware approaches is that although it still requires client and server-side modifications, applications usually are not required to be upgraded.

2.2.5 Summary

In this section, we summarize the issues that are common to the approaches from all layers we have covered in this survey. Specifically, we first discuss the packet reordering problem and how effective the widely used scheduling algorithms are to mitigate it. After that, we present the approaches which have adopted the cross-layer design. Next, we compare the approaches' compatibility capability, which is inherently determined by their stack positions. At last, we summarize the research problem evolution on each layer.

Packet Reordering

When packets travel through different paths which may have mismatched characteristics, they may arrive at the destination out of order. All the presented approaches deal, to some extent, with the reordering issue on the layer which they are located at. If they ignore or have no control over the reordering mechanism on the transport layer, their approach may suffer from performance degradation because of the misinterpretation of out-of-order packets. We group and sort them according to their effectiveness in terms of packet reordering and load-sharing capability.

The first group includes PCA [13], [41], [42], [156], PFA [18], [20], [40], [43], [51], [54], [58], [79], [144]–[146], and Multi-streaming [66]. They are the most effective mechanisms which can completely eliminate packet reordering incurred by multipath transmission because the data units required sequencing at the destination are assigned only to the same path. However, a multipath transmission protocol with them usually performs worse than without them in terms of load sharing. For example, if the number of flows

is less than that of available paths, there would exist paths which cannot be fully utilized.

FPS [77], [108], [110]–[112], [114], [120], [123], [130], [131] breaks the in-order scheduling rule at the source. Whenever a path has opportunity to send a new packet, it estimates that path's capacity and chooses a new data block accordingly so that out-of-order sent out packets could arrive at the receiver in-order.

In EDPF [60], [83], [84], [91], [92], [109], [143] and PET [46], the scheduler first sends data on a path with lowest RTT until it has filled its congestion window. Then, the data is sent on the subflow with the next lowest RTT. The larger the RTT difference is, the more out-of-order packets arrive at the destination. Moreover, EDPF and PET consider only path bandwidth and latency, ignoring packet loss rate caused by congestion. Therefore, it may achieve sub-optimal load-balancing in the presence of high losses or heavy congestion.

WRR [4], [17], [36], [37], [44], [45], [51], [53], [58], [59], [71]–[75], [78], [80]–[82], [86], [88]–[90], [93], [96], [99]–[101], [103], [126], [129]–[135], [137]–[142], [144]–[147] is the most widely used scheduling algorithm on all layers with maximizing the overall throughput as its first priority. It could achieve the goal if the multiple paths have similar characteristics. Otherwise, it would cause significant out-of-order packets at the receiver because it considers little about its impact on packet reordering. Therefore, WRR works better on link layer than other layers because link layer has relatively stable link state. Like FPS, WRR could fully utilize the available path capacity.

In practical network environments where path characteristics may change dynamically, the scheduling algorithms except PFA, PCA and Multi-streaming may fail to counter against packet reordering. Several additional mechanisms have been proposed to work coherently with the scheduling algorithms, such as ACK manipulation [48], [52], [58], [61], [79], [88], [90], [105], buffering management [47], [50], [58], [65], [68], packet coding [91], [105], [110], [111], [120], blocking warning and fast retrieving [80]–[84], [102], [120], slow-path penalization [102], [106], [124] and so on. The comparison of various combinations of the scheduling algorithms is out of the scope of our knowledge because they may be used in various network environments, triggered by different conditions, and supported by diverse assumptions.

To make a conclude on the discussion of packet reordering, the choice

of the best scheduling policy depends on path characteristics. There is no single scheduling mechanism which can handle all scenarios. To adapt to different network environments, several approaches, such as [51], [58], [79], [144], [145], could support both flow level scheduling and packet level scheduling strategies.

Layer-dependent scheduling algorithms

In our previous discussion, we could find that many scheduling algorithms are shared by different approaches on various layers. For example, FPS variants are used on transport (including [77], [108], [110]–[112], [114], [120], [123]) and application layers (including [130], [131]). EDPF variants are used on network layer (including [46], [48]–[50], [55]), transport layer (including [60], [83], [84], [91], [92], [109]) and application layer (including [143]). WRR variants are used from link to application layers. However, we argue that it does not imply that these scheduling algorithms are layer-independent. Instead, they are mostly layer-dependent.

Running a scheduling algorithm usually requires measuring bandwidth, delay, loss, or jitter to provide best effort or even QoS guarantees. No single measurement on a certain layer could always give accurate measurements. The measurements may even vary on different layers. Although how to measure those metrics is an entire problem unto its own, the efficiency of the algorithm is closely connected to the correctness of the measurements. Therefore, we argue that the scheduling algorithms which rely on the estimation of certain path characteristics are layer-dependent.

We discuss WRR as an example to support our argument. The same principle is applicable to other algorithms which rely on the estimation of path characteristics. WRR is the most widely used scheduling algorithm. Although each layer adopts many variants of the WRR algorithm, its effectiveness highly depends on how correctly the path capability is estimated in a dynamic fashion. For example, using the congestion window size on the transport layer to estimate the path capability is a more lightweight and accurate way than those using various probing methods on other layers. Furthermore, WRR works much better on link layer than upper layers because link layer has more stable link status.

Table 2.14: Cross-layer Support for Multipath Transmission.

Scheme	**Year**	**Base Layer**	**Additional Layers**	**Description**
PRISM [48], [49]	2005, 2007	Network	Transport	1) Use the transport-layer information carried in ACKs to mask adverse effects of out-of-order packet deliveries. 2) tune the TCP congestion control mechanism to handle, at a sender side, packet losses.
ATLB [130], [131]	2005, 2007	Application	Transport	Use the TCP connection's throughput history to calculate the queuing delay in the sending buffer and use TCP's smoothed RTT to calculate the network path delay.
Tavarua [132]	2006	Application	Transport, Link	Handle low-level issues related to congestion control and interface disconnection/reconnection.
SBAM [133]	2006	Application	Network, link	1) Send ICMP packets to periodically measure path delays. 2) read routing functionality to route packets through different network interfaces. 3) monitor link status information on both sides, for example, available network interfaces as well as their up/down status.
MultiTCP [136]	2008	Application	Transport	Dynamically change the receiver's TCP window size to control the throughput of each TCP connection.
PATTHEL [137]	2009	Application	Network	Add entrances to the routing table in order to deliver data over a certain path.
G-DBAS [146]	2012	Application	Link	Estimate the characteristics of each network interface as well as the energy consumption of each interface.
OPERETTA [147]	2012	Application	Link	Estimate the available bandwidth and the energy consumption of each interface.
DBAS [144], [145]	2012, 2013	Application	Link	Estimate the characteristics of each network interface, for example, the available bandwidth and packet error rate.
OpenFlow-MPTCP [115]	2013	Transport	Link	Use OpenFlow to discover the topology of the network, calculate multiple paths and configure these paths on the OpenFlow network. MPTCP distributes the load across the selected paths.
A-MPTCP [118]	2013	Transport	Network	Use LISP [17] to give better knowledge of the underlying IP topology to MPTCP enabled endpoints in cloud networks.
MPTCP-MA [121]	2014	Transport	Link	Use MAC information to estimate the path status so as to suspend or release a path based on the estimation.

Cross-layer Support

In this survey, we define that any attempt to violate the TCP/IP reference model is considered a cross-layer design. Among the approaches we have discussed previously, several of them have explicitly involved cross-layer interaction for purposes of estimating path status to avoid packet delays and losses, scheduling traffic over multiple paths according to their capacity, exploring path diversity to obtain high throughput, achieving better QoS for multimedia applications, and so on. Due to these benefits the cross-layer design may offer, there has been increased interest in protocols with interactions between various layers of the network stack. In the rest of this section, we discuss the cross-layer approaches based on our previous discussion and give a few observations on them without our own judgment. Instead, we refer readers to Kawadia and Kumar's work [208] which calls for a cautionary approach to cross-layer design. Although [208] examined the issue of cross-layer design in wireless networks, we believe the same principle is also applicable for multipath transmission.

Table 2.14 shows approaches based on a cross-layer design. The "Base Layer" indicates the layer where the data splitting is initiated, and the "Additional Layers" indicate which layers are required to provide support to the base layer. From the table, we observe that the transport and application layers are the main base layers. From 2005 to 2013, it was drawn most of the attention to implement the base layer on the application level. Some applications obtain low layer information to optimize their behavior in terms of interface selection, load balancing, and energy efficiency. The information includes throughput history and smoothed RTT from transport level, routing table from IP level, and link status (e.g., energy consumption, available bandwidth, and bit error rate). Some applications can even change the low layer protocol behaviors such as changing TCP window size to control the throughput, modifying the routing table to optimize path selection, and disconnecting/reconnecting certain interfaces for energy efficiency or partial failure. In most recent years, it has become more attractive to use transport layer as the base layer with additional support from network and link layers. Although transport layer approaches have advantages from the congestion control mechanism, they lack the choice of path diversity to free the constraint of fairness control. The path optimization support from network and link layers can compensate for the weakness in network and Ethernet levels

respectively. In addition, some transport layers can suspend or release a path based on the estimated MAC information.

We can also find that the interaction of cross-layer design may not be limited to adjacent protocol layers. Instead, it allows vertical communication to take place between nonadjacent layers. The cross-layer design approaches are actually not limited to the ones listed in Table 2.14 because although many approaches above the network layer did not mention or specify how packets are delivered through multiple interfaces, the cross-layer support from routing function on network layer may be assumed implicitly.

Compatibility

Table 2.15: Compatibility Evaluation (√ means being supported).

Compatibility	Link	Network (no proxy)	Network (1 proxy)	Network (2 proxies)	mptcp	CMT-SCTP	Session	Application
Application	√	√	√	√	√		√	
Backward		√	√	√	√	√	√	√
Middlebox	√				√		√	√
Host				√				
Infrastructure	√	√			√	√	√	√

In this section, we evaluate the compatibility capacity of the approaches on different layers. The evaluation is made in general because there may be exceptions in certain approaches. Table 2.15 presents the evaluation where we separate network layer approaches into three categories according to how many proxies required in each category. We use MPTCP and CMT-SCTP to represent the transport layer approaches and evaluate their compatibility separately. Likewise, we also separate session layer approaches from application layer approaches.

Application compatibility means that the lower layer changes do not require the legacy applications to be upgraded. Obviously, the link layer and all the network layer approaches are compatible with the legacy applications because they do not change the socket interface between the legacy applications and transport layer protocols. MPTCP presents a standard TCP socket API to the application so that legacy applications can run upon MPTCP transparently. However, the legacy applications running on TCP have to upgrade in order to take advantage of CMT-SCTP. Session layer approaches usually keep the same socket API to applications (e.g, PSockets [127]) so that they require no changes to the legacy applications. Applica-

tion layer approaches have a serious application compatibility issue because the multipath transmission property needs to be implemented for specific applications.

An updated approach has backward compatibility if it can either work with its communicating peer which uses the standard approach or automatically fallback to the standard approach if the communicating peer does not support the new features. Link layer approaches generally do not maintain the backward compatibility because most of them are proprietary approaches and require dedicated setup on both sides. Therefore, we mark that the link layer approaches are not backwards compatible. Network layer approaches with no proxy usually employ some of the fields in protocol headers (e.g., [44], [45], [52], [54], [55], [58]) to negotiate multiple IP addresses to be used or piggy-back information so as to provide backward compatibility. Network layer approaches with one proxy has backward compatibility because the communicating peer is unaware of the multipath transmission between the client and the proxy. The network layer approaches requiring two proxies are also backwards compatible because both sides are unaware of the multipath transmission between the two proxies. MPTCP is backwards compatible with plain TCP because it can fallback to single-path TCP if the communicating host does not support the extensions. CMT-SCTP related articles did mention at all whether it can fallback to SCTP if the server is not CMT-SCTP enabled. We believe CMT-SCTP could also fallback to SCTP if the server does not support concurrent multipath transfer. But anyway, CMT-SCTP has inherited the backward compatibility issue of SCTP itself. For example, if the server is only aware of TCP operations, a CMP-SCTP client may fail to create connection with the server. Session layer and application layer approaches are generally backwards compatible with the legacy applications. When connecting to a server, a client application usually specifies two different TCP ports, a probe one and a fallback one. First, the client tries to establish a connection using the probe port to check whether the server is upgraded. If the operation fails, another attempt is made by using the fallback port to create a standard connection.

The compatibility with existing middleboxes, such as firewalls and NATs, affects whether the “new” packets are able to traverse the legacy middleboxes. First of all, link layer approaches generally would not have this issue because the communication paths or trees are well designed between ded-

icated multipath-aware devices. Most network layer approaches (with and without proxies) split bytestream over multiple paths. Therefore, the single sequence space across more than one path leaves gaps in the sequence space seen on any individual path, which may upset certain middleboxes. To solve this issue, the double sequence space design was proved to be an effective solution [102], [197]. However, how to use the double space also influenced the outcome. For example, HIP based multipath transmission adopts double sequence space. But the additional sequence space is used for the purpose of anti-replay instead of resequencing. In contrast, MPTCP packets, which carry double sequence numbers for resequencing on two levels, can traverse most of the middleboxes. CMT-SCTP packets may fail to traverse through certain middleboxes due to its single sequence space design. Session and application layer approaches create multiple standalone TCP flows so that their TCP flows can travel through various middleboxes as normal TCP does.

Host compatibility means whether the approaches require changes in hosts. We found that all approaches need changes on hosts except the network layer approaches with two proxies because the multipath transmission between the two proxies are unknown to both communicating peers.

Infrastructure compatibility means whether an approach needs additional network infrastructure such as NAT box and proxy. We found that only the network layer approaches with the proxy support require additional infrastructure.

According to the previous discussion, it is hard to implement a generic multipath solution which can satisfy all the compatibility requirements. As summarized in Table 2.15, we have a few more observations. For example, MPTCP and session layer approaches have more compatibility support than others. CMT-SCTP has inherited SCTP's application compatibility issue, which becomes the major obstacle of its real deployment. Network layer approaches lack discussion on backward compatibility.

Evolution of Research Problems

Table 2.16 presents the research problems the approaches on each layer have tried to address. In the early stage of multipath transmission research, most approaches emphasized only bandwidth aggregation with various scheduling and packet reordering algorithms. Few of them considered the fairness

Table 2.16: Research Problems on Each Layer (√ means having tried to address).

Research Problems	Link	Network	Transport	Application
Bandwidth aggregation	√	√	√	√
Packet reordering	√	√	√	√
Fairness		√	√	
RP			√	
Path diversity	√	√		
Pareto-optimality			√	

and RP features. Today, these two problems as well as Pareto-optimality problem have become challenges along with the revolutionary development of multipath transmission.

The fairness requirement on multipath transmission was unclear in the beginning. For example, the early research on multipath transmission focused on bandwidth aggregation by taking advantage of the resources through multiple interfaces. The target in the research community matched the potential expectation of end users because an end user can benefit from the aggregated bandwidth if they have paid for both accesses. Thus, the fairness emphasized the fairness of each individual subflow; for example, each subflow gets as much bandwidth as a standalone TCP flow does. In recent years, the research focus was on the fairness of the multiple subflows as a whole at shared bottlenecks. The principle is that a multipath transmission should behave as a single TCP flow at shared bottlenecks. Coordinated congestion control algorithms are used as a powerful tool to achieve it.

The concept of RP [155] was proposed in 2008. The early approaches before it also proposed using less congested paths more, which is one of the RP principles. From the congestion control algorithm viewpoint, the difference between these approaches and the approaches after 2008 is that the latter ones use coordinated congestion control algorithms instead of independently tuning each path's congestion control behavior.

The Pareto-optimality is a state of resource allocation in which there is no alternative state that would make some people better off without making anyone worse off. MPTCP with LIA [96]–[98], [100] fails to satisfy the Pareto-optimality because upgrading some regular TCP users to MPTCP can reduce the throughput of other users without any benefit to the upgraded users. OLIA [4], [103] as an alternative for LIA could make MPTCP Pareto-

optimal and satisfy the three design goals of MPTCP.

In Table 2.16, we observe that the multipath transmission follows an evolutionary way mostly on the transport layer. We believe this is determined by the stack position at which the proposed approaches are located. For example, the fairness, RP, and Pareto-optimality features are achieved by the means of congestion control, which as default is managed on the transport layer. The link layer and application layer cannot intervene congestion control without breaking the protocol stack layered structure. In our literature review, mHIP [56] is the only protocol which provides fairness feature on network layer. However, mHIP is actually located on a middle layer between network and transport layers. Therefore, because of its closeness to transport layer, mHIP is able to manipulate the congestion control operations. The link layer and network layer can provide path diversity for cross-layer approaches. For example, OpenFlow and TRILL can provide Ethernet level path diversity [115], and LISP can provide routing level path diversity [118].

2.3 Summary

From the evolution of end-to-end multipath transmission viewpoint, we observe that multipath has become increasingly popular at the transport layer with features such as load balancing, fairness control, congestion control and Pareto-optimality. As a major extension to TCP which has not changed very much in the last decades, MPTCP has attracted more and more attention in recent years. Conventional TCP/IP always uses a single "best"path according to certain routing metrics, even if there may be more than one path between two endpoints. This behavior results in under-utilization of the available network resources. The proliferation of mobile devices equipped with multiple interfaces, represented by smart phones, brings with it a growing number of multi-homed hosts onto the Internet. Thus, this deteriorates the mismatch between single-path transport and the multitude of available network paths. Multipath transmission comes into the picture as a natural solution with several salient features, such as reliability, fairness, confidentiality and RP. As shown in Figure 2.2, MPTCP was proposed at an opportune time to draw the experience gathered in previous work and correcting the past mistakes. For example, MPTCP was designed with the

backward compatibility to legacy applications and middleboxes (see Table 2.15), which makes MPTCP instead of CMT-SCTP a big step towards being widely acceptable. In addition to solving the compatibility issue, MPTCP goes further in addressing issues in fairness and RP by joint increase and decrease rules (e.g., the coupled congestion control algorithm LIA [96]). Apart from the technical aspects, we believe the following factors also contribute to the success of MPTCP: kernel implementation in Linux, support from Internet standards organization (e.g., IETF), active and public academic community[8], and incentives from industry [27], [29], [30].

Despite these past progress, a few problems have yet to be overcame to allow for a wider deployment of CMT. First of all, some mechanisms need to be devised to better communicate and enforce the path diversity: there are many different network environments for multipath transmission. For example, in modern data centers, there are usually more than one path available between any pair of endpoints. The same applies to access networks, the core of the Internet, and ISPs. Nevertheless, these domains are usually autonomous and isolated from each other. There are specialized network entities, e.g., border gateway, to guarantee the autonomy of each domain. The downside of this network topology is that even though there might be abundant multipath resources available locally within each domain, the globally available multi-path resources are limited to the border gateways. Thus, how to break the border of each autonomous domain to enrich the multipath resources significantly is not only a technical problem, but also a management problem. It requires efforts from multiple domains, including ISPs, policymakers and end users.

As throughput aggregation is one of the main driver for MPTCP deployment, the respect of Pareto-optimality might be the most critical aspect of CMT. Even though work has been done on this topic (OLIA [103] and AO-LIA [107]), in practice, it is not guaranteed yet as MPTCP sometimes still perform worse than TCP ([5], [209]).

Currently, most work has been using one single scheduling algorithm all the time, which would definitely lead to low performance in certain network conditions. Therefore, context-aware scheduling policy is required to dynamically detect the network context and switch to the best performing

[8]MPTCP has an active mail list (mptcp-dev@listes.uclouvain.be) for sharing experience.

scheduling algorithm accordingly. The concept of context-aware scheduling is not new. Some initial work has been conducted. For example, WiMP-SCTP [74] has two data transmission scheduling algorithms used for different network conditions. When the network condition is good, the data-striping algorithm is selected to aggregate bandwidth. When the network condition becomes bad, the data-duplicating algorithm is switched on to increase destination reachability. In [51], [58], there are two scheduling algorithms which are used for legacy destination and updated destination respectively. Similar work can be found in [79], [144], [145]. However, from a user's perspective, boosting the throughput of a multi-homed mobile device may not be the first-priority goal all the time. Instead, some users may be willing to use cheap subscriptions in order to save a few dollars. Others would rather use the low energy-consumption interface(s) to save the battery life or on the contrary maximize the number of used paths to increase the security. Therefore, multipath transmission strategies which take user policies (price, energy...) into consideration should be provided to users so that they can choose from different transmission strategies to satisfy their demands.

Chapter 3

Presentation of MPTCP

MPTCP is a TCP extension enabling end-to-end multipath communications, with an emphasis on backward compatibility. As it is our main protocol of interest during this thesis, diving into more details than in the previous chapter can be of interest. We give first a brief overview of MPTCP in Section 3.1, then we present the connection initiation in Section 3.2, the transmission of the data is explained in Section 3.3 and some of the current challenges in Section 3.5.

3.1 High level design of MPTCP

MPTCP is a TCP extension formalized in RFC 6824 [183]; the MPTCP working group at the IETF was formed in October 2009; since the beginning it emphasizes backwards compatibility with the network and applications. This is an aspect to keep in mind when looking at some design decisions that may seem as counter intuitive at first. As a result, TCP applications can run unmodified with MPTCP.

An Application Programming Interface (API) is proposed to give the applications more control and finer tuning over MPTCP behaviors but it is not mandatory: the system falls back on default heuristics instead for its operations.

MPTCP signaling relies on the use of one TCP option in the three-way handshake, and of another TCP option during the connection to open and close subflows as a function of connection, subflow and network states, as depicted in fig. 3.1.

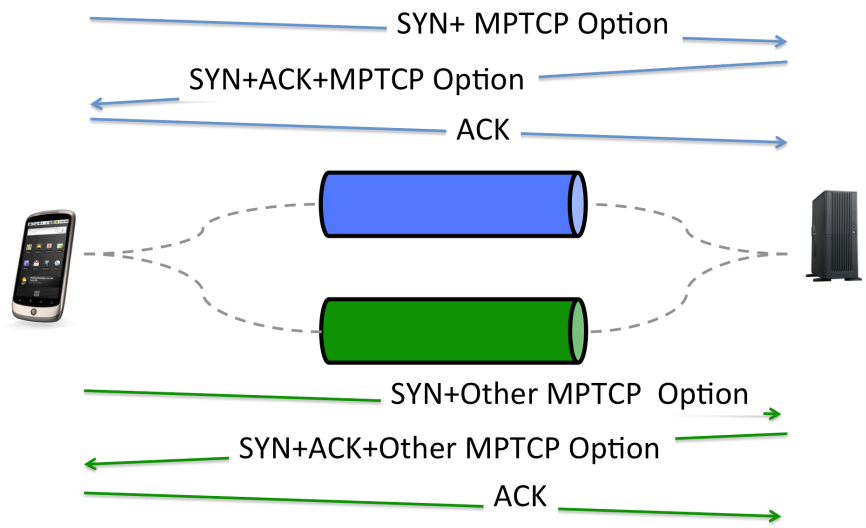

Figure 3.1: Simplified representation of the MPTCP handshake (Source: [210]).

Listing 3.1: The DSS option format.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------+----------------------+
|     Kind      |    Length     |Subtype| (reserved) |F|m|M|a|A|
+---------------+---------------+-------+----------------------+
|           Data ACK (4 or 8 octets, depending on flags)        |
+--------------------------------------------------------------+
|   Data sequence number (4 or 8 octets, depending on flags)   |
+--------------------------------------------------------------+
|              Subflow Sequence Number (4 octets)              |
+-------------------------------+------------------------------+
|  Data-Level Length (2 octets) |      Checksum (2 octets)     |
+-------------------------------+------------------------------+
```

As explained in [210] nowadays, many middleboxes (e.g., TCP Optimizers, firewalls, Port Address Translation (PAT) NAT) to modify the TCP header (e.g., change sequence number) or drop the packets (e.g., if it detects unknown protocols), and thus can hinder TCP extensions deployment. A list of known TCP fields tampered with by middleboxes can be seen in Figure 3.2.

If any problem of the kind is detected by MPTCP, it falls back to legacy TCP, as it is the case if the remote endhost is not MPTCP compliant. In order to ease the work for firewalls, MPTCP defines only one TCP option but defines several subtypes as seen in the TCP option packet format in the verbatim 3.1.

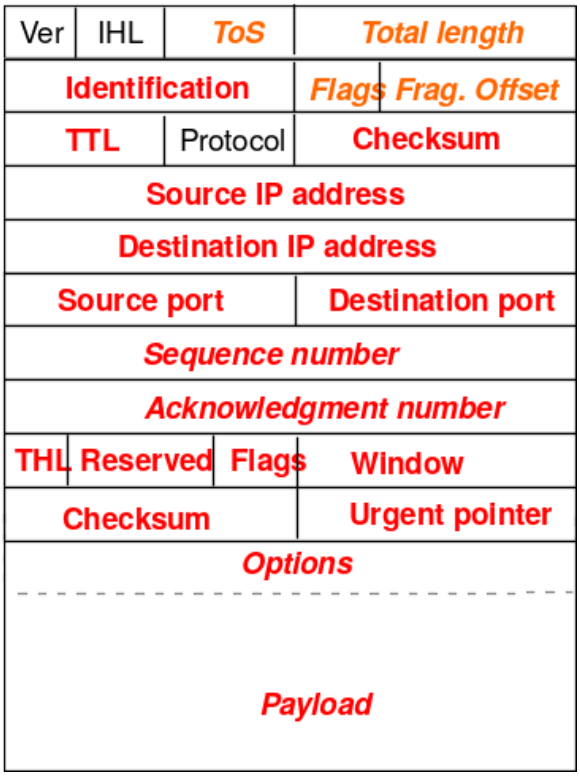

Figure 3.2: List of TCP fields possibly changed by middleboxes in red (Source: [210]).

Once an MPTCP connection is established, endhosts can advertise combinations of (IP, TCP port), add or remove MPTCP subflows anytime. These subflows, which we could define as TCP connections children of a more comprehensive parent TCP connection, can be used to achieve greater throughput or resiliency (indeed with the "backup" flag, MPTCP can create a subflow used only in case other subflows stop transmitting). It is worth noting that a host can create/advertise subflows with a same IP address, but with a different port number. The main challenge of such a protocol is the congestion control mechanism. It should not be more aggressive than

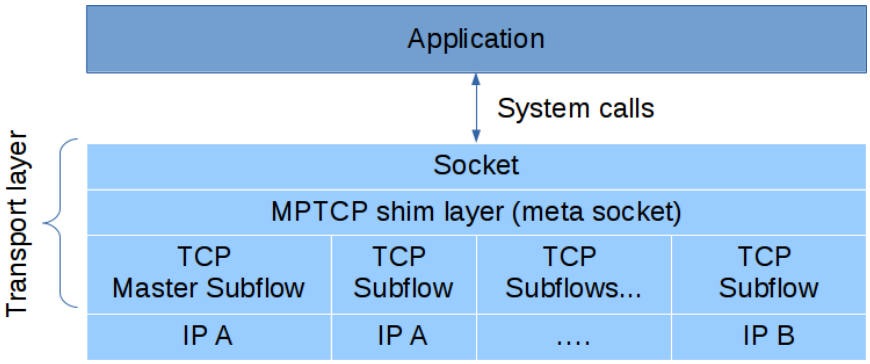

Figure 3.3: MPTCP: a shim layer in the stack. Subflows can share the IP address (using a different port) or have different IPs.

MPTCP, but at the same time it should use the unused capacity; it should balance the load over different paths, but without causing too much packet disordering so that TCP buffering can reorder them. Adequate path discovery is part of the solution and that is where LISP can help.

MPTCP consists in a shim layer as it can be seen on Figure 3.3, it is built between the application and the TCP stack that unifies several TCP connections, called "subflows" in the MPTCP context. A subflow is a TCP connection characterized by a tuple (IP_{source}, TCP $port_{source}$, $IP_{destination}$, TCP $port_{destination}$) and is assigned a unique subflow id generated by the MPTCP stack. We can alternatively define an MPTCP connection as a set of one or many subflows aggregated to feature at least the same set of service as a single path TCP communication.

MPTCP signals information with its peer through the use of TCP options. All the MPTCP options share the TCP option number 30 but the MPTCP option is further distinguished by one of the MPTCP subtype number listed in table 3.1.

The RFC6182 [211] lists a few functional goals that are deemed mandatory for a wide deployment of the protocol:

- MPTCP must support the concurrent use of multiple paths. The resulting throughput should be no worse than the throughput of a single TCP connection over the best among these paths.

- MPTCP must allow to (re)send unacknowledged segments on any path to provide resiliency in case of failure. It is advised to support "break-

before-make" scenarii, e.g. buffer the data when a (mobile) user loses temporarily all connectivity, to allow resuming the communication as soon as a new subflow gets available.

Rather than adding new features as SCTP does, MPTCP guidelines focus at not being worse than TCP and on wide deployment problems. [211] also lists three compatibility goals:

- The applications must be able to work with MPTCP without being changed, for instance via an operating system upgrade. It also implies that MPTCP keeps the in-order, reliable, and byte-oriented delivery[1].
- MPTCP should work with the Internet as it is composed today, that is with middleboxes blocking unusual payloads or even modifying the payload such as internet accelerators, Network Address Translator (NAT) etc. The best way to do this is to appear as a singlepath TCP flow to the middleboxes. Hence MPTCP relies on TCP options for signaling. TCP option space is scarce (40 bytes maximum per packet).
- MPTCP should be fair to single path TCP flows at shared bottlenecks, i.e. not be greedier. At the same time, MPTCP still shall perform better.

3.2 Connection process

3.2.1 Initiation

Supposing that the MPTCP extension is not disabled, and that the application remained unchanged, the MPTCP connection is initiated through the TCP socket interface via the **connect** system call. As per the MPTCP Linux system nomenclature, we call this first TCP connection the master connection. This call must generate a random key to be used during the TCP handshake as can be seen on fig. 3.4. This key is later hashed (only SHA-1 is standardized at the time of writing) and used by MPTCP to authenticate additional subflows.

[1] Nevertheless an extended API is being standardized in [212] for applications to squeeze more out of MPTCP.

Table 3.1: Exhaustive list of MPTCP options as presented in RFC6182 [211].

Subtype name	**Subtype id**	**Description**
MP_CAPABLE	0	This is the first option ever sent by TCP and must be present at each step of the initial handshake. It carries the cryptographic key that is used to identify the MPTCP connection and authenticate additional subflows.
MP_JOIN	1	This can be used to open new subflows once the first subflow connection has been completed, i.e., once a data ack has been received on the first subflow as shown in fig. 3.4.
DSS	2	The Data Sequence Signaling option can transport several information - concurrently - depending on the flags field. It can convey one mapping between DSN and SSN, e.g., the DSN 3000-3100 maps to the SSN range 420-520. It can convey the data ack that would acknowledge good reception of DSN up to 3100. When the FIN field is set, it also signals to the peer the final DSN of the connection.
MP_ADD_ADDR	3	This allows any host to advertise additional ports or IPs towards which the peer could initiate a subflow. The Linux implementation lets by default the client initiate new subflows in case there is a Network Address Translator on path.
MP_REMOVE_ADDR	4	This is the opposite of the previous option. In case the device loses an IP because of mobility (e.g., the access point became out of range), it may want to update its peer information
MP_PRIO	5	The sender of this option advertises a binary preference whether it prefers to receive data on this subflow or not. For instance a user connected via both wifi and 3G may prefer to prevent the other host . The receiver of this option is free to ignore it or to take into account
MP_FAIL	6	MPTCP provides optional checksum to detect network meddling with the packets. If such interference is detected (i.e., a wrong checksum), the subflow must be closed with an MP_FAIL option that indicates the last valid MPTCP sequence number received
MP_FASTCLOSE	7	Equivalent to a TCP RST, it signals to the peer the immediate closure of the connection and that it does not accept new data

Once other subflows are established, the master subflow can be removed as any other and holds no specificity. Upon SYN reception, the server generates also a key which is reflected by the client in the final TCP handshake ack. This allows the server to operate in stateless mode. Indeed an MPTCP stack needs to allocate more data structures than a legacy TCP connection to save the key, the list of subflows, their ids etc. For efficiency, the allocation of these data structures can be deferred until the moment the MPTCP negotiation succeeds.

As part of the network compatibility goal, MPTCP should provide an automatic way to negotiate its use, and upon failure of such a negotiation, fall back to legacy TCP. This fall back is also possible even after successful completion of the MPTCP handshake, in case no data ack is received during a certain time, or checksums are invalid.

3.2.2 Addition and closure of other subflows

The host can open a new subflow as soon as a DSS option with a data ack is received, which requires at least two RTTs since the very first handshake. Hence, the choice of the initial subflow can have an impact on the throughput, all the more important for short connections. Both the client and the server can create new subflows. Either the host initiates the new connection or it advertises a couple (IP, port) that the peer can choose to connect to. As a matter of fact, MPTCP map IPs to an id to convey subflow related advertisements since IPs are not reliable: they may be rewritten by external middleboxes. The policies are local and for instance in the Linux implementation, the server advertises its ports but let the subflow creation initiative to the client because of NATs that could invalidate It is worth noting that several subflows can be created from the same IP address with different ports. As a consequence, MPTCP refers to IPs with an identifier rather than by the IP in order to prevent the confusion induced by Address Translators (PAT/NAT). This may prove worthwhile to exploit the network path diversity, in case the network runs load-balancing and is exploited in Chapter 4.

There is no standard procedure and the subflow opening/closing strategy depends on local policies. It may be wiser to let clients initiate the connection though due to the presence of NATs. Subflow control can also be delegated to a third party controller [213], [214].

MPTCP supports in theory *break before make* scenarios, i.e., it can lose the connectivity of all its subflows, create a new one and resume the connection via the retransmission on this very subflow of the lost segments. This is especially useful in the context of mobility where the user can not always predict connectivity loss.

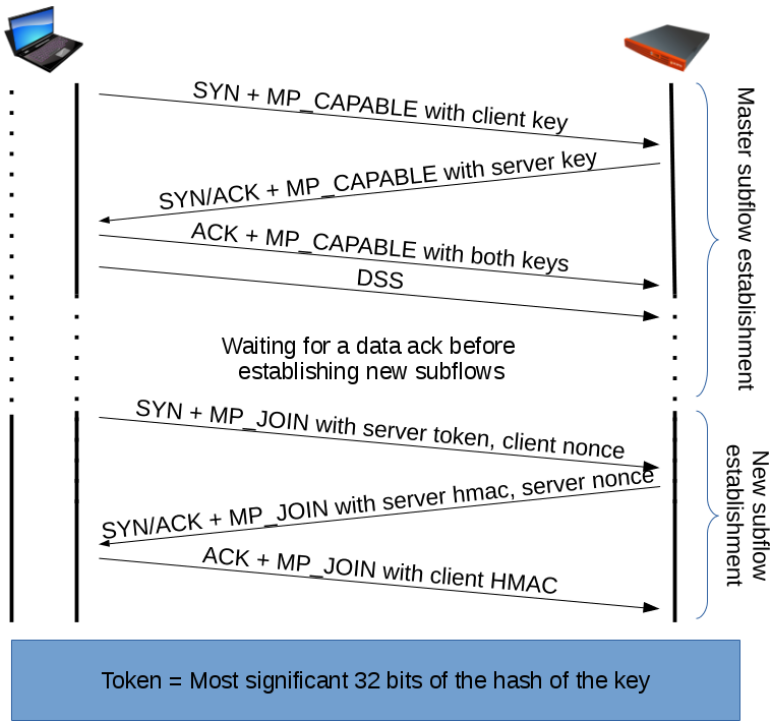

Figure 3.4: Illustration of used notations for two subflows.

3.3 Transmission of the flow of data

MPTCP allows an application to transmit a single flow of data, hence MPTCP needs a single Data Sequence Number (DSN) namespace to be able to split this flow of data across several subflows and later reassemble it. An immediate idea is to write the DSNs in the TCP sequence number field but as some middleboxes rewrite the TCP sequence numbers (for security reasons) or check that they are in-order (e.g., firewalls), MPTCP can not use the TCP vanilla sequence number. MPTCP solves this dilemma by adding a global DSN namespace shared among subflows and map the DSN to the TCP vanilla sequence number, then called Subflow Sequence Number

(SSN), of the subflow(s) they are sent. The mappings are exchanged through TCP options called DSSs (see Table 3.1). A similar approach needs to be undertaken for the acknowledgment of the DSN: we refer to MPTCP-level ACKnowledgements (ACK) as Data Acknowledgment (DACK) in the rest of this dissertation. They are exchanged through the same DSS option.

3.3.1 Congestion control

TCP fairness can be a controversial topic: a malicious TCP user who wants more bandwidth can create additional TCP connections (as many download accelerators do) to increase its share at the bottleneck. In the following, we consider well-behaved hosts since this is the usual framework priori to any congestion control reasoning.

Without specific congestion control algorithm, a multipath transport protocol would adopt a similar behavior at the bottleneck since being an end-to-end technology, it has no information over the topology. TCP users would see their bandwidth decrease and MPTCP deployment hindered. Under these conditions, how to achieve both the fairness and higher throughput ? Knowing if two subflows share/make use of a same resource (e.g., a link or a router) would allow to run a congestion control on sets of subflows. Clustering techniques ([215], [216] for instance) have been developed to detect bottlenecks based on delay and loss patterns. These techniques need to be foolproof as false negatives generate bandwidth stealing. This is a difficult task without help from the network as the heuristics need to work across a wide range of configurations, such as the router buffering policies etc... Their efficiency is also difficult to evaluate for the same reasons but even if a perfect scheme existed, relying on it depends on the fairness notion.

This conservative approach considers that all subflows share a bottleneck and that their additive component should be coupled. MPTCP current congestion control standards only modify the congestion avoidance phase of TCP: the decrease phase remains the same as in TCP but they couple the increase MPTCP congestion window with the congestion window of its subflows. Several congestion control have been proposed such as LIA (Oda, Hisamatsu, and Noborio [217]), OLIA (Khalili, Gast, Popovic, *et al.* [4]), MPCubic (a variant of Cubic for TCP [218]) wVeigas (weighted Veigas [219]), eXplicit MultiPath (XMP) [220]..... XMP is interesting because it uses external information (i.e., ECN signaling) to influence traf-

fic splitting. This can be useful to relay congestion information as loss events can be very damaging to MPTCP communications. Similarly, CWA-MPTCP [114] relays radio signal strength to the stack to dynamically adjust the window sizes. This allows to take active instead of reactive measures.

From our literature survey, LIA and OLIA seem to be the most popular congestion controls so we present the formulas used to update its congestion window. First for LIA:

- $w_i = w_i + \min(\frac{a}{w_i}, \frac{1}{w_r})$ per acknowledgment on path i
- $w_i = \frac{w_i}{2}$ per congestion event on path i

with a being an aggressiveness factor updated once in a while (per window a priori) and equal to:

$$a = \frac{\max_r(\frac{w_i}{rtt_i^2})}{\sum \frac{w_i}{rtt_i}^2} * \sum_i w_i$$

with :

$$\begin{cases} w_i \text{ the window size on path i} \\ rtt_i \text{ the round trip time on path i} \end{cases} \tag{3.1}$$

The *min* in the first equation ensures that MPTCP is never more aggressive than TCP on a single path.

It is important to remember that the advertised receive window is shared between subflows. As such there may be cases where a subflow is capable of sending data, i.e. has free space in its congestion window, but there is no more space in the receive window. To mitigate these instances of HoL, a feature called Opportunistic Retransmission (OR) was implemented [221] in the Linux kernel, which in such cases retransmits data hoping to solve the head of line blocking. OR can be used in conjunction with slow subflow penalization: if a subflow holds up the advancement of the window, MPTCP can reduce forcefully its congestion window along with its slow start threshold.

MPTCP is said to be more efficient for long communications because of the subflow overhead, yet Chen, Lim, Gibbens, *et al.* [222] see a positive effect. The results are surprising but the data is not available. The authors explain that the congestion control being uncoupled during the slow start phase, MPTCP does allow to carry more packets (when the subflows follow disjoint paths). On the contrary, this very behavior can be detrimental when

the subflows share a bottleneck as the successive slow start bursts may create congestion events. A solution proposed in [223] and being studied by the MPTCP working group is to couple also the slow start phase.

3.3.2 Scheduling

The scheduler chooses when and on which subflow to send which packets. A good scheduler should attempt to reduce the probability of HoL. For instance opportunistic retransmission and penalization are reactive mechanisms that waste bandwidth. The Linux implementation we used included two schedulers:

- The 'default 'scheduler sorts subflows according to their RTT and sends packets on the first subflow with free window.
- A round robin scheduler that forwards packets in a cyclic manner on the first subflow with free window available.

Retransmission timeouts (RTO and delayed acks) need to be chosen with great care since a subflow RTO or out of order arrivals can provoke HoL blocking faster than in the single path case, as also explained in [224]. For instance, some of the state of the art schedulers propose to send packets out of order so that they can arrive in order [225].

3.4 MPTCP state machine

As a preliminary step before implementing MPTCP in NS-3 (See Chapter 7), we formalize the current status of the specifications. We extended the connection closure Finite State Machine (FSM) described in [183] to cover the whole protocol in Figure 3.5, i.e. while the active and passive close are presented as a diagram in [183], we extended the visual description to our interpretation of the standard. While being similar to TCP, we chose to split the **ESTABLISHED** state into the **M_ESTA_WAIT** and **M_ESTA_MP** states to distinguish between a state where MPTCP waits for a first Data Acknowledgment (DACK) (**M_ESTA_WAIT**) before being fully established (**M_ESTA_MP**) and be able to create additional subflows. We also mapped for each MPTCP state the states in which TCP subflows can be as well as which MPTCP options could possibly be sent. The tabulated study report is available online [226].

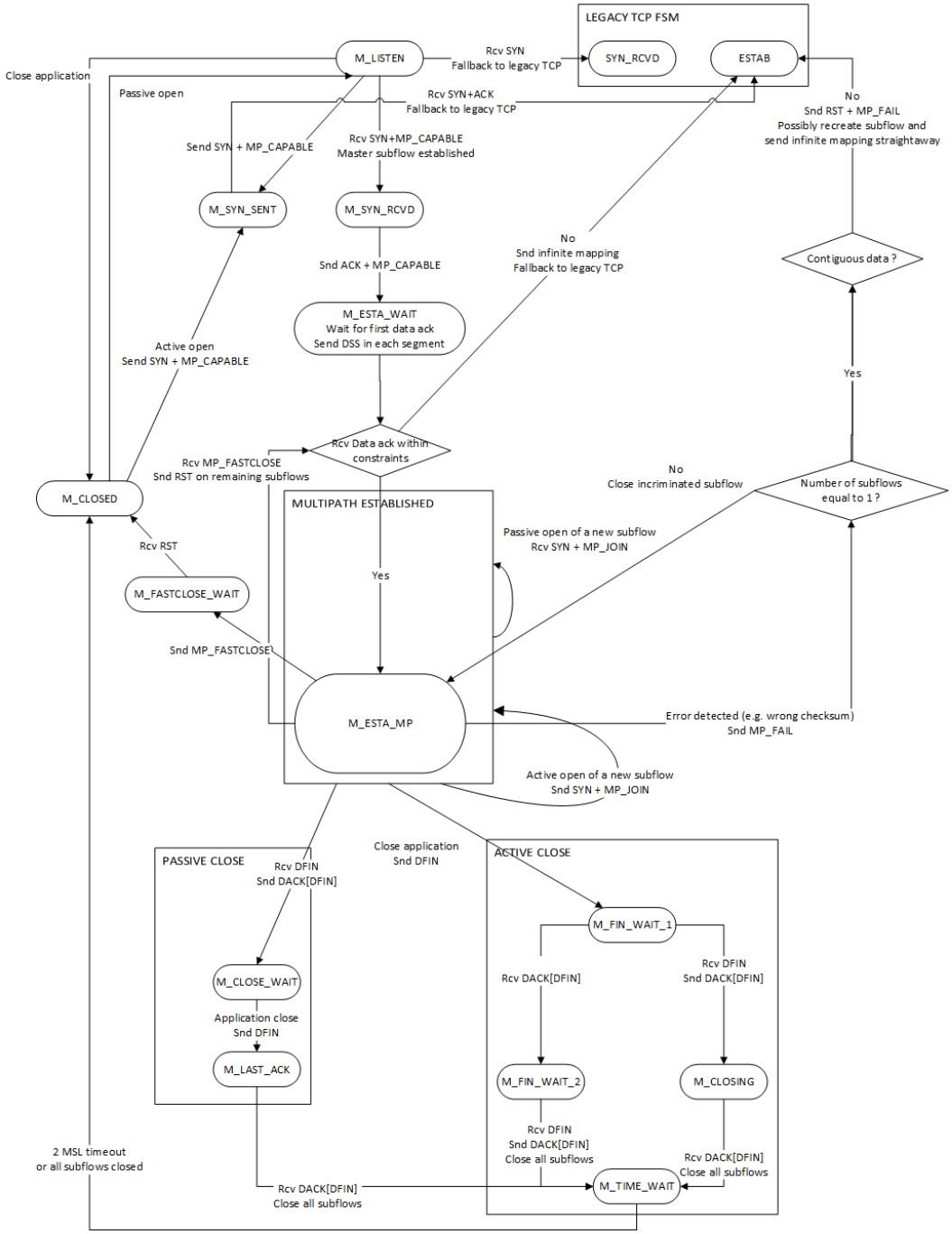

Figure 3.5: MPTCP state machine.

3.5 Associated challenges

We already mentioned a few challenges in the previous sections. Our stance is that MPTCP is already robust enough by design to fulfill the network and application compatibility goals (as confirmed by the trust of several companies such as OVH, Apple, Citrix that developed MPTCP-based products).

The resiliency goal has also been proved to work, i.e. when one link fails, retransmission of the packets is done on another subflow. The main obstacle to MPTCP and multipath deployment protocols today remains the throughput and fairness goals. While there are examples of increased throughput through the use of MPTCP (e.g., the fastest TCP connection was done with MPTCP [227]), this requires specific conditions such as enough buffer and homogeneous paths; there are also cases, as in [209], where MPTCP performs worse than TCP (in terms of throughput) on the best available path. This does not comply with the objective of doing always better than TCP. MPTCP must acquire the intelligence to distinguish when and which subflows to use to perform well. Reaching this goal is made even harder with the throughput goal since MPTCP is less aggressive than TCP on every subflow.

As for the latency criteria, in latency sensitive applications such as Voice over IP (VoIP) applications, path heterogeneity with MPTCP can lead to higher jitter and decrease Quality of Experience (QoE) metrics. In [5], Wifi and LTE used in a single path context generate a latency variation between 12 and 24ms. When both channels are combined with MPTCP it led to a 97ms application latency variation.

Path management is also a problem - though less studied - since creating many subflows with the hope of exploiting path diversity can hurt the performance (due to competition between subflows). The problem is threefold:

1. transport protocols being end-to-end, hosts do not know the topology;

2. even if the hosts knew the topology, they can not enforce a forwarding path. Segmented routing may provide a partial solution in this regard.

3. As shown in Nikravesh, Guo, Qian, *et al.* [5], some services such as Domain Name System (DNS) may try to answer differently depending on the user location, retrieved from its IP. This can lead to suboptimal

answers.

As for wide area networks topologies, there usually is more than one path between source and destination. It can be because of intra-redundancy or because several ISPs compete on the same path. There is ongoing work to exchange topology information between nodes that could solve Item 1, for instance with Path Computation Elements (PCEs) or at the Application Layer Traffic Optimization (ALTO) working group [228].

Topology is a critical information that operators may not be fond of leaking, hence some approaches look at how to provide an overview of the topology through sparsification techniques [229]. From the previous technologies, a host can deduce an optimal number of subflows, but this may prove pointless if the forwarding problem 2 is not solved. As such, solutions in locally controlled environments such as an Software Defined Network (SDN) datacenter seem appropriate.

If the throughput goal 3.1 constraint is relaxed, one can imagine a realm of other possibilities for multipath protocols. Multipath incentives do not stop to throughput aggregation and as such one could imagine modes where the cost of an interface helps choosing over which interface to send packets as in Secci, Pujolle, Nguyen, *et al.* [230]. The cost could be given by the energy consumption of the interface or depending on its fare rate. The user could also set trade offs such as losing 30% of the optimal throughput if it allows for a fairer distribution between subflows. Ledbat-multipath [231] is one of such alternative modes. Information that used to be of little interest with one path are now helpful in a multipath context. For instance, if the MPTCP layer is aware of the data emission profile, it can adapt the scheduling to favor throughput (bulk transfer) or schedule packets so that they arrive early at the receiver (at the end of a burst).

3.6 Summary

Though simple at first sight, MPTCP embeds several mechanisms to work across a variety of network and middlebox-related difficulties. According to [232], this is successful as only 5% of the Apple voice recognition system fall back to legacy TCP. This compatibility implies that MPTCP features only a subset of SCTP does but it also increases the implementation complexity as MPTCP is by nature intertwined with TCP. The network stack needs

to handle not only the seemingly one MPTCP option but at least the eight subtypes visible in Table 3.1, plus the different fallback mechanisms as well as mapping DSNs to their respective sequence number.

Choosing the optimal number of active (MPTCP can create more subflows but mark them as backup subflows) subflows remains an open problem. The path management problem also explains why many of the commercial products embed MPTCP into proxy middleboxes (Gigapath[2], OVH[3], Tessares[4]); certainly they grant the benefits of MPTCP to legacy clients, but the middle boxes also know more about the network diversity. The scheduling and congestion control aspects are closely related but require more work to reach Pareto-optimality.

[2] `https://www.ietf.org/proceedings/91/slides/slides-91-mptcp-5.pdf`.

[3] `https://www.ovhtelecom.fr/overthebox/`.

[4] `http://www.tessares.net`.

Chapter 4

A multipathed crosslayer network architecture

The added value of each path increases with its disjointness relative to the previous path(s): otherwise, the forementioned increased confidentiality, reliability do not happen. Many challenges are easier to solve in a Data Center (DC) as the whole architecture is owned by a single tenant, who can adapt its infrastructure as it sees fit. In a first step we propose a crosslayer architecture to ensure end-to-end disjoint multipath communications within the DC. External reliability in data-center networking is today commonly reached via forms of provider multihoming, so as to guarantee higher service availability rates. In parallel, Internet users also resort to multihoming via the different interfaces (Wi-fi, 3G, Wired) of their device. Both practices add path diversity between Internet users and servers: it is difficult for end-systems to influence the forwarding path within the Internet but it remains possible to influence the edge routers. Thus in a second step we present a holistic end-to-end multipath architecture for Cloud access and inter-DC communications, and defend its possible implementation using three recent protocols at the state of the art functionally acting at three different layers: MPTCP, LISP and TRILL[1].

[1]The contents presented in this chapter have been published in[213], [233].

4.1 Introduction

Multipath communications represent both a chance and a hassle for the current Internet. On the one hand, legacy Internet protocols have mainly been designed with a single active path paradigm in mind. On the other hand, multihoming practices at both endpoint and network levels can offer path diversity to data connections, potentially allowing effective end-to-end load-balancing and multipath communications [234]. Cloud networking is at the forefront of this trend. Indeed it increases Cloud availability guarantee via different techniques: the external interconnection of DCs with multiple independent provider links, the deployment of intra-DC multipath layer-2 protocols and IP endpoint multihoming. We tackle the challenge of establishing coordinated multipath communications in a Cloud environment composed of multihomed data-center networks and users.

We adopt a protocol interoperability perspective, aiming at increasing throughput and resiliency of Cloud communications, within and across administration domains. We present both stateless and stateful solutions to end-to-end path diversity management, involving novel protocols standardized in the last months: the Multipath TCP (MPTCP) [183], Locator/Identifier Separation Protocol (LISP) [235] and the Transparent Interconnection of a Lot of Links (TRILL) protocol [236].

In the first sections we describe the architecture in an abstract way, then in Sections 4.5 and 4.5.5 we present concrete implementation and results of in order a LISP-MPTCP cooperation, and LISP-TRILL cooperation.

4.2 General Architecture

Our goal is to resort to multipath communication to improve cloud access and inter-cloud performance, more precisely to increase throughput . Decreasing transfer times improves the user QoE, and boosts storage synchronization and virtual machine migration as well. As depicted in fig. 4.1, the envisioned network environment involves Cloud service users, potentially mobile, and data-center networks, with user-Cloud, intra-DC and inter-DC communications. Under this perspective, there are major challenges to address: How to send and receive data packets along different paths without disturbing applications? How to select disjoints paths in order to provide higher resiliency and to avoid bottlenecks? How to ensure packets really

follow disjoint paths?

It is worth noting that using different paths in a uncoordinated way, TCP performance can be decreased rather than increased, namely because packet arrival disorder may trigger retransmission that may disturb the application workflow. Moreover, selecting Internet-wide end-to-end disjoint paths is commonly considered as an unrealistic dream, given the high heterogeneity and versatility of the Internet ecosystem. However, in the following we present rather simple functional blocks and protocol coordination mechanisms that are one step toward this goal, under realistic assumption and partially already available functionalities.

In the following, MPTCP is considered as the transport layer protocol, as it is undoubtedly more scalable than other proposed alternatives and already deployed, even if mostly at an experimental extent for the moment. MPTCP [183] is a TCP extension making use of several TCP subflows when possible to improve throughput and resiliency. It first asserts if the destination is MPTCP compliant (middleboxes such as firewalls might prevent the use of unknown TCP extensions), otherwise it falls back to legacy TCP. Once an MPTCP connection is established, endhosts can advertise their ips, add or remove TCP subflows at anytime. Basically a subflow could be defined as a TCP connection embedded in a more comprehensive TCP connection. MPTCP can mark a subflow as *backup* only and will use it only if the other subflows stop transmitting. Joint congestion control techniques using many subflows are documented in standardization documents.

4.2.1 Cloud Network Elements

The Cloud network architecture we envision should be sufficiently flexible to be extended and adapted to different technologies. The key role is taken by four node types:

- Virtualization Cloud servers, with MPTCP enabled at the hypervisor level.
- Cloud users, able to establish MPTCP connections even when single-homed.
- Border nodes, i.e., routers at the DC and user borders with its Internet Service Providers.

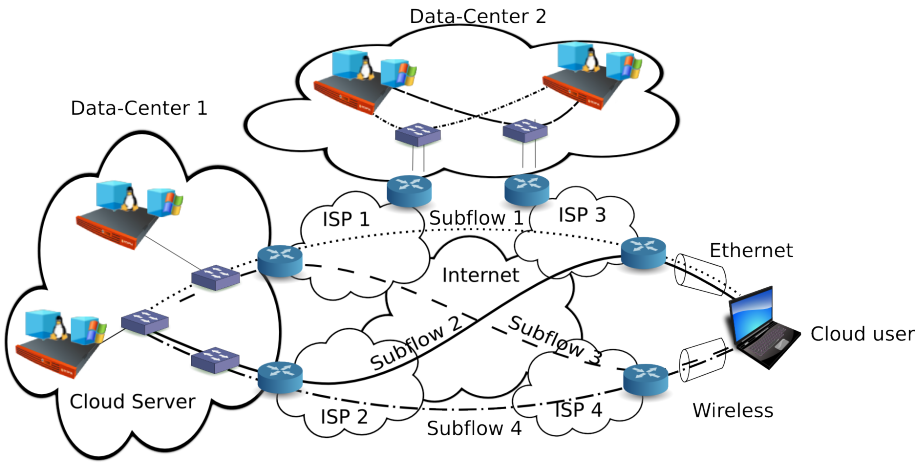

Figure 4.1: Representation of the Cloud Networking Context.

- Cloud Controllers, managing a DC network, enabling path discovery and establishment.

The implementation of MPTCP at the hypervisor level allows its scalable deployment: it represents a sort of TCP optimizer and allows virtual machines to be MPTCP agnostic. MPTCP can open multiple TCP subflows for a single TCP connection. These subflows differ with respect to their source or destination IP (if node(s) multihomed), or via the subflow (in case one server is singlehomed). Our proposition is an advanced yet simple protocol architecture that basically stitches MPTCP subflows to Ethernet- and IP-level paths, which are as much disjoint as possible. The delivered network service improves resiliency and throughput, but comes at a cost: finding diverse paths adds an overhead in processing time as well as in latency that may impede performance and scalability. Therefore, we foresee a special treatment only for flows for which the pros outweighs the cons, as described hereafter.

4.2.2 Functional blocks

A generic crosslayer multipath protocol architecture should implement the following blocks:

- Flow Qualification Service: at the Controller or hypervisor level, it identifies which connection benefits from a multipath communication. The online classification ranks different metrics of a flow, e.g., jitter, latency, throughput, security, so that if needed we can sort flows according to policies. For instance, it appears appropriate to distinguish *elephant* flows (i.e., long-lived flows) from *mice* (i.e., short-lived) flows. The hypervisor triggers signaling for multipath communication for elephant flows only.

- Flow Monitoring Service: at the Controller or hypervisor level, the state of either the network or the application might evolve during a communication, thus becoming obsolete. If a physical path becomes unreachable or underperforms, one may want to update the multipath configuration.

- Path Discovery Service: this service can be considered as a Traffic Engineering Database (TED) service; at the Controller or hypervisor level, it collects various information about the available path diversity, such as DC Link States, MTUs, load-balancers, DC and Internet topology, multipath capabilities, between the Cloud servers and users through border nodes . While DC-level discovery can be performed using existing operational tools, retrieving Internet path information can be difficult due to various factors: high versatility, unreliable knowledge of the topology, loose paths, etc... A major obstacle might be the reluctance of ISPs to share their topology with potential attackers, i.e., outsiders. This can be mitigated through techniques such as sparsification topologies, i.e., you apply transformations to your graph in order to hide the detailed topology while answering the question answered as is done in Scharf, Wilfong, and Zhang [229]. In practice, the service is to be decomposed into an Intradomain Discovery Service (i.e., a controlled network scope, such as a DC network), and an Interdomain Discovery Service. SDN technologies or the recent Interface to the Routing System (I2RS) IETF working group [237], [238] are ways to share the routing system information. One might also want to replicate the TED as a distributed database or cache, splitting it per layer or per subdomain, each using a pull or push discovery (these considerations are out of the scope of this document).

- Path Computation Service: at the Controller or hypervisor level, this service is in charge of selecting paths to assign to flows when requested. The decision process takes into account constraints specified in the request (e.g., a jitter-sensitive path), and uses the Traffic Engineering Database (TED) to resolve matching paths and node capabilities.

- Path Enforcement Service: once paths are computed, packets have to use these paths. This is possible through stateful and stateless modesÂă:

- Stateful signaling: network path setup on per flow basis, e.g., using SDN or MPLS-based architectures.

- Stateful explicit forwarding: the source (i.e., the Cloud server's hypervisor, actively or passively via the Controller) lists in each packet the nodes the packet must pass through.

- Stateless forwarding: the source exploits network load-balancing algorithms and crafts packets headers so that they follow foreseen paths [239].

4.2.3 Multipath Communication Signaling

Figure 4.2 highlights the signaling process and the associated functional blocks. For example, a TCP connection is established between two Virtual Machines (VMs) in two different DCs in site A and site B. The hypervisor hosting the VM at site A detects the flow and queries the Flow Qualification process (could be a process running at the hypervisor, or an external controller). If the flow qualifies for a multipath connection. The hypervisor acting as an MPTCP proxy verifies if endpoint B is MPTCP capable (or its hypervisor is); if yes, it queries the Controller for multipath capabilities to its destination. Controllers at the two sites may be able to collaborate in the computation of the paths. The Controller(s) compute(s) and enforce(s) the paths based on the information recovered by the Path Discovery Service (running ex-ante in a push mode or ex-post in a pull mode). In the stateful mode, intra-DC paths are stitched with MPTCP subflows by the hypervisor, and at border nodes inter-DC (loose) paths are stitched with intra-DC paths. In the stateless mode, the hypervisor can be informed

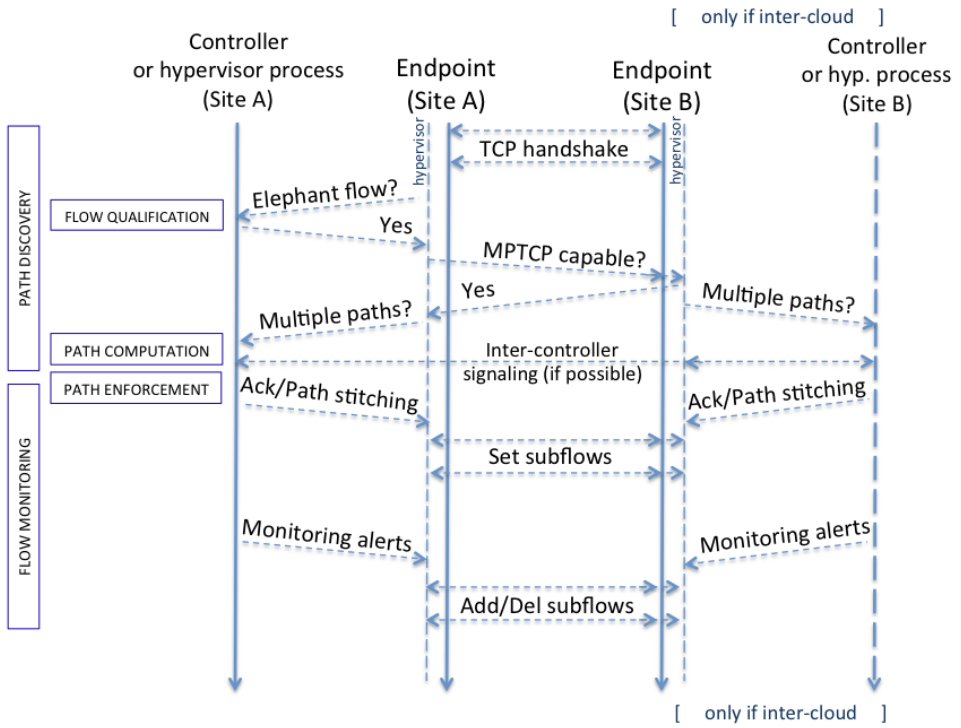

Figure 4.2: A signaling example in the case of extra-DC multipath communication.

about the presence of load-balancers and possibly of the available path diversity at intra-DC and/or inter-DC segments. Data is eventually transferred using the multiple end-to-end paths. Guaranteeing a level of disjointness at the intra-DC and/or inter-DC level can allow boosting throughput, thus reducing transfer times. Appearance of network events such as node/link congestions/failures/additions could trigger respective hypervisors, causing subflow deletion or addition, possibly path re-computation and enforcement.

In case the endpoint B is a user terminal, the right side of fig. 4.2 does not apply: there is no need for path stitching at site B. On top of that if the user terminal is multihomed, the more access interfaces there are the higher number of subflows can be opened. It is worth noting that neither the user terminal nor the DC nor the Cloud virtualization server need to be all multihomed: to open more than one subflow path, just one segment needs to have path diversity, where the segments are the intra-DC segment between Cloud servers and DC border nodes, the extra-DC segment between

border nodes, and the end-to-end transport segment between terminal interfaces. , At each Cloud network segment, different protocols can act, hence interoperability between them is needed to propagate path diversity across segments. One solution could be having SDN protocols such as OpenFlow as ubiquitously as possible between source and destination, with however important scalability concerns. A reasonable alternative is to rely on three new protocols recently defined to independently handling path diversity at each segment, with an adequate coordination as proposed hereafter.

4.3 A design using MPTCP, LISP and TRILL

In this section, we present one possible peculiar implementation of the previous architecture making use of three novel protocols: apart the already mentioned MPTCP [183] at the end-to-end transport segment, LISP [235] handles path diversity at the extra-DC segment between border nodes, and TRILL [236] can enable multipath Ethernet-layer communications within the DC. All these protocols emphasize compatibility with existing infrastructure and incremental deployment.

Our intention by using these protocols is to address all the previously described use-cases as well as distributing path diversity across layers in a scalable and practical way. A key factor for this purpose is the multipath ability they all offer (though optional in TRILL). These protocols do have also in common the functionality of mapping one upper layer logical point to many lower-layer logical points: one application port to many IP interfaces for MPTCP, one IP address to many IP routing locators for LISP, one MAC address to many Ethernet routing locators for TRILL. None of them are fully standardized, yet they have all been implemented to some degree: many vendors have started commercializing TRILL since a few months (e.g., Cisco, Huawei, Fujitsu); a TRILL OpenSolaris opensource version exists, and a Linux version is expected to be released closely. FreeBSD (OpenLISP) and Linux (LISPmob) versions of LISP are available; LISP is also already available in Cisco routers. Finally, a stable Linux version of MPTCP exists, adaptable to Android smartphones, so having it implemented at customer end-points and Cloud virtualization servers is not unrealistic. In the following, we first synthetically describe LISP and TRILL in Sections 4.3.1 and 4.3.2, and then discuss cross-layer coordination in the specific architecture.

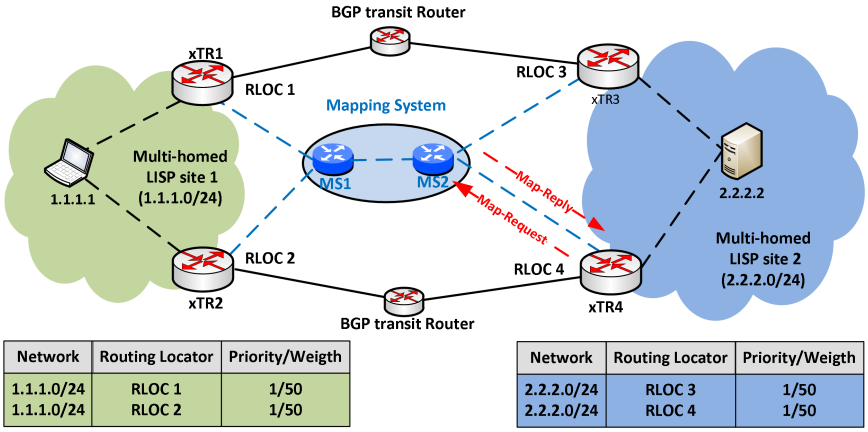

Figure 4.3: LISP communications example.

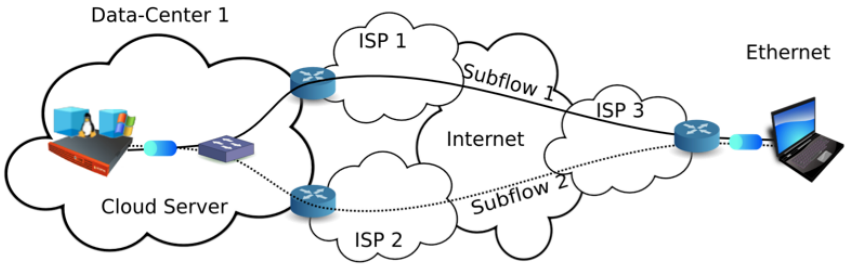

Figure 4.4: Example of a multihomed data center and associated MPTCP subflows.

Finally, we present the results of partial experimentations.

4.3.1 Locator/Identifier Separation Protocol

IP addresses assume today two functions: localization and identification of its owner, which induces a few problems one of which is scalability. IP addresses need to be distributed according to the network topology, which conflicts with ease of use. Provider independent addresses as well as the Internet growth tend to increase global Border Gateway Protocols forwarding information databases, thus slowing lookups and increasing device costs.

Consider the example in fig. 4.3: the traffic sent to the 2.2.2.2 host is encapsulated by the source's Ingress Tunnel Router (ITR) toward one of the two destination's RLOCs. The one with the best (lowest) priority metric is selected, which at reception acts as Egress Tunnel Router (ETR) and

decapsulates the packet, before sending it to the destination. On the way back to 1.1.1.1, RLOC4 queries a mapping system and gets two RLOCs with equal priorities, hence performs load-balancing as suggested by the weight metric (RLOC1 is selected in the example's packet). In order to guarantee EID reachability, LISP uses a mapping system that includes a Map Resolver (MR) and a Map Server (MS). Typically merged as a single MS/MR node as depicted in fig. 4.3, a Map Resolver holds a mapping database, accepts MAP-REQUESTs from xTRs and handles EID-to-RLOC lookups. A Map Server receives MAP-REGISTERs from ITRs and registers EID-to-RLOC in the mapping database. In practice, many MRs are geographically distributed and relay MAP-REQUEST messages using a specific protocol called LISP Delegated Database Tree (LISP-DDT) protocol. This is the case of the LISP Beta Network testbed[2].

As a backward compatibility feature, if a source (or destination) site is not yet LISP compliant, the traffic might get encapsulated (or decapsulated) by a Proxy ITR (or ETR). Furthermore, the usage of RLOC priorities and weights in the mapping system allows inbound traffic engineering, suggesting a best RLOC or an explicit load-balancing. In short, LISP routers are border nodes that advertise their RLOCs so that a remote LISP site knows it needs to tunnel packets to those routers to reach hosts in that remote site (i.e. an EID). LISP allows IP mobility (i.e., a machine keeps its EID but updates its RLOCs, for both mobile users and mobile virtual machines [240]), and also give hints if the site is multihomed or not, i.e., if the site is reachable by different providers. In the case the RLOCs of a site belong to different prefixes, they could belong to different providers, and the site is said to be multihomed. In this case, it would be interesting to have additional MPTCP subflows, whose path is split at the LISP site border, since they could follow different Internet paths, if not end-to-end, at least along a segment of the Internet path.

4.3.2 Transparent Interconnection of a Lot of Links

TRILL implements a logic close to LISP's at the Ethernet layer in order to solve switching tables scalability problems and improve network efficiency for DC environments. As LISP, TRILL uses data encapsulation with an outer standard header and a shim specific header, and it tunnels data units

[2]LISP Beta Network testbed (website): `http://www.lisp4.net`

from an ingress node to an egress node, with a mapping system to update association of endpoint (MAC) addresses to network locator (MAC) addresses (this is a key feature as VM are migrated across DC racks, hence their location can be updated). On the other hand it differs from LISP since it integrates a multi-hop routing logic between ingress and egress nodes based on IS-IS (though LCAF somehow fills this gap). This is the reason why TRILL bridges are called "RBridges "(Routing Bridges). It is incrementally deployable: RBridge may be incrementally deployed, with standard Ethernet segments between RBridges, with each RBridge terminating a spanning tree instance. Multipath communications are therefore possible between RBridges at the Ethernet layer. It is worth mentioning that alternative Ethernet routing protocols exist, namely the IEEE 802.1aq SPB and OpenFlow, which do allow Ethernet multipathing too, yet both require a complete deployment at all bridges and especially for this reason they are, for the moment, considered as less scalable and versatile.

4.4 Specific Architecture

In an heterogeneous Cloud environment with MPTCP, LISP and TRILL, the Cloud network has MPTCP enabled at both endpoints, at the DC side either directly in the VM or (more reasonably) as a virtual middle-box in the hypervisor. TRILL is deployed between the Cloud virtualization servers and the DC border nodes, noting that an increasing interest in standardization activities is given to the implementation of RBridges also as virtual bridges at the hypervisor level. LISP is implemented at DC border IP routers, and it can be implemented by user endpoints (see [241]) handling each access interface as an RLOC, hence complementing MPTCP with an inbound traffic engineering control-plane when both are enabled in the user endpoint.

Therefore, from the one hand (DC side) TRILL and MPTCP coexist at the hypervisor level, and from the other hand (access side) LISP and MPTCP coexist at the user mobile node. It is also worth noting that, since a few weeks, the LISP data-plane is implemented in the well-known virtual bridge called OpenVSwitch, hence pushing down to the hypervisor LISP encapsulation/decapsulation and letting the three protocols coexist in the same DC node. While MPTCP capability at the endpoints is mandatory in order to enable multipath communications to the application layer, our

architecture does not require LISP or TRILL to be implemented concurrently; a partial deployment with at least one of them is needed to associate loosely diverse path to subflows, yet both together would allows reaching higher performance. Finally, about the Cloud Controller, it can be based on classical Network Management Systems and the like, or on an SDN open controller accessed for instance via OpenFlow, or on a PCE [241] generalized for non-MPLS environments, or a mix of these technologies.

In the following, for each component described in the Functional Blocks section of the general architecture, we associate a specific solution based on MPTCP, LISP, TRILL and peculiar functionalities to close the gap between theory and practice.

Flow qualification: the process can be implemented either as collocated with the hypervisor or as external server. The advantage of the latter case is to exploit network-level information and offload the hypervisor by excessive signaling, and also to implement advanced qualification criteria for example based on destination's information and white/black lists. However, if simple criteria are used, e.g., a binary classification as elephant or Âămice flow, the implementation in the hypervisor does not need external information and appears as more appropriate. In such a case, a possible solution could be Mahout [242] implemented at the hypervisor level; basically, whether the socket fills up quicker than its emission rate, it marks the flow as elephant. In this case, we pursue the process, and proceed as explained in fig. 4.2.

Flow Monitoring: it is left to the MPTCP congestion control mechanism using per-subflow windowing. Upon failure of a TCP subflow (i.e. window size inferior to a threshold during a certain period of time), the cloud server may request a new path to the Controller in order to replace this subflow with a new subflow following a new path. This request may notify the deficient path to the discovery service which may update the states about that specific path. As the MPTCP congestion control mechanism is per subflow, it is important that a subflow keeps using paths of equivalent quality; indeed, a change to a path of lesser quality may decrease the window, yet it takes time to recover the original window value.

Path Discovery: it can be decomposed into two sub-services with different constraints. intra-DC discovery service: in DC environments, the as-

signment of VMs to servers should be orchestrated by a Cloud Management System that knows where each VM is placed. TRILL support this through an oracle-like system (non mandatory) called "directory system"[243], which returns the destination RBridge associated with a MAC, instead of resorting to ARP flooding, thus reducing L2 broadcast traffic. Associated to this information, the IS-IS substratum readily allows TRILL campus topology synchronization at RBridges, which can be enriched with TE metrics adopting the ISIS-TE extensions defined for IP networks [244] to TRILL-based Ethernet networks. Interdomain discovery service: MPTCP discovery is quite straightforward; if the distant host does not answer with the MPTCP capable option, then the connection falls back to legacy TCP. If both hosts are MPTCP compliant, they can either advertise their different IP interfaces to the other host, or directly try to open new subflows with these EIDs. As far as LISP is concerned, the hypervisor or the Controller can be easily enabled to query mapping servers to retrieve the RLOCs of the local site and the destination, along with LISP load-sharing information. Moreover, we may also try to rank Internet paths between xTRs, for example adding a TED companion to the LISP mapping information, using providers' services such as [245] or PCE-based [241], knowing however that Internet path information accuracy can be low. The Controller could manage a single TED built merging TE metrics related to TRILL Ethernet routing tables, LISP RLOC metrics and inter-domain path metrics, as a separate database regularly and on-demand pulling data from TRILL, LISP and external databases.

Path Computation: this service will use the information gathered by the Path Discovery Service to compute a number of paths according to different constraints, which might be passed in the request (e.g., latency, jitter, throughput, bottleneck bandwidth). In the case of inter-cloud communications, the PCE communication Protocol (PCEP) [241] could be used to allow distributed computation between DCs, including also the possibility to involve the PCEs of external ISPs. The extra-DC path computation between LISP endpoints can include the possibility of involving Reencapsulating Tunnel Routers (RTRs) between the ITR and the ETR [246] if LCAF is enabled.

Path Enforcement: both stateful and stateless methods are conceivable.

- Stateful mode: the computed path is enforced in a source-routing fashion. The way this can be implemented in TRILL and LISP does not rely on end-to-end reservation as in MPLS networks, but it implies the hypervisor is able to craft TRILL and LISP packets. This is proposed by [247] for TRILL: the hypervisor, called "TRILL smart endnode" directly queries the TRILL directory and encapsulates the packet with a TRILL header to lighten the load on the next RBridge. Even if not proposed elsewhere, with the above mentioned implementation of the LISP data-plane in virtual bridges such as OpenVSwitch, the hypervisor could become a "LISP smart endnode" as well. Conversely, explicit path prepending is described for LISP with RTRs and LCAF [239][246], allowing the enforcement of multihop xTR chains, and is currently not offered by TRILL. On the other hand, TRILL supports extensions, so we can imagine a similar feature implemented at RBridges.

- Stateless mode: with TRILL, we cannot use VLANs anymore to force a physical path, since any VLAN can get encapsulated in shared transport VLANs by the TRILL campus. However, with multipath load-balancing enabled, one can carefully compute the TRILL header's Time To Live (TTL) field so that packets with the same TTL follow the same path as of the result of hashing functions used in load-balancing, similarly to how described in [236]. This technique allows Controllable per-flow load-balancing and prevents packet disorder. Similarly, the TTL in the IP header can also be tuned to enforce extra-DC egress load-balancing at ITRs.

To sum up, the stateful method adds LISP and TRILL headers overhead to packets, but in a controlled environment such as a DC, the MTU should not be a problem. As for the stateless method, enough LISP and TRILL nodes need to share the same hashing algorithm to make it interesting, which is a reasonable assumption. However middleboxes may also change the TTL value, thus nullifying the effect. Finally, in this specific architecture, hypervisors can implement all the services, but the heaviest ones such as the Path Discovery and the Path Computation Services that shall be taken on by the Controller.

4.5 Cross-Layer MPTCP-LISP cooperation implementation

At the time of writing (2013), there was no open source TRILL implementation as there is today [248]. On the other hand, two open source LISP routers were available as well as a MPTCP linux kernel.

The purpose of the implementation concerns the establishment of additional subflows between MPTCP servers and users in a Cloud network where, at least at one side, path diversity is available at the IP layer, but not usable with native protocols. More precisely, we propose a specific cross-layer signaling to allow a MPTCP endpoint to profit from path diversity offered by a LISP [235] network. LISP can give hints to MPTCP about the Wide Area Network (WAN) paths diversity between two MPTCP endpoints. By allowing sharing of such an information among endpoints, we influence the number of subflows to create.

The current MPTCP path discovery does not explicitly limit the number of subflows to create, so current implementations create by default a mesh of subflows between two hosts' IPs. Most of the times, the more the subflows, the higher connection throughput, under appropriate congestion control. (note that there are also cases in which this default mechanism would prevent MPTCP from increasing the throughput, and cases where fewer subflows could provide the same gain by using fewer network resources). We target the specific case where more subflows could be created if intermediate multipath forwarding nodes (at the border between the local site and the WAN) can be used to split subflows over different WAN paths, under the hypothesis that the connection throughput is limited by WAN capacity rather than by LAN capacity.

In the following, we describe how the MPTCP path discovery can be augmented in this sense in a LISP network. Then we present the required signaling between MPTCP and LISP network elements, and the possible methods to perform the required multipath subflow forwarding.

in certain cases where the current MPTCP implementations would not obtain any gain.

4.5.1 Augmented Multipath TCP path discovery

The MPTCP specification - see the path management section in [211] - states that the path discovery mechanism should remain modular so that it can be changed with "no significant changes to the other functional components". We leverage on this specific part of the MPTCP protocol to define an augmented subflow discovery module taking profit of LISP capabilities, to augment MPTCP performance while preserving endhost resources.

Figure 4.4 presents a Cloud access situation where MPTCP could perform better if its path discovery module were informed of LISP path diversity and forwarding capability when creating subflows. In the example situation, there is one IP on the client host and one IP on the server; as such, the legacy MPTCP mechanism creates only one subflow. Under the assumption that commonly connection throughput is limited by WAN capacity rather than by LAN capacity, limiting to a single subflow prevents from benefiting of the WAN path diversity and the likely greater throughput achievable if forms of multipath forwarding at intermediate nodes exist. A LISP network offers the signaling capabilities to retrieve path diversity information, and the switching mechanisms to ensure multipath forwarding. In the fig. 4.4 example, this is possible establishing two subflows instead of one, assuming each subflow is forwarded to a different IP transit path (guaranteed as explained next) thanks to the LISP-capable border router. It is worth highlighting that as of the specifications - and as implemented in the MPTCP Linux implementation [102] - different MPTCP subflows can share the same source IP provided that they use different TCP ports. This subflow identification mode should be used to create the additional LISP-enabled subflows.

More generally than the fig. 4.4 example, the number of MPTCP subflows can be significantly increased thanks to LISP capabilities in the case the endpoints dispose of multiple interfaces. We assume communications between hosts are strongly asymmetric, the largest volume being from server to client, so that the client-to-server flow essentially consists in acknowledgments. Let l_1 and l_2 be the number of interfaces of server endpoint (or of the hosting hypervisor in case of VM server) and client endpoints, respectively. A LISP site can be served by one or many LISP routers, each disposing of one or many RLOCs. Let L_1^r and L_2^r be the number of locators of the r^{th} LISP border router at site 1 (Server side) and 2 (Client side), respectively. Therefore, the maximum number of subflows that can be opened by

legacy MPTCP is $l_1 l_2$. Following the design choice to route only one subflow over one RLOC-to-RLOC inter-site path to avoid bottlenecks and other management issue in the WAN segment, the number of maximum number of subflows that could be opened thanks to LISP multipath forwarding is $(\sum_r L_1^r)(\sum_r L_2^r)$. Then we can distinguish two cases:

- if the LISP site network allows full reachability between endpoints and corresponding ITRs, and endpoint's traffic reaches any corresponding ITR in a non deterministic way, then the maximum number of subflows that shall be opened is:

$$N_a = \max\{l_1 l_2; (\sum_r L_1^r)(\sum_r L_2^r)\}. \tag{4.1}$$

  LISP-based augmentation would likely be effective only if the second term is higher than the first. The non deterministic interconnection between the endpoint and its ITRs can be due, for instance, to adaptive L1/L2/L3 traffic engineering changing routes and egress points from time to time even for a same application flow, or load-balancing such as equal-cost multipath L2/L3 routing so that different subflows may exit by different ITRs. It is worth highlighting that (4.1) is only a suggested upper bound; in the case the right term is the maximum, in such non deterministic situations there is no guarantee the N WAN paths will be used, especially if $l_1 < L_1$ or $l_2 < L_2$.

- if each of the endpoint's interfaces can have its traffic routed via one single ITR, then we can take the maximum profit from the LISP path diversity. The best situation is when $l_1 = L_1$ and $l_2 = L_2$; functionally, it is like if the endpoint's interfaces were virtualized in a number of interfaces equal to the number of the RLOCs of the ITR it is linked to. Such a setting is the de-facto setting when the user is a LISP mobile node MPTCP-capable user; for the Cloud server side, it is technically relatively easy to create a number of virtual interfaces and bind them to different egress routers via virtual LAN mechanisms and the like. In such configurations, the maximum number of subflows that shall be opened is:

$$N_b = l_1(\sum_r L_1^r)(\sum_r L_2^r)l_2 \tag{4.2}$$

The situations described in the previous points can hold only at one site or even only partially within each site. (4.1) and (4.2) are indeed upper

bounds. Nevertheless, it does not hurt creating a number of subflows slightly higher than the number of paths, at the expense of a higher processing and signaling burden on the network. Therefore, (4.1) or (4.2) can be used to compute the number of subflows for the MPTCP-LISP augmentation, depending on the situation, upon appropriate customization of the path discovery module settings.

4.5.2 Signaling requirements and implementation aspects

From a mere signaling perspective, the requirement is therefore that the MPTCP endpoint be able to query the LISP mapping system to get the RLOCs of the other endpoint (the number of interfaces being already transported via MPTCP options). Standard LISP control-plane messages can natively assume this task (i.e., MAP-REQUEST and MAP-REPLY messages). Once this information is obtained, the number of subflows to be opened can be computed, and the MPTCP endpoint can either advertise these possible subflows to the other endpoint or initiate the required subflows following a procedure such as the one we propose hereafter.

Let us describe in higher detail the different steps, depicted in fig. 4.5, needed by the MPTCP host to retrieve the adequate number of local and remote RLOCs, allowing to compute the number of subflows to create. In our implementation, these communications are managed by a specific module at the kernel level. In the case of servers hosted in VMs, the MPTCP logic and our path discovery module could be implemented via an MPTCP proxy (see [249]) at the hypervisor level. In the following, we also give necessary hints on implementation aspects related to each step.

1. endpoint C (Client) first establishes an MPTCP connection with endpoint S (Server). Once endpoint S is assessed as MPTCP compliant, the path discovery process can start.

2. The kernel of endpoint C calls a function of the previously loaded MPTCP kernel module, with as parameters the current MPTCP connection identifier and the EID we want to find the RLOCs associated to.

3. Upon reception, the kernel module translates it into a Netlink message and sends it to the MPTCP Netlink multicast group (Netlink is the

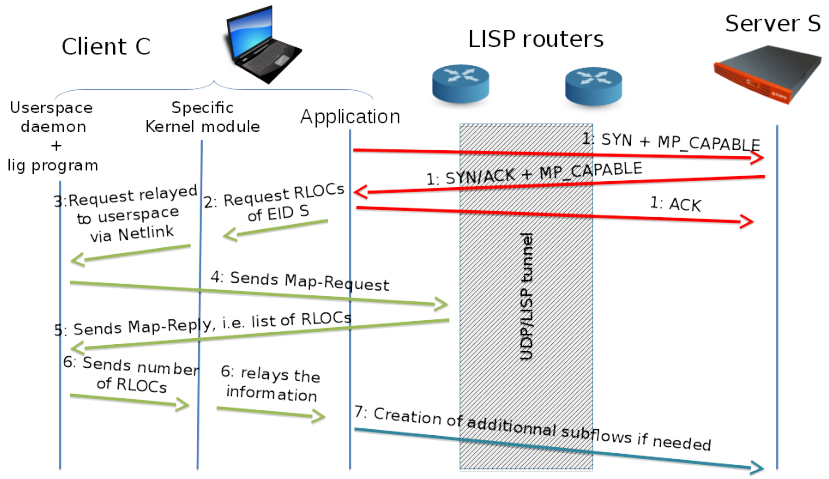

Figure 4.5: Signaling process chronology.

networking protocol used by the linux kernel space to communicate with user space and vice-versa).

4. A specific daemon in the user space is registered with the MPTCP netlink group. Upon reception of the Netlink message, it sends a Map-Request to the LISP router C, i.e., the daemon asks for the RLOCs responsible for the EID (i.e., the IP of Server S). Note that normally one should send a Map-Request to a Map Resolver, which in existing vendors implementations can be collocated in a LISP router; in our implementation, we modified an open source LISP router for this purpose. Also note that an ad-hoc DC network controller could do this job too.

5. The LISP router C queries its configured map-resolver if the answer is not in its cache. It then sends to that user space daemon a Map-Reply listing RLOCs responsible for the requested EID, thus retrieving the number of RLOCs.

6. Upon reception of the Map-Reply, the userspace daemon forwards the answer via Netlink to the kernel module.

7. Finally the module relays the information to the kernel, triggering the creation of new TCP subflows.

It is worth noting that the described process is expected to be efficient only for long-lived flows. Indeed, on the one hand short flows may finish before the procedure; on the other hand, the different requests sent from the hosts to the routers consume bandwidth and add an avoidable load to the router. Caching associations between endpoint identifiers and the number of subflows to create mappings in the source host could alleviate signaling traffic. To avoid applying our proposed cross-layer interaction to any flow, a system that triggers the procedure only for long-lived flows would be very reasonable, for example based on a whitelist or a dynamic recognition system such as the one described in [242]. In the case of servers hosted in a VM, this could be implemented at the hypervisor level.

4.5.3 LISP multipath forwarding requirements

In order to ensure that subflows indeed follow different WAN paths, it is possible to implement both stateful and stateless load balancing mechanisms.

- A possible stateful solution would be to resort to the LISP-Traffic Engineering (LISP-TE) feature described in [246]. This feature allows enforcing an "Explicit Locator Path (ELP)" via the LISP control-plane, i.e., with little or no impact on data-plane packet crafting. For a given EID, the mapping systems can build a LISP overlay using intermediate LISP routers between the ITR and the ETR, as RTRs), which decapsulate the incoming LISP packet and re-encapsulate it toward the next RTR (or the ETR) in a xTR path. In the context of all subflows being addressed to the same destination EID, the usage of options in the packet header (e.g., using the LISP Canonical Address Format, LCAF, features [239]) can allow routing concurrently multiple subflows via different xTR paths. However, in the case LISP is implemented at the hypervisor level, hence without touching the source host (VM), the LISP-TE encapsulation with LCAF options can be defined as a switching rule in virtual switches such as OpenVSwitch – knowing that OpenVSwitch already implements basic LISP encapsulation as switching rule since a few months[3].

- A stateless solution could be to carefully choose IP or TCP header fields so that IP packets follow a foreseen IP path. The computation

[3]See: `http://openvswitch.org/pipermail/git/2013-February/003666.html`.

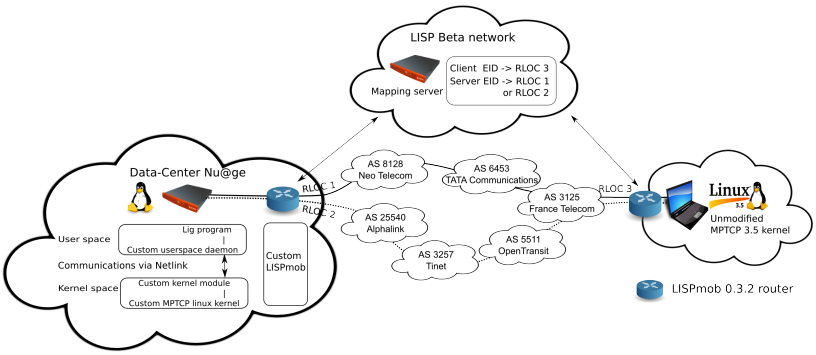

Figure 4.6: Cloud network test bed scenario.

of these fields (for instance the TTL value in IP packets or source port in TCP packets) require the knowledge of both the topology and the load-balancing algorithms running in the nodes to be able to correctly choose the path. Nodes usually use an hash function over IP and some TCP fields to compute a unique flow identifier that decides the egress interface. Hashing functions, by construction, make it hard to foresee the egress interface of a packet. On the contrary, invertible functions can prove helpful as explained in [250].

In our implementation detailed hereafter, we opted for the second option as the first is not yet available in any LISP implementation to date. At the last step in the previous subsection, subflows source ports are chosen so that their port number modulo the number of disjoint paths are all different. The LISP router can then deterministically route each subflow to a specific path. For instance, in the case of 2 paths, we need to have one subflow whose (source + destination port number) % 2 equals 0 and the second subflow (source + destination port number) % 2 equals 1. This mechanism could be replaced with a more scalable solution as those described in [250], yet it was easier to implement. The load balancing is fully effective when installed on both remote and local sites (as in the experiment). Still, in case communications are asymmetric, it can perform well even if installed only on the side sending data (in this case the server side).

It is worth mentioning that other strategies could be adopted according

to the level of control we have on the different nodes. In case only the DC-side is under control, the server could advertise the same IP until the client creates subflows with adequate source ports. Or it could accept enough communications to ensure that some of them will go through different paths; the MPTCP congestion mechanism would then send data only on the best subflows, which will likely follow different paths.

The stateless approach is not limited to L3 routing; it can also apply to L2 routing protocols disposing of additional fields (e.g., the hop-count) such as the Transparent Interconnection of a Lot of Links (TRILL) [236] and Shortest Path Bridging (SPB) [251] protocols.

In DC controlled environments, for packet-based intra-DC communications, the computation of the different fields could be left to an advanced layer-2 fabrics allowing DC traffic engineering such as TRILL, SPB or OpenFlow. As for the extra-DC segment, stateless solutions do not allow enough control to choose paths. The LISP-TE features could thus prove helpful since it explicitly encodes some LISP nodes to go through, but requires the usage of a wide enough WAN operational LISP network to prevent packets from making a prejudicial detour because they have to go through LISP routers.

4.5.4 Experimental results

In this section, we report and discuss the results obtained performing several MPTCP communications over the experimental test bed. First, we present the test bed setting and the developed node features we publish as open source code, then we assess the achievable gain with our solution, and finally we qualify the data-plane overhead of the cross-layer interaction.

Network test bed

Let us illustrate the experimental test bed we used for the experimentation of our solution, displayed in fig. 4.6. It implements our basic reference augmented TCP scenario described in fig. 4.4. We used a MPTCP-capable virtual machine hosted in the Paris-Est DC of the French NU@GE project [252], disposing of two Internet Service Providers, Neo Telecom (AS 8218) and Alphalink (AS 25540). On the other side, we use a single-homed and MPTCP-capable Cloud user. The Internet-working between the VM and user is such that two disjoint Internet paths are used down to the user's ISPs, and such

that the two intra-DC paths between the VM's hypervisor and the DC border LISP router are disjoint ones.

In such a configuration, the highest improvements can be reached when the Cloud user's provider access and transit links are not congested. The test bed scenario can be considered as quite representative for real cases, where typically Cloud users are not multihomed and the DC has a few ISPs. Moreover, by targeting the more basic configuration with only two subflows we can more precisely demonstrate the benefit of our LISP-MPTCP cross-layer cooperation solution.

Open Source Nodes

In terms of open source software, we used the open source MPTCP Linux implementation [102] and the LISPmob [253] implementation (preferred over the BSD OpenLISP router [254], [255] because more easily customizable as it runs in userspace). We then applied these necessary modifications to the open source nodes. We published the patches in an open source repository [226].

- to the MPTCP kernel, to add our path discovery feature that retrieves the number of local and remote RLOCs for each destination it is talking to as previously described;
- to the LISPmob router so that: (i) it acts as a Mapping Resolver too; (ii) it balances the two subflows deterministically between its two RLOCs. For (ii), instead of resorting to a hash of the tuple (source IP, destination IP, Source port, Destination Port, TTL), we replaced the load balancing mechanism by a single modulo (number of remote RLOCs) on the TCP source port and force MPTCP to start TCP subflows with certain source port numbers.

To ease the development and to decorrelate the path discovery mechanism from other MPTCP mechanisms a stated in [183], we minimized the modifications to the kernel and instead created a kernel module called by the kernel each time MPTCP establishes a new connection. We also integrated an existing user space program named LIG (LISP Internet Groper, [256], [257]) to request EID-to-RLOC mappings (equivalent of DIG but for LISP networks), using MAP-REQUEST and MAP-REPLY control-plane messages. The LISP nodes were connected to the LISP Beta Network test

bed [258]. In order to retrieve a list of RLOCs responsible for an EID (and implicitly their number), we created a user space python daemon listening to requests transmitted by the kernel module, which then interacts with the LIG program. In the following, we do not differentiate between the two user space programs since it is just a matter of implementation.

Transfer times

In order to demonstrate the previously described architecture, we worked on the same topology as in fig. 4.4. A single homed client (with one link to AS3215) downloads 30 files from a single homed server (one link to a LISP software router) in a dual-homed data center (both links active, one towards AS25540, another one towards AS6453). Each transferred file is set bigger than the previous one by an increment of 256KB to assess the performance for different volumes. We record 20 transfer completion times for each file. We set *net.ipv4.tcp_no_metrics_save* to true so that linux discards tcp metrics at the end of experiment rather than caching them to improve throughput. We repeat this whole procedure using four different configurations:

- legacy TCP: with no cooperation with LISP, and a single TCP flow.
- MPTCP: with no cooperation with LISP, and a single MPTCP subflow.
- LISP + MPTCP: with cross-layer cooperation, creating as many subflows as the product of remote and local RLOCs, i.e., 2 subflows.
- LISP + MPTCP: we manually override the daemon result to create 3 subflows instead of 2

One can reasonable expect the cases 1 and 2 to be very similar in the provided configuration since MPTCP should use one flow only. During our tests, the server upload speed could fill the client download speed (8 Mbit/sec, corresponds to RLOC 3 in Figure 4.6) with a single TCP connection. In order to exhibit an increase in throughput via the use of several subflows, we limited RLOC 1 and 2 throughput via the linux QoS utilities so that each each RLOC could send no more then 4Mbit/sec. On fig. 4.7, we see as expected that unassisted MPTCP (i.e., MPTCP without LISP support, marked 'MPTCP') and TCP transfer times are quite close, MPTCP being a

little slower than TCP; the cause being the additional 20 bytes overhead per packet introduced by MPTCP. More importantly, we can see that our solution (marked 'LISP+MPTCP') performs significantly better. For almost all file sizes, we get a gain close to 90%, i.e., with the additional LISP-enabled subflow we can nearly halve the transfer time. This very clearly shows the important benefit we can reach with the MPTCP-LISP cross-layer cooperation module we propose.

We also get an interesting result by forcing the creation of three MPTCP subflows even if there are only two RLOCs at the server side. We notice that the throughput is in average worse than for 2 subflows, but it sometimes does perform better than with 2 subflows only. The decreased throughput makes sense as having three subflows going out 2 RLOCs means at least 2 of the subflows compete for bandwidth on some part of the WAN. The improvement is likely due to some further load-balancing occurring on the WAN, hence not as deterministic as in the two subflows case.

It is worth noting that due to the use of linux QoS utilities to cap the bandwidth, it would induce from times to times some latency (the RTT changing between 60ms to 300ms on each path) during the transfers without much impact on MPTCP.

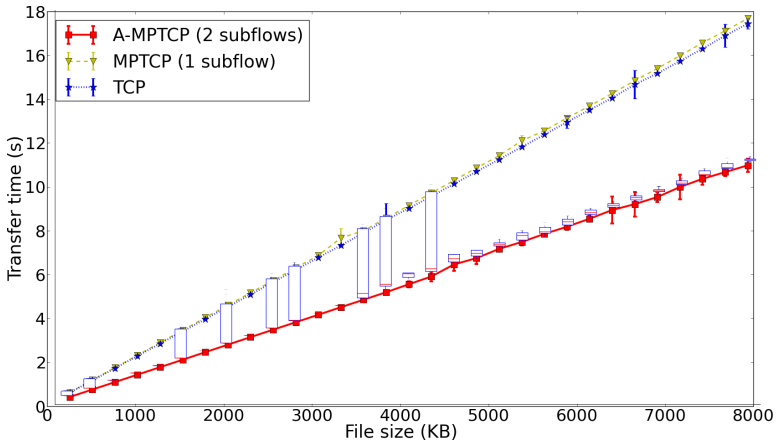

Figure 4.7: Completion times for different file sizes.

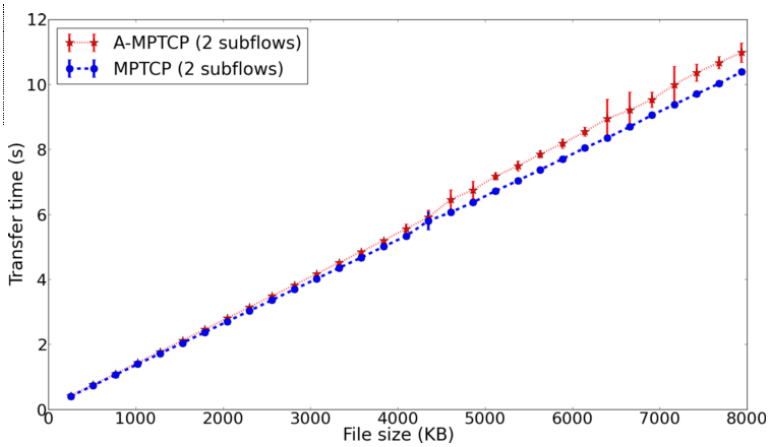

Figure 4.8: Transfer times for two subflows with and without LISP.

Data-plane overhead

From a data-plane forwarding perspective, we recall that LISP performs an IP-in-IP encapsulation with shim UDP and LISP headers. In order to get an idea of the LISP overhead, we recorded the transfer times in two cases. A first case with two MPTCP subflows enabled via LISP, and a second case with the same two MPTCP subflows, but manually configured, without LISP, hence avoiding data-plane encapsulation overheads. The results are shown in fig. 4.8. At a first look one can immediately notice that the overhead is most of the time negligible. It is worth noting that neither our connection or LISPmob allowed us to test higher rates (Nevertheless, we suspect that at higher rates, we might see a more important processing overhead of LISP traffic since the UDP encapsulation prevents the system from offloading TCP segmentation to the NIC).

4.5.5 TRILL and LISP unification for distributed DC networking

The master student Roua Touhiri undertook as part of her internship to partially unify the LISP and TRILL control planes. The proposed setup is a usecase of interest for the hosting company Alphalink[4], several DC are connected via a TRILL network overlay. The different DCs are accessible

[4] `www.alphalink.fr`

via LISP ETRs as well. One fundamental hypothesis is that the inter-DC latencies dominate over the DC access latency. As a consequence, the LISP egress point is of utmost importance to improve the QoS. As TRILL has the information about the internal overlay metrics, it can communicate them to the LISP control place, which in turn can advertise these metrics as LISP priorities to influence the user traffic. In [259], the authors implement an agent installed at the hypervisor level so that it can monitor a change in TRILL metrics due to a link failure or on the contrary generate a metric update because of a VM migration. In both cases, the agent computes the new cost from the LISP router to the hypervisor and propagates this information to the xTR. The xTR must then update the MSs so that user of this VM can be notified of the new optimal ETR and use it. The evaluation of the agent during a network impairment resulted in a 13 fold improvement in latency compared to the vanilla solution (absence of cooperation between LISP and TRILL); with a third quartile from 26,9ms to 2 ms overall latency.

4.6 Summary

Multipath communications remain a complex networking research field due to the necessary coordination between network layers and protocols. The architecture we propose unifies the different control planes thanks to the controller knowledge and coordinated routing mechanisms. Multipath communications are becoming essential to augment Cloud communications; for instance, very recent experiments have shown that significant throughput could be achieved either locally, a local MPTCP throughput of 51.8Gbit/s between 2 servers containing each 3 dual-port 10 Gig NICs has recently been proved [227] or on long distancesÂă: an intercontinental testbed [260] achieved a throughput of 15Gbits/s from Geneva to Salt Lake City. Our cross-layer multipath architecture can potentially overcome these values so as to further push network innovation at Internet edges.

Indeed, coordination between the MPTCP, TRILL and LISP protocols, splitting flows at the transport, network and Ethernet layers, can allow carefully selecting communication paths while controlling the computational load and guaranteeing a semi-distributed reliable nature to the communication environment. At the same time, the presented coordination can also lead to the Braess paradox, i.e., partially optimal routing may lead to worse

overall network performance [261] so some form of formal analysis must be undertaken to ensure the validity of the cooperation.

We have shown that MPTCP can achieve better performance thanks to a better knowledge of the underlying IP topology gathered via LISP protocol in a LISP network.

Our experimentations on a real large-scale test bed involving one data-center network show that the throughput, hence the transfer time, can be greatly improved, thanks to the cross-layer MPTCP-LISP protocol cooperation. We show that with just one additional LISP-enabled subflow, a transport-layer connection spanning the Internet via disjoint Autonomous System (AS) paths can terminate file transfers twice faster. It is therefore reasonable to generalize these results stating that, in absence of middleboxes filtering the MPTCP signaling, the achievable gain is directly proportional to the number of additional LISP-enabled subflows. Hence the higher the multi-homing degree of Cloud clients and data-centers, the higher the achievable gain.

While our path manager only supported the connection events, a similar but more comprehensive Netlink approach with the support for more MPTCP events was published in [262] and is used by Tessares[5]. The same authors also concurrently implemented a complementary MPTCP API [263]. The API allows the application to have full control but the Netlink module remains a valid solutions to enforce policies on applications that have not been upgraded yet.

Ironically, the existence of middleboxes heavily influenced the design of MPTCP but the mainstream industrial approach to MPTCP is to propose MPTCP proxies as it is done at OVH[6], tessares[7] and visible in the IETF drafts [264]. The advantage of the middlebox is that it usually has a better knowledge of the path diversity and does not require the specific signaling we used in this chapter. Relying on Dynamic Host Configuration Protocol (DHCP) as is proposed in Boucadair, Jacquenet, and Reddy [265] is a lightweight way to convey the source path diversity but may need more tweaking to take into account destination diversity as we did.

[5] `www.tessares.net`

[6] `https://www.ovhtelecom.fr/overthebox/`

[7] `www.tessares.net`

Chapter 5

Pacing on multiple paths to estimate difference in One-Way Delays

In this chapter, we propose sender-only modifications to obtain an estimation of the difference in OWDs between different paths, motivated by its utility for multipath transport protocols such as MPTCP and SCTP[1]. A high-level description of the estimator is as follows: the sender sends a series of MPTCP packets according to a specific schedule; if there are no losses, the sender should receive the matching acknowledgments, it can thus analyze the acknowledgments pattern to deduce a difference in path delay.

5.1 Introduction

Accurate estimations of OWD in networks could improve network utilization and in particular user's QoE: applications such as voice-over-IP or video streaming depend more on the forward time delay than the reverse one, for example.

As another example, the TCP Vegas [267] family of congestion control algorithms rely on RTT inflation to detect congestion but is unable to distinguish between forward or reverse path inflation. Including congestion control also on ACKs and not only on data packets could improve TCP performance as noted by Aylene and Weigle [268]. and [269]. In those experiments, dis-

[1]The contents presented in this chapter have been published in [266].

abling delayed ACKs resulted in an increased number of ACKs, that entered into competition for buffer space, incurring more ACK loss and a degraded throughput. In the case of multipath transport protocols, being informed about OWDs could permit sending ACKs over the best path and hence lead to finer latency sensibility and higher performance.

In particular, throughput increase, though an important driver for multipath transport protocols, is harder to achieve than it seems. Indeed, sending packets along heterogeneous paths (e.g., paths that differ in latencies, loss rates, etc) can result in head of line blocking, which decreases throughput, to the extent that sometimes MPTCP, e.g., would achieve even less than a legacy TCP connection [5], [209]. The OWD knowledge - or rather the ΔOWD between subflows - is then even more crucial in multipath communications than in single path communications, knowing the forward ΔOWD allows to reduce in-arrival packet disorder, thus reducing buffer requirements or increasing throughput. In Kaspar [270], taking into account ΔOWD in a multipath UDP scheduler leads to a 30% increase in throughput. A recent MPTCP scheduler [271] achieves a similar gain via modifying both the sender and the receiver.

As for the reverse ΔOWD, knowing which is the fastest reverse path lets the multipath transport protocol the possibility to acknowledge packets on the fastest path. Coupled with non-renegable selective acknowledgments, this allows to free sooner the send buffer in order to send fresh data faster [272].

While the benefits of knowing ΔOWDs look interesting, propositions to retrieve the information rely either on clock synchronization or some form of cooperation, none of them being standardized. With these constraints in mind, we elaborate an alternative to estimate the difference in OWDs between subflows of two end hosts.

The rest of the chapter is organized as follows. Section 5.2 describes related work on OWD estimation. Section 5.3 precisely describes our solution. Finally, in Section 5.4 we expose and discuss simulation results.

5.2 Related work

In this section we present OWD and ΔOWD online estimation techniques for the single path case first, followed by techniques specific to multipath

protocols. Offline OWD estimation techniques (i.e., where transit times are corrected a posteriori, as in Paxson [273]) also exist but they require cooperation from the network, or clock synchronization between end hosts. Thus we only consider in the following online techniques, able to dynamically adjust scheduling parameters.

From a practical point of view, the presented techniques may better apply to Wide Area Network (WAN) communications than to Local Area Network (LAN) communications (or intra-datacenter communications) as LAN smaller RTTs require more precision. It is also worth noting that until the linux kernel 3.15, the linux network stack resolution was 125 µs, while it is now set to one µs [2].

5.2.1 Clock synchronization in packet switched networks

The dissemination of clock synchronization information in packet-switching networks can be achieved through different protocols, with different precision levels. The most adopted solutions being the Global Positioning System (GPS) [274], the Precision Time Protocol (PTP) [275] and the Network Time Protocol (NTP) [276].

A GPS terminal can infer the clock by correlating the positioning signals from a constellation of satellites, each satellite embarking several atomic clocks, reaching a precision skew of a few nanoseconds only. GPS receivers are precise but expansive, thus not all computers can be equipped with one: time has to be distributed.

PTP [275] is able to distribute this time in packet-switched networks through continuous offset correction between a grandmaster clock and hierarchy of master-slave clocks. With sub-microsecond accuracy, this approach allows reaching very high precision, suitable for 4G base station synchronization.

Similarly NTP relies on a hierarchy of clocks, the higher the stratum, the lower the precision, which is in the order of a milliseconds at the endhost.

NTP assumes that OWDs are half the RTT, which is an approximation considered as too coarse by many studies [277], [278]. As noted in Paxson [279], when applied to Internet connections, this approximation may lead to noticeable errors as the Internet does not guarantee symmetric routing.

[2] `https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=740b0f1841f6e39085b711d41db9ffb07198682b`.

Even when routing is symmetric, there can still be noticeable differences between OWDs: because directions may have different characteristics (loss rates, capacities) due to policies (e.g., different resource reservation levels in cellular networks) or physical constraints (e.g., asymmetric bandwidth in ADSL networks).

5.2.2 TCP variations

Some variations of TCP such as TCP Vegas [267], FAST TCP [280] try to infer the congestion level of the network from the evolution of the RTT. TCP Illinois and TCP Compound[281], [282] are hybrid approaches taking into account both the delay and losses.

Two ways exist to retrieve the RTT for a TCP sender:

1. Compute the time needed to send a full window and receive the matching acknowledgments.

2. Use the TCP Timestamp option [283].

The TCP timestamp option can be used for RTT measurement. Its usage must be negotiated between the end-hosts during the connection establishment. Once negotiated, each host records the time at which it created a packet into this very packet. Upon reception of that packet, the receiver puts its own timestamp along the received one and returns the packet. Upon reception of the acknowledgment, the TCP sender retrieves the timestamp it previously sent and subtracts its value to the current node time.

There is little one host can deduce from the remote host timestamps since the standard only guarantees the remote TCP clock to be 'monotone-increasing' [283]. Alternatively, propositions exist to negotiate the clock skew during the connection establishment, as presented in [284]. This would allow each host to interpret the timestamp of the remote host. Knowledge of the receiver skew allows the sender to correct the remote timestamp so that the duration matches its local skew. It also gives an indication on the possible precision. The major drawback of TCP variations relying on RTT estimations is that TCP takes a coarse decision based on the maximum level of congestion between the forward and backward paths.

Choi and Yoo [285] modify the roles of the fields in the timestamp option: timestamps are replaced with the actual RTT values measured by the different nodes. Providing both hosts relayed RTTs to their peer since the

beginning of the communication, the OWDs can be computed analytically with a precision depending on the value guessed for the first OWD. Gurewitz and Sidi [286] take another approach that requires cooperation from the network for the forwarding aspect, but no clock synchronization. To remove the uncertainty about an untrusted remote clock, a node sends probes through different paths in the network (there must be some mechanism to enforce these paths, e.g., source routing, traffic engineering) that must come back to the initial sender. The different values obtained through the use of the probes add constraints over the values of the different OWDs. When enough constraints have been harvested to solve the system of equations, the algorithm tries to determine the OWDs that yield the least square error.

Some studies [287]–[289] show that there can be a low correlation between delay and losses which can raise some skepticism about the use of delay for congestion control. These concerns are balanced with other real experimentation McCullagh and Leith [290] or the recent TCP BBR from Google [291] which seems to obtain good results [3] even for aggregated flows.

5.2.3 Multipath control techniques

For a multipath transport protocol such as MPTCP, multiple connection 'subflows' can be opened between the end-hosts, regardless of the number of interfaces. Having an estimation of the difference between subflow OWDs (noted as $\Delta_F OWD$ for the forward delta OWDs, and $\Delta_B OWD$ for the backward delta OWDs in the rest of the article) would already allow some improvements. Moreover, in case of low traffic, this knowledge should be needed by delay-sensitive applications to send packets on the fastest path (provided loss ratios are similar).

As for the $\Delta^B OWD$, the faster an ACK reaches the sender, the faster the sender can free space in its send buffer and send new data packets. The knowledge of the fastest reverse path would thus let a multipath transport protocol such as MPTCP the possibility to acknowledge packets on the fastest path, as explained in Ribeiro and Leung [292].

Relying on [284], authors in [293] assume clock skew synchronization based on the TCP timestamp option. The sender, instead of discarding receiver timestamps, saves them; in this way, it can deduce the difference in arrival time at the receiver between packets that followed different paths.

[3] `http://blog.cerowrt.org/post/bbrs_basic_beauty/`.

The difference in backward and forward delay can easily be deduced afterwards by the sender.

In [294], Zhou et al. propose to deduce the OWDs of two subflows from the size of the receiver input buffer. The upside is that it does not require any receiver modification or network cooperation but it assumes constant transmission rates and no loss. It also requires several measurements to get an accurate value.

5.3 Proposed OWD estimator

The primary objective of our estimator is to provide a practical estimation of forward $\Delta OWDs$ to a multipath transport protocol. The desirable OWD estimator shall thus:

- not require clock synchronization nor network cooperation; timing should be done on the sender's clock exclusively;
- be able to deliver an estimation within a few RTTs since most connections are short-lived.

Briefly, the OWD estimator algorithm we propose is such that once it judges its $\tilde{\Delta}^F OWD$ estimation as accurate enough, it is able to identify the backward slow path and provide improved estimations of OWDs, as compared to halving the RTT.

In the following, we first present a few notations related to our delay model, then we present our algorithm.

5.3.1 Delay model

This section exposes a few notations for a baseline 2-subflow case. We make no assumption about the number of interfaces of the host (mono or multi-homed) or the physical paths followed by the subflows (i.e., partially or fully disjoint). The disjoint case seems popular though, as it is likely to exhibit a higher difference between the OWDs, as it is the case for smartphones with both cellular and wifi interfaces.

Let $i \in \{1, 2\}$ be the subflow index, f_i and r_i be respectively the forward and reverse transfer delay on subflow i.

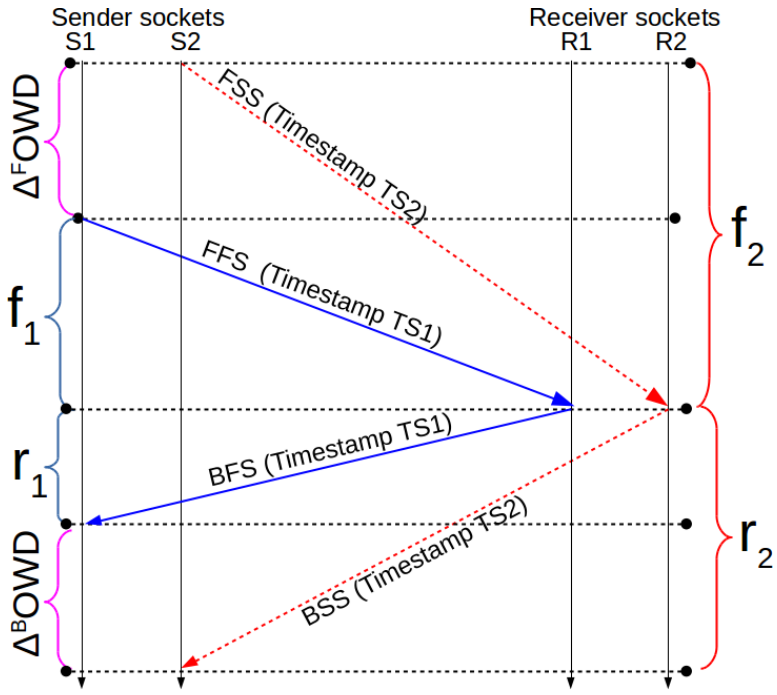

Figure 5.1: Illustration of used notations for two subflows.

The following notations are illustrated in Figure 5.1, let the OWD of the Forward Fast Subflow (FFS), the Forward Slow Subflow (FSS), the Backward Fast Subflow (BFS) and the Backward Slow Subflow (BSS) be respectively:

$$f_{FFS} = \min_i(f_i) \tag{5.1}$$

$$f_{FSS} = \max_i(f_i) \tag{5.2}$$

$$r_{BFS} = \min_i(r_i) \tag{5.3}$$

$$r_{BSS} = \max_i(r_i) \tag{5.4}$$

It is worth noting that the Forward Fast Subflow (FFS) can be a BSS, i.e., a subflow is not necessary the shortest in both forward and backward directions. Obviously, the RTT of subflow i can be computed as:

$$RTT_i = f_i + \text{processing delay} + r_i \tag{5.5}$$

f_i and r_i can be decomposed into a deterministic and a stochastic part.

The deterministic part corresponds to the time needed for a bit to propagate through its medium (supposing the route does not change during the connection) while the stochastic part refers to generic queuing delays.

The queuing delay is the sum of several queuing events that can stem from network queuing but also the host processing delay. For the algorithm to work better, all source of noise in the measurement should be avoided. By analogy with TCP, this means that the Nagle algorithm [295] (which prevents TCP from sending many small packets) and delayed ACKs [296] (TCP receiver waits a certain amount of received packets or a timeout before acknowledging packets) should be disabled to keep the processing delay negligible. Likewise, TCP Segmentation Offload (TSO) can add noise to the measurement with no relation to the congestion level. TSO is atechnique where the network stack hands over huge segments to the NIC so that NIC converts these into smaller packets. In general the NIC waits for a timeout before forwarding these packets, which results in increased burstiness and queuing delay. The sending buffer should also be empty for the same reason, except if hardware timestamping is in use. Duplicate acknowledgments caused by probes should not trigger retransmissions either (it is already true in MPTCP). Packet loss of any packet during a round can be detected and results from the round dismissed. The forward and backward delta delay are then defined as:

$$\Delta^F OWD = f_{FSS} - f_{FFS} \tag{5.6}$$

$$\Delta^B OWD = r_{BSS} - r_{BFS} \tag{5.7}$$

Both values are computed so that they are not negative.

The sender can easily retrieve the RTT, but it does not know f_i or r_i and traditionally assumes that $f_i = r_i = RTT_i/2$. In the following paragraph, we describe our estimator algorithm allowing to alleviate this issue.

5.3.2 Algorithm

As mentioned above, we aim to rely only on the sender's clock and require no cooperation with network elements. Before describing the algorithm, we need to highlight the following assumptions:

1. The clock precision is at least one magnitude higher than the maximum RTT.

2. The hosts can send packets at a locally precise determined time.

3. The server acknowledges every received packet immediately.

4. Timestamps can be embedded into packets.

1) ensures that we can measure accurately enough the time we want to estimate. 2) may be more or less feasible, but is required to pace packet emission. 3) is assumed for convenience, but the algorithm could work with this additional constraint at the expense of precision. 4) ensures the host can identify each probe, to detect and discard rounds where probes arrive in disorder or are lost.

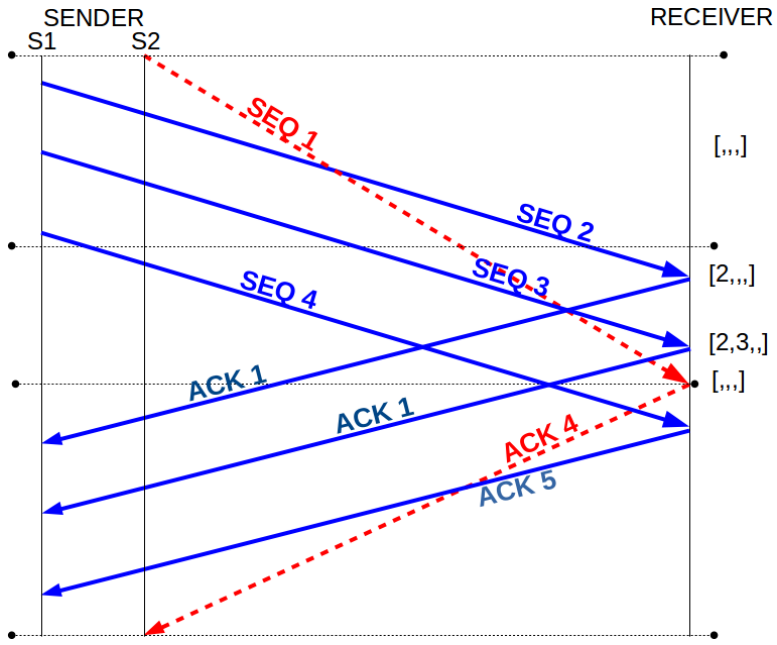

Figure 5.2: An example of Head-of-Line blocking.

The first objective of the algorithm is to compute $\tilde{\Delta}^F OWD$, then it can compute a reverse $\tilde{\Delta}^B OWD$ value and finally provide an estimation for OWDs. The key idea to deduce the $\Delta^F OWD$ is to forcefully create and detect head of line blocking at the receiver from the sender side, as depicted in fig. 5.2 (the receive buffer is represented between square brackets). This is done by sending a packet on the FSS, the sender then waits for the currently estimated $\tilde{\Delta}^F OWD$ and sends a train of probes on the FFS until these

probes match either the head of line blocking pattern in fig. 5.5 or that in fig. 5.6, i.e. packets sent on the FFS frame the packet arrival on the FSS. Every probe embed its own unique timestamp in order to identify itself.

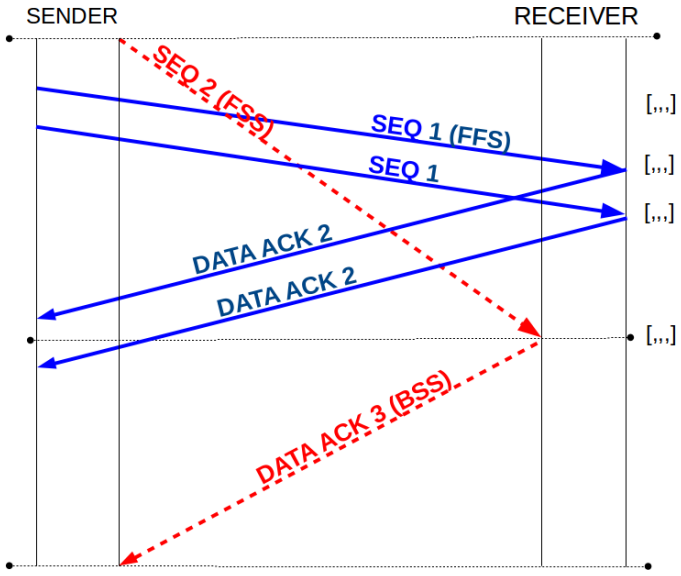

Figure 5.3: Early probing: $\tilde{\Delta}^F OWD$ is too low.

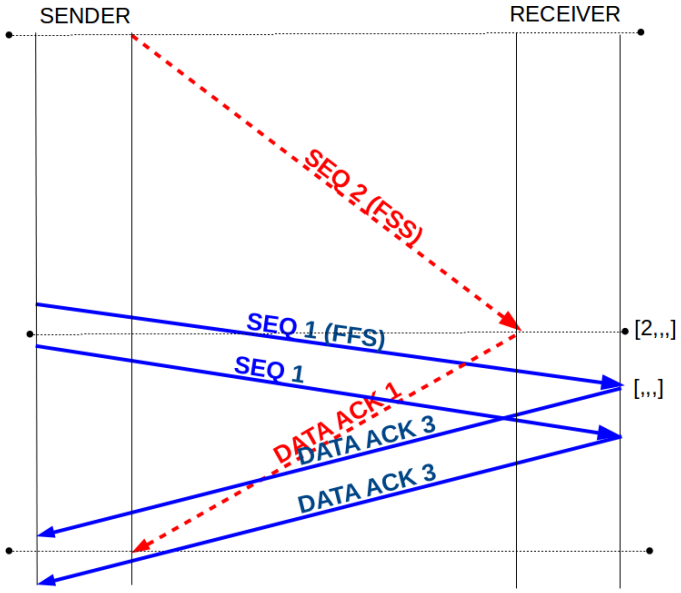

Figure 5.4: Late probing: $\tilde{\Delta}^F OWD$ is too high.

More precisely, the algorithm runs in two steps:

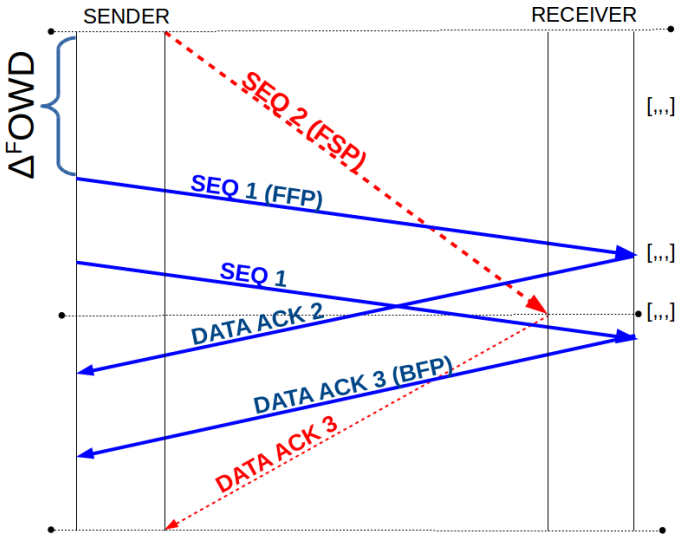

Figure 5.5: Identical forward and backward fast subflows.
The algorithm has converged to a valid $\tilde{\Delta}^{F}OWD$.

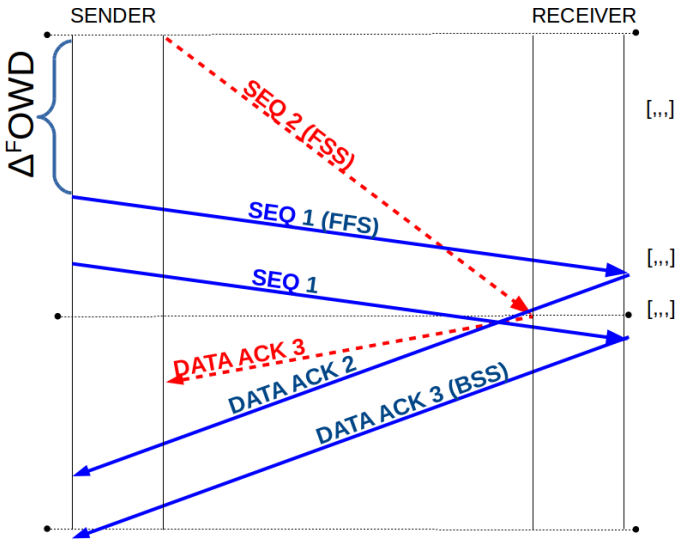

Figure 5.6: Different forward and backward fast subflows.
The algorithm has converged to a valid $\tilde{\Delta}^{F}OWD$.

1. the sender needs to determine which subflow is the FFS. To do so, it sends two packets (one on each subflow) with consecutive sequence numbers at the same time. The server acknowledges each packet with the highest in-order sequence number. Thus it is possible for the sender to deduce from the ACKs which packet arrived first at the remote host. Note that the FFS can be the BSS, i.e., the first packet to come back to the sender is not necessarily the one that arrived first at the remote node. This step is quite straightforward and can be obtained in one RTT as described in the algorithm 1 line 2.

2. the sender then computes $\tilde{\Delta}^F OWD$. This is an iterative process that can run as long as FFS and FSS remain the same, e.g., the algorithm could restart if RTT varies too much. To achieve this, the sender sends a packet with sequence number $X + 1$ on the FSS, then sends probes (two for instance in Figure 5.3) with sequence number X on the FFS with a delay close to the current estimation of $\Delta_F OWD$. In order to increase the change of getting proper values, one could choose to send probes when the application does not send any traffic, or as in [291], pace the traffic beforehand to free some space in the network queues. There are four different patterns of packet arrival order at the receiver Depending on the case, the current $\tilde{\Delta}^F OWD$ is updated accordingly:

   - On Figure 5.3, all probes arrive before the packet on the FSS. The sender can deduce it from the head of line blocking exhibited by the acknowledgments of the probes. If the packet on the slow subflow arrives before any of the probes on the FFS, then the server would acknowledge that probe with the sequence number three. The sender can deduce from this that the current delay before sending the probe is underestimated.
   - Contrary to the previous case, in Figure 5.4 all probes arrive after the packet on the FSS. The sender receives an out-of-order ACK on the FSS. This means that all probes arrived after the packet on the FSS, i.e., the current $\tilde{\Delta}^F OWD$ is too high and should be decreased.
   - In Figure 5.5 are the cases that allow to deduce more precisely the $\Delta^F OWD$, i.e., probes framing the arrival of the packet on the FSS. The sender identifies these cases when it receives successive

probes with different ACK numbers. To compute the forward delay of the FFS, we consider that the ACKs on each path were concurrently sent by the server as shown on the similar Figure 5.1 - so the estimations change as follows:

$$\tilde{f}_{FFS} = \frac{RTT_{FFS}}{2} \tag{5.8a}$$

$$\tilde{\Delta}^F OWD = TS2 - TS1 \tag{5.8b}$$

where TS2 and TS1 are the timestamps exposed in fig. 5.1. Reverse OWDs are then deduced from the RTTs as shown later.

- Figure 5.6 case differs from fig. 5.5 case in that the FFS is the BSS. To compute the forward delay of the FFS, we consider that the ACKs on each path were concurrently sent by the server so the FFS OWD estimation becomes:

$$\tilde{f}_{FFS} = \frac{RTT_{FSS} - \tilde{\Delta}^F OWD}{2} \tag{5.9}$$

To summarize, when probes frame the packet arrival on the FSS, the value $\tilde{\Delta}^F OWD$ may be considered as correct and allows to compute estimations for the following values:

$$\tilde{f}_{FFS} = \min(RTT_{FFS}, RTT_{FSS} - \tilde{\Delta}^F OWD) \tag{5.10}$$

$$\tilde{f}_{FSS} = \tilde{\Delta}^F OWD + \tilde{f}_{FFS} \tag{5.11}$$

$$\tilde{r}_{BFS} = RTT_{FFS} - \tilde{f}_{FFS} \tag{5.12}$$

$$\tilde{r}_{BSS} = RTT_{FSS} - \tilde{f}_{FSS} \tag{5.13}$$

5.4 Simulation results

We have chosen to implement the algorithm in a network simulator rather than doing real experiments because comparing delay estimations require very high precision values, we would have needed either synchronized clocks or complex tunneling to use the same host as client and server. Both are no easy task so we have implemented the algorithm in a slightly modified version of the NS-3 [297] (version 3.20). NS-3 is a well-maintained open-source event-driven packet-level network simulator.

Algorithm 1 Instance of the algorithm with three probes.

```
1: interval ← 3ms                                   ▷ Time interval between probes on FFS

2: procedure FindForwardFastSubflow(seqNb)          ▷ Returns a tuple (FFS, FSS)
3:     SendPacket(1, seqNb)
4:     SendPacket(2, seqNb + 1)
5:     Wait for both acks
6:     if (Ack received on path 0) = seqNb then return (1, 2)
7:     else return (2, 1)
8:     end if
9: end procedure

10: procedure SendPacket(pathId, seq)
11:     Send sequence number seq on path pathId
12: end procedure

13: procedure StartEstimationRound(FFS, FSS, deltaEstimation)
14:     SendPacket(FSS, roundLowestSeqNb + 1, 0)
15:     Wait for Max(deltaEstimation − interval, 0)   ▷ Wait for a positive duration
16:     SendPacket(FFS, roundLowestSeqNb)
17:     Wait for interval
18:     SendPacket(FFS, roundLowestSeqNb)
19:     Wait for interval
20:     SendPacket(FFS, roundLowestSeqNb)
21: end procedure

22: Begin
23:     roundLowestSeqNb ← 0
24:     Δ̃^f OWD = 0                                  ▷ Start with an estimation of 0
25:     FFS, FSS ← FindForwardFastSubflow(roundLowestSeqNb)
26:     while no drastic change in RTT do
27:         roundLowestSeqNb ← roundLowestSeqNb + 2
28:         StartEstimationRound(FFS, FSS, Δ̃^f OWD)
29:         Wait for all acknowledgments
                                  ▷ In case of losses, a timeout restarts a new round
30:         situation ← DeduceSituationFromAcks

31:         if situation = Figure 5.3 then                          ▷ Early probes
32:             Δ̃^f OWD ← Δ̃^f OWD + interval
33:         else if situation = Figure 5.4 then                     ▷ Late probes
34:             Δ̃^f OWD ← Δ̃^f OWD − interval
35:         else                          ▷ Probes on FFS framed packet arrival on FSS
36:             Δ̃^f OWD ← Average delay of the 2 framing probes
37:         end if
38:     end while
39:     Restart algorithm from the beginning
40: End
```

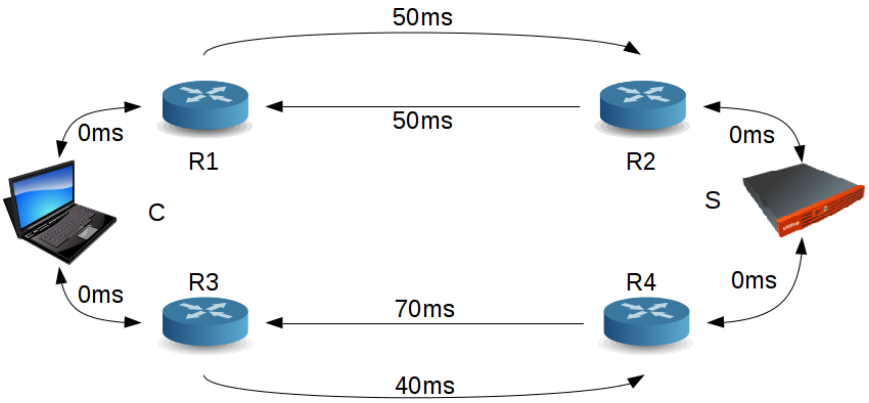

Figure 5.7: Test topology with asymmetric paths.

MPTCP being unavailable in vanilla NS-3, we simulated its behavior via two custom applications - client and server - over UDP. The client application sends timestamped packets with a sequence number over the different subflows. Upon reception, the server generates a reply containing the received timestamp, plus the server own timestamp and the highest in-order sequence number it received. The server timestamps allow the computation of the real OWDs (i.e. these values are used for plotting but are ignored by the algorithm).

On the Internet, the asymmetry in OWDs can originate from several reasons, asymmetric bandwidths or asymmetric propagation delays. We test both of them via binding one on/off TCP application on each client interface, to create jitter. The topology used is visible on fig. 5.7 and the queue size of the routers is set in packet unit. The source code of the simulation is available at [298].

5.4.1 Results

While fig. 5.8 displays $\tilde{\Delta}^F OWD$ at all times in order to better understand the estimation update process, it is worth noting that this estimation should be considered valid only when $\tilde{\Delta}^B OWD$ is plotted as well. This is particularly visible from rounds 40 to 50. Around round 40, we simulate a rerouting event by adding a 30ms delay from R1 to R2. The algorithm detects the situation in fig. 5.3 and reacts according to the algorithm 1 line 31 and thus increases the $\tilde{\Delta}^F OWD$. Notice that no backward estimation is plotted at that time, which means the algorithm has not converged yet

and that the current estimation sets only a minimum. In the range 40 to 50, where the FFS is different from the BFS, rounds last as much as $max(RTT_{FSS}, \tilde{\Delta}^F OWD + RTT_{FFS})$ which should be close to the RTT_1. Thus the algorithm converged to the new $\Delta^F OWD$ in ten RTTs. While our parameters adopted a conservative approach (starting with an estimation equal to zero, constant delay between probes), a dichotomic or cubic approach may allow for a faster convergence.

In fig. 5.9 the estimations are closer to the real OWD than halving the RTT - when they exist. For instance in round 60, the forward delay estimation on path one is 10ms more precise. Under low-jitter conditions, this should be true for all situations where the FFS is the same as the Backward Fast Subflow (BFS), else it depends on the per-direction delay difference between subflows. The more difference there is, the more gain the algorithm should exhibit compared to halving the RTT.

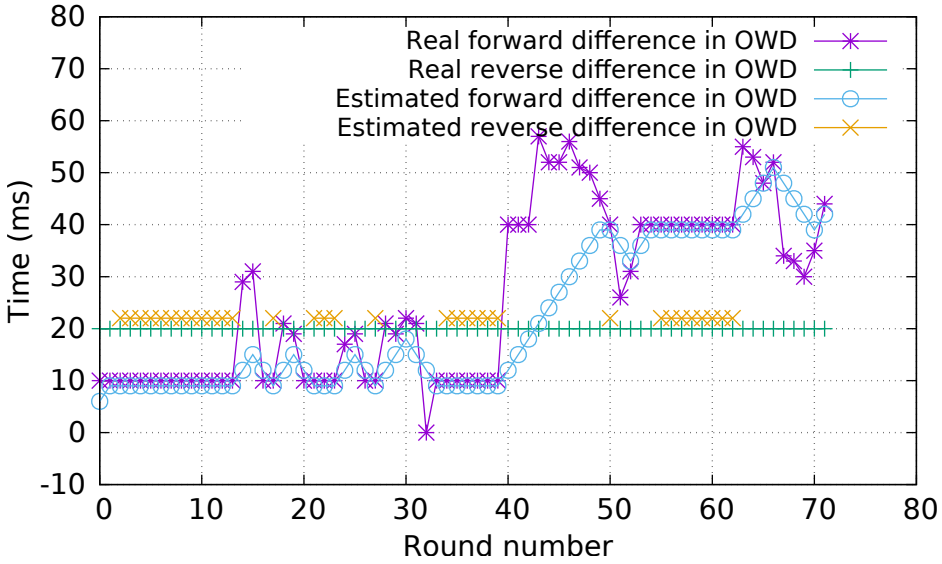

Figure 5.8: Real and estimated ΔOWDs (forward and backward).

.

5.4.2 Discussion

Our original algorithm causes sporadic packet arrival disorder in order to deduce the $\tilde{\Delta}^F OWD$. This information then allows the sender to prevent constant packet arrival disorder. This may look contradictory at first but

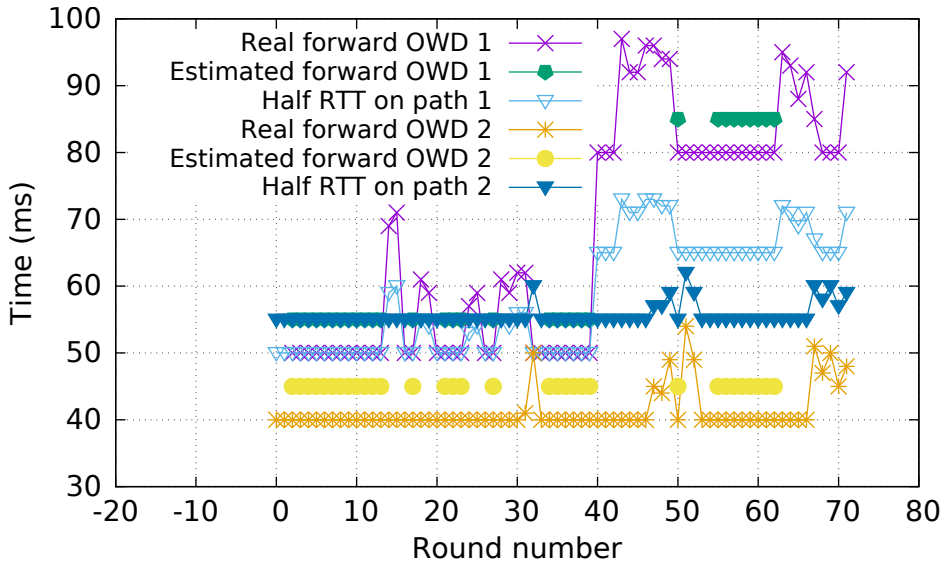

Figure 5.9: Real and estimated forward OWDs.

the packet arrival disorder provoked by the algorithm should occur less frequently then it would without the algorithm, thus proving beneficial in the long run. To mitigate the consequences of packet arrival disorder, one can envision the use of an MPTCP SACK option.

If we consider the adoption of such a scheme in MPTCP, the standard as well as its Linux implementation, integrating the described algorithm would require no compatibility breaking change since the linux implementation already acknowledges each packet. Pacing requires the sender to send packets according to a precise timing, which may be impaired by the numerous batching mechanisms in place at the different layers, be it Nagle algorithm or TCP segmentation offload. This may represent an additional concern.

5.5 Summary

More precision in the estimation of OWDs could improve the performance of various protocols. While single path protocols are limited in options to improve and use that estimation, multipath protocols that require an even more correct estimation hopefully provide additional possibilities. We devised a novel mechanism which has few requirements and tested it against asymmetric delays in the network simulator NS-3. While the algorithm was

applied to two subflows, it could run with more subflows, either running on appropriate pairs of subflows or sending concurrent probes on several subflows.

Chapter 6

Window Management and Buffer Dimensioning for Multipath TCP

MPTCP might improve throughput provided that the TCP buffers are big enough, otherwise the opposite may happen. When facing a situation with many paths available, it might be efficient for MPTCP not to use all of them to prevent throughput degradation because of HoL. How many paths should be used remains an open question. Depending on the use case, it may be important to keep a path alive for confidentiality reasons or because of the financial cost associated with transmitting over the other paths.

We document the MPTCPNUMERICS tool we designed with two roles in mind: first to compute a new MPTCP buffer size depending on the traffic, scheduling and topology. The second role consists in, considering a fixed buffer size, computing the traffic distribution that maximizes the throughput and respect user wishes, e.g., a user can request MPTCPNUMERICS to send a maximum (or a minimum) 40% of the data over a specific subflow[1].

6.1 Introduction

In our era of rich network topologies and multihomed devices (e.g., WiFi/Ethernet for laptops or WiFi/cellular for smartphones), sending data concurrently on multiple paths has become reasonable from a hardware perspective.

[1]The contents presented in this chapter have been published in [299].

Concurrent multipath transmission promises to make better use of the network, increasing the throughput via bandwidth aggregation. It incidentally also allows for smoother device mobility as when one path breaks, instead of losing and creating a brand new connection, the communication can keep going on an alternate path. However, the throughput aggregation expectation has to be re-considered as concurrent multipath transmissions traditionally generate packet reordering (besides possible fairness problems).

MPTCP being backward compatible with TCP, the receiver can deliver only in-order data to the application. Asymmetry in the RTTs among the paths or losses can thus easily provoke HoL, especially in case of small buffers. From a protocol perspective, these problems appear to be best solved at the transport or application layers since these layers have the most information about RTTs, and possibly also about the traffic pattern as explained in [300]. On the other hand, SCTP is a powerful protocol that raised to provide applications with a specific API to use multiple paths; it is available in the vanilla linux kernel but requires to update applications as well as middleboxes that would block by default new protocols. These difficulties seem to have impeded SCTP development so far. MPTCP, instead, takes a backward compatible approach in which applications do not need to be rewritten to benefit from MPTCP. In fact, MPTCP appears as a legacy TCP socket and uses TCP options to inform the distant host to enable MPTCP, if available, otherwise the connection falls back to legacy TCP. In case MPTCP is in use, it can be considered as a shim layer protocol between the application and the (possibly several) underlying TCP connections, also called subflows. Nevertheless, contrary to SCTP, MPTCP cannot disable in-order delivery of data; combined with a small enough buffer, this can degrade the communication to a lower goodput than a legacy TCP connection as shown in [209].

Hence we can summarize several challenges faced by multipath transport protocols:

1. First of all, there is a fairness problem - how should MPTCP users compete with plain TCP users? Should the congestion control fake a single TCP connection over the complete set of subflows, or on sets of subflows sharing a bottleneck (as in [216])?

2. Within an unmodified application, the transport layer does not know the traffic pattern or profile. This makes some optimization difficult,

for instance for short lived communications, establishing additional subflows might end up as a superfluous overhead as the data transmission may end before or short the handshake completion.

3. As mentioned previously, the MPTCP receiver has to deliver in-order data to the application. Depending on the network configuration, concurrent multipath communication can generate out of order packet arrival (due to heterogeneous RTTs for instance) and as such generate HoL, thus resulting in decreased throughput.

4. The previous point limits the number of subflows an MPTCP connection is capable of effectively using. This is even more true if you take into consideration RTOs: for the MPTCP connection to run at full speed during an RTO, it means the receive buffer should be as large as to contain the Bandwidth Delay-Product (BDP) of each subflow for the duration of the RTO + one RTT as explained in [224].

Some of the previously mentioned challenges have full or partial solutions. The fairness issue has been refereed by the MPTCP working group in favor of running backward compatible congestion control on the full set of subflows. The traffic pattern can be characterized using a foreign system (e.g., doing a lookup on TCP ports and/or destination ips as in [262]). Even in an ideal situation where the previous problems are solved, with a high diversity of available paths, some challenges remain, for instance how to learn or infer the degree of path diversity (a potential solution is to consult a network oracle with more information as in [213]).

MPTCP opens up several new scenarios that might benefit the user. For instance, for security or network cost reasons, an application may want to spread the load evenly or in an arbitrary way over all subflows, hence regardless of subflow quality, or trading performance with cost (as suggested in [230]). We thus developed a tool that can help arbitrating some of the previous questions, more precisely how to size buffers to withstand a RTO, i.e., to continue to send new data without HoL? The tool can also compute congestion windows that maximize the throughput under some user constraints (e.g., to better split the load).

In the following, we first present the tool section 6.2, and we then present the results in in section 6.3. Section 6.4 discusses its current limitations.

6.2 Presentation of can compute optimal congestion windows for a specific connection

MPTCP can be seen as a shim layer between the application and TCP connections that is enabled only if both hosts support MPTCP, otherwise the communication falls back to legacy TCP. Once the first TCP connection is established and both hosts acknowledge that MPTCP options are not blocked by the network, it is possible for the MPTCP stack to open additional connections called "subflows" (typically a full-mesh between the client and the server IP addresses). Hosts exchange MPTCP signaling through TCP options.

The more subflows, the better it shall be in terms of path diversity, however too many subflows increase the likelihood of packet arrival disorder and consequently HoL. Thus in some situations, it may be wise to limit the number of subflows actually in use (they can remain as backup subflows), ideally keeping the best subflows. A subflow can be better than the others for various reasons: it can exhibit a good BDP or because it enhances the path diversity (for instance a user might consider that a cellular subflow is less likely to share nodes along its path with its wired subflow than with a WiFi subflow).

When dealing with a wide choice of subflows to schedule data to, it can be interesting to have a tool to help refereeing such trade offs. MPTCPNUMERICS is such a tool; it accepts as inputs a scheduler, a network configuration along with some constraints added by the user.

The network configuration input gives the size of the send and receive buffers, a description of each subflow in terms of OWDs, MSS, etc as shown in listing 1. There are currently three greedy schedulers available - they send data on a subflow as soon as possible, and their behavior differ in the way they sort subflows during the simulation setup (based on the OWD for instance). The source code for MPTCPNUMERICS is available at [301].

MPTCPNUMERICS works under two different modes:

1. The first mode advises a buffer size to accommodate the MPTCP flow control constraints and can help to parameterize its operating system to maximize the MPTCP throughput. Optionally the user can ask the program to cap the maximum congestion window so that a RTO does not trigger a HoL.

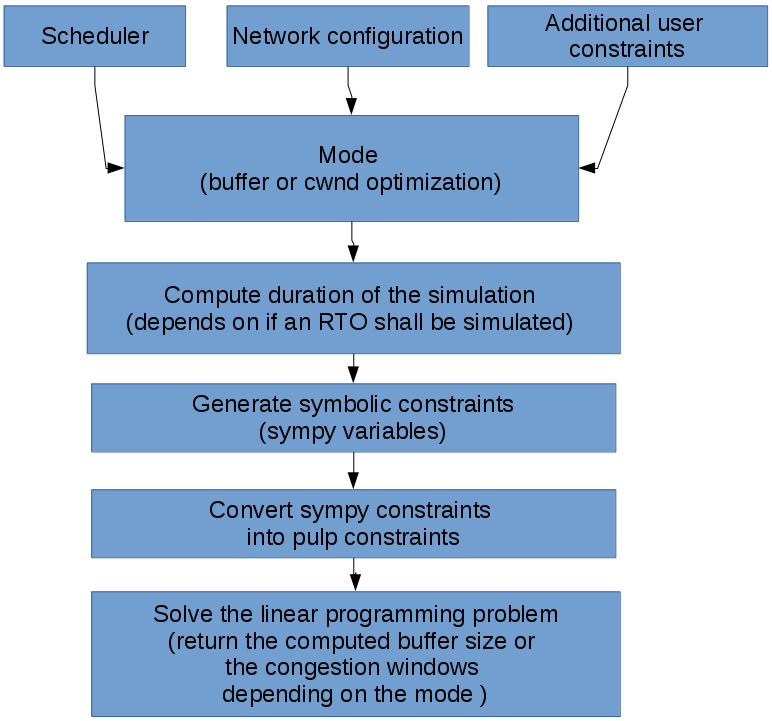

Figure 6.1: An overall run of the MPTCPNUMERICS program.

2. The second mode computes the congestion window that allows for a maximum goodput within the boundaries the different user constraints, given the size of the receive buffer. For an instance, a user may suggest that a subflow contributes to 40% of the throughput.

The two modes are detailed in section 6.3.

An example of a program run is pictured in fig. 6.1.

Ideally such values could be computed in closed form formulas but this appears challenging due to the non-linear behavior of the window and the multiple features to take into account.

On the other hand, implementations such as [227] or time discrete simulators such as [302] just react to events and do not anticipate values for the congestion windows. MPTCPNUMERICScan help an underlying MPTCP stack to schedule the data stream depending on the congestion window targets computed. To achieve this, MPTCPNUMERICS makes a few hypotheses that are presented in the next section.

6.2.1 Implementation

MPTCPNUMERICS is a minimal discrete event simulator written in python, just enough to be able to simulate TCP flow control, packet acknowledgment and out of order arrival.

It loads the network setup via a JavaScript Object Notation (JSON) configuration file which contains the size of the receive buffer, a description of each subflow in terms of OWDs, MSS as shown in listing 1. The user can then interactively add several constraints such as limiting the maximum congestion window on a subflow or on the contrary make sure a subflow is used via setting a minimum window.

MPTCPNUMERICS is not meant to replace discrete time simulators such as [302]: its objective is to foresee a coarse view of what would happen in a nominal state so as to help the MPTCP stack prioritize subflows over others (schedule more often or use a more aggressive window growth). The simulator in [302] only reacts to events without trying to establish a strategy beforehand. In the MPTCP context, capping the share of some subflows can prove rewarding in the long run as it allows the congestion control to be more aggressive on the preferred subflows.

The objective of the simulator is to look for the desired state of the system in a situation where the hosts are buffer limited rather than network limited, i.e., there are no losses due to congestion and the congestion windows are capped because of buffer constraints. As such the simulator does not simulate TCP window growth and congestion control issues are not taken into consideration. Subflows do not experience losses except when enforced to simulate a RTO event. As in [304], we suppose that windows are sent in rounds and that in-order packets are forwarded immediately to the application. When applied to MPTCP, it implies that the transmission of a subflow window must be negligible compared to any of the OWD. In order to keep these hypotheses reasonable, the simulation duration is short and consists of the least common multiple between RTTs. When a RTO is simulated, the minimum simulation duration is set to $RTO + RTT$ of the subflow experiencing the timeout.

It mostly relies on two open source Python libraries:

- SymPy [305] is a library for symbolic mathematics, which allows us to generate flow control constraints as well as user constraints without any concrete values.

- pulp [306] is a linear programming modeler that can delegate the problem to several linear programming solvers such as cplex [2], glpk [3], Cbc (Coin-or branch and cut) [307].

As the number of constraints is limited, the performance of the different solvers should not be relevant hence we chose to rely on the default one (Cbc). Indeed the number of possible subflows on Linux is currently limited to eight and the simulation needs only to be updated when a parameter referenced in Table 6.1 significantly changes (e.g., the measured RTT on a path).

One difficulty we met is that SymPy and pulp variables are not compatible with one another, the program has to convert SymPy expressions to pulp expressions, mapping each variable to its twin in the other library.

6.3 Running modes and Results

MPTCPNUMERICS can run with two different objectives: in the first mode it minimizes the buffer, while in the second mode it maximizes the estimated throughput, constrained by subflow-specific policies. Depending on the mode, certain values of listing 1 are ignored and replaced with a more precise mathematical notation resumed in table 6.1. Parameters and variable are presented in table 6.1.

6.3.1 Minimize the Multipath TCP buffer size

The official guidelines (see [211]) recommend to use a buffer γ of size:

$$\gamma \geq 2 \sum_{i}^{N} BW_i * \max_i RTT_i \tag{6.1}$$

to deal with fast retransmits (hence the factor '2 '), N being the number of subflows and BW_i the bandwidth of the subflow i. Similarly, to prevent HoL one needs:

$$\gamma \geq \sum_{i}^{N} BW_i * \max_i RTO_i \tag{6.2}$$

[2] `http://www.cplex.com/`

[3] `http://www.gnu.org/software/glpk/glpk.html`

An RTO is defined in the TCP RFC793 [308] as:

$$\begin{aligned} RTTVAR &= (1-\beta) * rttvar' + \beta|srtt - rtt| \\ SRTT &= (1-\alpha) * SRTT' + \alpha * RTT \\ RTO &= max(SRTT + 4 * RTTVar, RTO_{min}) \end{aligned}$$

The recommended values in Jacobson [309] for α and β are respectively 0.125 and 0.25 with RTO_{min} default is of 200ms on linux. The previous constraint can end up requiring a huge buffer to handle the worst case of a bulk transfer but for burstier traffic patterns, it might possible to run without HoL with smaller buffers, especially in lossless networks. The buffer upper bound then depends on the scheduling and the various TCP options in use. Requiring a smaller buffer is desirable on constrained devices such as sensors or handheld devices. Even for a server, using smaller buffers might allow to accepts more connections.

In this mode, MPTCPNUMERICS helps to size the buffers depending on the scheduler and the topology. We describe the expected inputs and outputs of the program in fig. 6.2. First, the program needs a configuration similar to listing 1, from which it discards the `rcv_buffer` and `send_buffer` parameters since these are the values it is meant to compute. The parameters "cwnd" are used as the maximum congestion windows w_i^{max} and "mss" as mss_i in **??**. In practice, the maximum congestion window value could be set to the maximum value witnessed during the subflow existence for example. The second optional input enforces a loss on a specific subflow. Under our assumptions, this is equivalent to losing the whole subflow window and triggers a RTO. This allows to find the buffer size to accommodate a RTO on a specific subflow. It is thus possible to find the subflow most likely to trigger HoL and discard it from the pool of active subflows.

$$\begin{aligned} &\min \gamma \\ &\forall i \in [0; N] \\ &mss_i \cdot cwnd_i^{max} \leq \gamma - \sum_{j=1}^{N} \sum_{p \in P^j(t)} 1^{inflight}(p) \cdot mss_j \cdot cwnd_j^{max} \end{aligned} \tag{6.3}$$

In particular, $1^{inflight}(p)$ is a boolean parameters that holds 1 if the packet $p \in P^i(t)$ is in flight (i.e., sent but unacknowledged).

The optimization problem is solved at time d, d being the duration of the simulation during which constraints are generated.

Thus we run our program on three different network configurations: the first one with 2 subflows, the second with 3 subflows and the last with 6 subflows. The subflows have the same congestion window set to 20 MSS and the same RTT of 40ms. Only the repartition of the RTT differs between the forward and the backward OWDs such that $owd_{forward} + owd_{backward} = RTT$. Each of these configurations is then run against three different basic schedulers:

- the increasing ('Inc.') scheduler sorts subflows according to their forward OWD before sending data. Notwithstanding the variance in delay, it means that the data arrives in order and can be acknowledged immediately, then passed to the upper application thus freeing the receive buffer.

- the decreasing ('Dec.') scheduler does the opposite, i.e., it sorts subflows from the biggest to the lowest forward OWD. It exhibits the worse results as packets systematically arrive in disorder: the buffer can only be freed with the last packet.

- the default scheduler just forwards data on the subflows in the order they appear in the configuration file. Its performance is in-between the two others.

Figure 6.3 reports the results of the simulation. The recommended buffers for fast retransmit (Eq. 6.1) and RTO (Eq. 6.2) situations (respectively 'FR' and 'RTO' in fig. 6.3) are plotted for reference (they are not directly comparable with the previous results as they take into account potential losses but, even so, lower bounds are possible if the application does not transmit continuously). We can observe that the difference between the Inc. and Dec. matches the size of the out of order windows. In the '2 subflow'simulation, using Dec. doubles the required size compared to the Inc. scheduler because subflow windows are equal. These results show the importance of taking OWDs into consideration during the scheduling process. Most schemes rely on the RTT because OWDs are difficult to obtain and less reliable [266], but even a difference between OWDs is enough to help schedule packets for in order arrival.

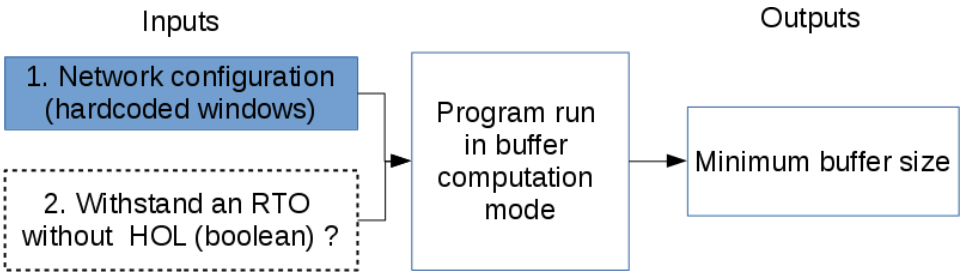

Figure 6.2: Inputs and outputs for the buffer computation program. Dotted frames are optional constraints.

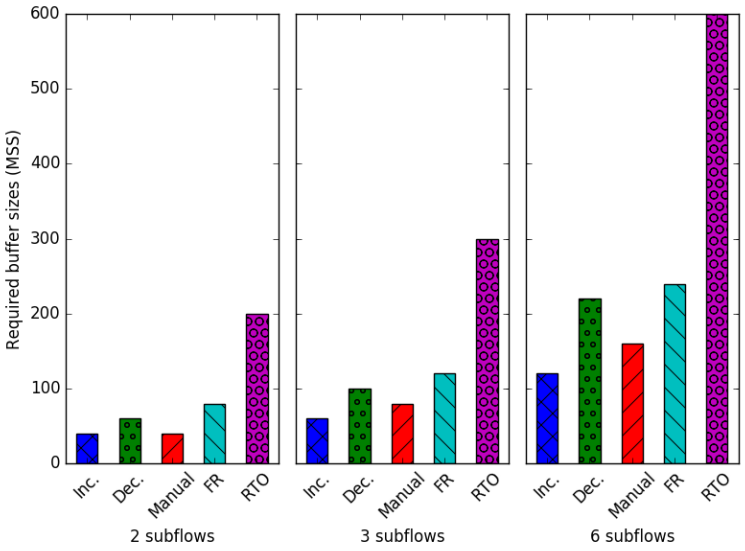

Figure 6.3: Required buffer sizes in number of MSS with different schedulers (Inc. stands for Increasing scheduler, Dec. for Decreasing scheduler, and 'Manual' means the subflows are used in file order).

6.3.2 Maximize the estimated throughput

Contrary to the previous mode, in the following mode buffer sizes are fixed. The objective is to compute what could be optimal congestion windows for an MPTCP connection in congestion avoidance mode.

Imposing constraints on some subflows to prevent HoL has already been used in the literature: for instance in [310], the authors propose an online scheduler that computes if there is enough free buffer to account for out of order packets if such subflow is used. An alternate approach (for SCTP) is to keep per subflow buffers [311].

Compared to the previous examples, we add some constraints to force traffic on otherwise discarded subflows, to improve the supposedly path diversity, which can be interesting from a security point of view. Alternatively it can be interesting for a user to limit the throughput of a well-performing subflow in case transmitting on this subflow represents some cost for the user (energy or financial cost, ideally MPTCP should allow for [230]).

We describe the expected inputs and outputs of the program in fig. 6.4. The first two options are the same as in fig. 6.2. The inputs (3) and (4) allows respectively to set the α_i and β_i to impose restrictions on subflow i contribution. First, the program needs a configuration similar to listing 1, then it accepts an optional flag to enforce a loss on a specific subflow.

Once the MPTCP stack recovers the target congestion windows, it can create different sets as is done in OLIA [4]. Legacy OLIA sorts subflows into different sets, and adapts the congestion control aggressiveness according to the set. In our case, once a subflow reaches its target congestion window, its aggressiveness could be reduced or even capped in favor of other subflows whose congestion windows have not yet reached their optimal value.

Table 6.1: Parameter and variable summary.

Variable	Description
$T \in \mathbb{N}$	The global throughput of the connection.
$T_i \in \mathbb{N}$	The throughput of the subflow i.
$\gamma \in \mathbb{N}$	The buffer size common to both hosts. In the buffer mode, it is a variable to minimize.
Parameter	Description
$N \in \mathbb{N}$	The number of subflows
$\alpha_i \in [0;1]$	Let the user limit subflow i throughput.
$\beta_i \in [0;1]$	Let the user enforce a minimal throughput contribution from subflow i.
$n_i \in \mathbb{N}$	The number of acknowledged windows on subflow i during the simulation.
$mss_i \in \mathbb{N}$	The MSS on subflow i. Always set to one in the experiments, hence the throughput can be read as a number of packets.
$cwnd_i^{max} \in \mathbb{N}$	The maximum window available on subflow i. Its value is read from the field "cwnd "as can be seen in the configuration file in listing 1. In number of MSS.
$\gamma \in \mathbb{N}$	The buffer size common to both hosts. In the congestion window mode, it is read from the configuration file.
$P^i(t)$	The chronological set of packets p^i sent on subflow i before time t.
$1^{inflight}(p)$	boolean value that holds 1 if the packet $p \in P^i(t)$ is in flight (i.e., sent but unacknowledged), else 0.

$$
\begin{aligned}
&\max_{cwnd_i} T \\
&T = \sum_{i=1}^{N} T_i \\
\forall i \in [0, N] \quad & T_i = n_i \cdot cwnd_i \cdot mss_i \\
\forall i \in [0; N] \quad & T_i \leq \alpha_i \cdot T \\
\forall i \in [0; N] \quad & T_i \geq \beta_i * T \\
\forall i \in [0; N] \quad & 0 \leq cwnd_i \leq cwnd_i^{max} \\
\forall i \in [0; N] \quad & mss_i \cdot cwnd_i \leq \gamma - \sum_{j=1}^{N} \sum_{p \in P^j(t)} 1^{inflight}(p) \cdot mss_j \cdot cwnd_j
\end{aligned}
\tag{6.4}
$$

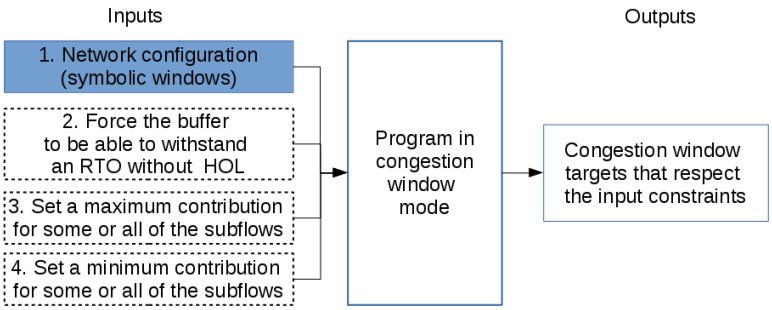

Figure 6.4: Inputs and outputs for the congestion window computation program. Dotted frames represent optional constraints.

In fig. 6.5, we compare the outcome of the program in two network configurations and with different constraints;

- Simulations 1, 2 and 3 are done with the same configuration of listing 1, which consists of 2 subflows, one slow subflow with twice the RTT of the slower subflow. The buffer sizes are fixed, as well as the maximum congestion window on each subflow. In simulation one without any constraints, we see that only the first subflow is used. In simulation 2, we added a constraint so that the contribution of the first subflow does not exceed 40% of the total throughput. As it is the best performing subflow, the limit of 40% is reached and mechanically the second subflow contributes for 60% of the total MPTCP throughput. We notice that the global throughput suffers a net loss. In simulation 3, we this

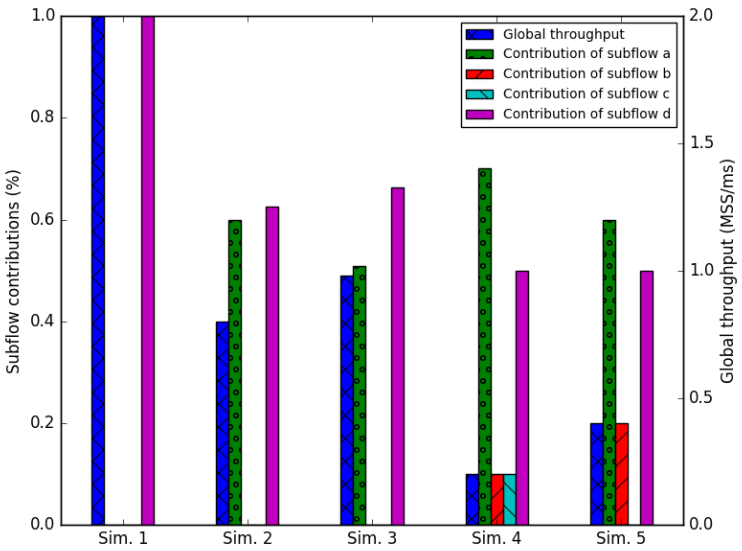

Figure 6.5: Global throughput right-most bar (MSS/ms) preceded by subflow contributions to this throughput in percentage.

time set a minimum contribution of 50% for subflow 2. The results are pretty similar to simulation 2.

- In simulations 4 and 5, we use a 4-subflow topology with the second subflow with a higher maximum congestion window. In simulation 4, we see that this subflow contributes more than the others to the global throughput. In simulation 5, we request a minimal contribution of 20% for the first and third subflow. The global throughput does not change but the traffic is more spread on these 2 subflows compared to simulation 4.

6.4 Limitations and future work

In this section we describe how MPTCPNUMERICS can be further improved.

6.4.1 Addition of new constraints

In [310], the authors take into account that applications are not solely about bulk transfer but can also be latency sensitive, as such they propose a constraint delay (static or dynamic) that discards a subflow from the scheduling

process even when this subflow would not cause HoL.

6.4.2 Support of proposed MPTCP features

Any technique related to alleviate the impact of RTOs is likely to dramatically increase the worst case performance as it is the case with opportunistic retransmission scheme [102]. When a subflow suffers from a Transmission-Induced Sender Buffer Blocking (see [311]), rather than waiting for the acknowledgments on the slow paths, it reinjects the unacknowledged data with the hope of freeing the sending window sooner. Non-Renegotiable Selective Acknowledgments (NR-SACK) [272] is also an interesting feature to help alleviating the sender blocking that could be added.

6.4.3 Integration with an MPTCP stack

Eventually, MPTCPNUMERICS should be able to communicate with a real stack. While the buffer mode can be used offline to compute in advance the required buffer size, the congestion window mode is best run online as network characteristics such as RTTs or loss rates can evolve during the connection; even subflows can be created or deleted, especially in mobility scenarios. To accommodate an online scenario, our program would need to learn from the underlying MPTCP stack the network context (number of subflows, RTTs), do its computation and then return the per subflow target windows. The interface between user-space and kernel-space is the netlink interface. An easier path might be first to interface MPTCPNUMERICS with the user-space simulator [302], as it allows to use user-space utilities.

6.4.4 More evolved trade offs

In our previous scenarios, we would emphasize to force traffic over otherwise disabled subflows to improve the path diversity or to reduce an overall cost. These constraints are explicitly set on a subflow basis but a friendlier approach could rely on an utility framework or let the user set a threshold, for instance, the user could allow a decrease of 20% of the anticipated throughput to better spread the traffic.

6.4.5 Taking into account the RTT variance

A variance in RTT can generate further delay in the receive buffer because of out of order packets, i.e., a higher variance in RTT leads to worse worst case situations. We should also be able to set a different send and receive buffer size so that schedulers can take this difference into account to decide to pace packet transmission (if the sender buffer is bigger than the receive buffer).

6.5 Summary

We created a program that can assist a MPTCP stack in realizing more elaborated scenarios than just looking at increasing throughput. Our program, called MPTCPNUMERICS, is also capable of providing lower limits for the buffer size taking into account scheduling possibilities. It makes a few strong hypotheses that can be considered reasonable in respect to the goal of the program, i.e., to provide targets for congestion windows. We plan to interface it with the NS-3 MPTCP simulator [302] to measure how accurate our estimations can be, despite the model approximations.

```
{
    "name": "mytopology",
    "sender": {
        "snd_buffer": 40,
        "capabilities": ["NR-SACK"]
    },
    "receiver": {
        "rcv_buffer": 40,
        "capabilities": []
    },
    "subflows": {
        "subflow0":
        {
            "cwnd": 20,
            "mss": 1,
            "var": 10,
            "fowd": 19,
            "bowd": 20
        },
        "subflow1":
        {
            "cwnd": 20,
            "mss": 1,
            "var": 10,
            "fowd": 20,
            "bowd": 20
        }
    }
}
```

Listing 1: A typical network configuration file (JSON format [303]) as used in fig. 6.3.

Chapter 7

MPTCP in NS-3: implementation and evaluation

When working on the topic described in Chapters 4 and 5, we realized how long and difficult it could be to prepare an experiment and then interpret the results without any tool understanding the MPTCP protocol. We would have liked to analyze and plot some metrics such as the OWD or Data Sequence Number (DSN) inter-arrival durations, both inter and intra subflows but we were unable to do it in a straightforward manner.

This motivated us to look into ways to ease the analysis of MPTCP traffic as well as ways to speed up the prototyping phase[1].

7.1 Introduction

The MPTCP technology being still recent, the related software ecosystem used to be scarce and overall still is. All the automation expected from the environment (for instance the Operating System (OS)) - such as routing tables autoconfiguration to leverage all interfaces - is non-existent, and to our knowledge still is.

Likewise when confronted with trace analysis, the packet analyzer wireshark would only translate the MPTCP bytes to a human readable string: the very basic operation of mapping TCP subflows to their respective MPTCP

[1]The contents presented in this chapter have been exposed in [302].

connections remained unavailable.

The lack of support is consubstantial to innovative technologies. In the case of communication protocols though, the burden is made true that even if you tweak positively your environment, the procedure has to be replicated over the communicating nodes. Industry has started relying on tools such as [2], Chef, Ansible [3], Salt [4] etc... which require to setup installation scripts once to automate deployment. These solutions profitability grow with the scale of the deployment. As for typical researchers whose experiments typically concern pairs of host, the gain is hypothetical.

These shortcomings make MPTCP experimentation very costly as we witnessed first hand during our experimentations in Section 4.5.

Consequently developing features and testing features for the MPTCP linux kernel proved time consuming. Hence we decided to rely on existing MPTCP models for discrete time simulators for our experiments as it allows to run reproducible self-contained experiments. While experimenting and modifying with the most complete available implementation at the time, we realized that many aspects of the protocol were missing and that the current architecture would not allow for wide adoption of the implementation, which we see as critical for a software to improve and develop. Thus we ended up spending considerably more time than anticipated on the implementation. In a second time, we found another tool called DCE that provided an interesting middle ground between the solutions, i.e., the ability to run the proved linux implementation in a discrete time simulator. We decided to exploit this to assess our new model. Towards the end of the PhD, we learned of yet another promising simulation technique to bridge the gap between discrete time simulations and real hardware experimentation.

Our goal is to allow researchers develop and evaluate new features of MPTCP using our simulator much faster than they would with a kernel implementation, hence boosting MPTCP research.

In Section 7.2, we introduce different concepts like simulators and testbeds with a few chosen examples of interest.

[2] `www.puppetlabs.com`.

[3] `www.ansible.com`.

[4] `www.saltstack.com`.

7.2 Simulation frameworks and testbeds

As we investigated frameworks that we could use for MPTCP simulations, we discovered a rich diversity of available tools, each software filling a niche. Commonly used platforms in networking systems research include simulators, testbeds, and emulators.

Testbeds are in general a shared infrastructure (such as PlanetLab [312] or GENI [313]) which allow to engage experiments on real hardware but are expansive to maintain and in general with an uneasy access.

There are two main aspects to take into consideration:

- Functional realism: the scenario relies on the same functionalities implemented in real hardware and can execute the exact same code. Functional realism narrows the gap between emulation/simulation and real deployment.

- Timing realism: The timing behavior of the system must be close enough to the behavior of deployed hardware such that the researchers should reach the same conclusions.

There are other more prosaic characteristics to consider such as ease of use.

Realtime simulators are simulators that can run at the same rate as the actual "wall clock" time. For instance, the real-time simulation of a ten minute transfer runs for ten minutes in the real world.

7.2.1 Mininet

Mininet [314][5] is an emulator that leverages the so-called linux “containers”to create virtual networks, hundreds of virtual nodes running real code within a real linux kernel on a single machine. Linux containers, similarly to Berkeley Software Distribution (BSD) jails, provide a lightweight form of OS virtualization: the creation of virtual bridges, virtual filesystems, etc. . . is thus possible with a limited cost. Mininet exhibits functional realism as it runs unmodified applications.

The main problem with emulators such as Mininet concerns timing realism: even if OS virtualization lowers the cost compared to traditional

[5] `http://mininet.org/`.

virtualization, it still exists and it is difficult to imagine that a single machine emulating bulk transfer between hundreds of nodes can reach the same results as a hundreds machines. Nevertheless for simpler scenarios, Mininet Hi-Fi [315] mitigates some of the problems with the original Mininet via the addition of resource isolation and monitoring. This is accomplished via the linux 'cgroups'that allow to limit the Central Processing Unit (CPU) consumption on a per virtual node basis. The linux tool TC is also used to control network links characteristics such as loss, variance, throughput.

The claimed objective of the Mininet authors is to demonstrate that network research can be made repeatable, ideally in a simple click (to launch the Mininet script). In order to polish their emulator along these guidelines, the Mininet research group asked a class to reproduce published results with Mininet [6]. While it is successful at being a one-click launcher for research experiments, there are still reports of unrealistic results under high-load experiments as reported in [316], [317].

7.2.2 Discrete time event-driven simulations

Discrete time event-based simulators first the simulator selects the next available event, then it updates its internal counter (i.e., clock) to the scheduled time of the selected event. Compared with realtime simulators which increase their counter with a constant step, this approach allows to skip periods of time where nothing happens to speed up the simulation run. It also provides - per construction - frequency and time synchronization.

Presentation of NS-3

NS-3 [297] is the successor of NS-2 a popular network simulator in the research community, which organic development threatened the simulation quality. NS-3 success is likely due to its General Public License (GPL) and also because the technical base as well and the support team are trustworthy. It is best described as a C++ discrete time simulator, i.e. events are scheduled in the simulator time and once all events at the specific time are processed, the simulator updates the current time with the time of the next scheduled events.

[6] `https://reproducingnetworkresearch.wordpress.com/`.

There are many NS-3 extensions, some of them providing interesting new features:

- [318] allows to connect Mininet nodes via NS-3 links, providing more accurate traffic patterns
- Network Simulation Cradle (NSC) [319] extracts the network stack from OSs like FreeBSD so that it can used in the simulator.

It allows the simulator clock to be independent from the wall clock, most of the times faster.

DCE: a bridge between simulators and emulators

DCE is a NS-3 extension that allows to load applications compiled with specific options within the NS-3 environment. The advantage is that the simulation runs in discrete time and thus provides results independent of the host CPU, i.e., perfectly equal whatever the processing power and even across hosts.

More precisely a binary executed by DCE perceives time and space of NS-3, rather than the real environment.

To do this, DCE does a job equivalent to an OS like:

- DCE loads in memory the code and data of executable,
- DCE plays the role of intermediary between the executable and the environment through the systems functions called by executables,
- DCE manages and monitors the execution of processes and handles liberate the memory and close open files when the stop process.
- DCE manages the scheduling of the various virtual processes and threads.

C programs typically rely on a few C libraries such as:

- the C standard library also called ‘libc‘ for system related operations; for instance memory-related functions like *malloc*, *free*...)
- libm for maths operations ("ceil", "floor" ...)
- libpthread to manage Portable Operating System Interface (POSIX) threads ("pthread_create")

DCE generates *shim* libraries which aim at providing the same C symbols (function names). Some of these symbols are just wrappers for the original function while other functions are implemented so that programs can work in the simulator. As an example, it makes sense in a discrete time simulators to override all functions related to time management such as *gettimeofday* or *setitimer*.

Even more interesting, DCE is capable of running a fork of the linux kernel called LibOS [320] which once combined with the MPTCP kernel allows to run a linux MPTCP kernel in NS-3. This is this precise feature we leverage in Section 7.4 in order to compare fairly our implementation to the linux one. DCE proposes an interesting middle ground between pure limitation and real experimentation. There is still an overhead as some functions have to be reimplemented, also the official implementation is only compliant with a specific compiler (GCC) extension which limited the portability and caused several failures depending on the system. We spent a bit more than a month to adapt DCE to our platform.

Discussion

Table 7.1: Comparison between experimentation technologies.

	Functional Realism	**Timing realism**	**Ease of debugging**	**Ease of setup**
Container emulation	√	√	1	3
DCE	√	√	3	1
VMSimInt [321]	√	√	2	0
NS-3-like simulators		√	3	3

We summarized the current state of the different approaches in Table 7.1. We ranked the ease of debugging and ease of setup between 0 (harder) and 3 (easier). Though in theory DCE, can run real code, in practice software often needs to be slightly patched to deal with bugs related to 0-duration operations (due to the time discrete environment), and recompiled with specific compiler flags to match DCE expectations. DCE itself is a complex program reimplementing a virtualization layer but in a way different as QEMU for instance, as it needs to map NS-3 API to the POSIX standard. The ease of setup mark for VMSimInt is *currently* the lowest because the software itself is not documented, not updated The ease of debugging should be similar between VMSimInt and container emulation, notwithstanding the additional possibility ability of VMSimInt to debug kernels without making

its host crash.

On the one hand, emulators allow for faster transition between in-lab experiments and real-world experiments. On the other hand, timing realism is not always guaranteed. Results may be trusted for small scale experiments but the boundaries defining small scale experiments are not exactly defined. A practical solution would be for the system to detect when timing realism is violated and warn the researcher. Yet we do not know of emulators with such a capability. An interesting approach is to bias the time perceived by the applications or virtual nodes with a Time Dilation Factor (TDF) in a technique called *time dilution* first presented in [322] and then applied to Mininet in [317]. It allows to trade wallclock time against fidelity, i.e., the experiments run longer because with higher fidelity. Advanced emulators monitor resources such as CPU utilization and can adjust the Time Dilation Factor (TDF) accordingly.

Discrete time simulators are still popular with the two main contenders being OMNet++ [323] and NS-3. The functional realism these simulators lack is partially balanced by the timing realism they achieve and also the perfect reproducibility. With equal randomization generator parameters, consecutive runs yield the exact same results which makes it easier for researchers to understand the results. The downside is that you need to prepare two sets of code, one for your simulator and one for the real world experiment.

Hopefully, there is a clear trend to blend the two approaches. The front runner of this approach is DCE, an extension to NS-3 which allows to run real code within NS-3, even a linux kernel. It is the most mature solution we could find, even with its many shortcomings: code needs to be compiled with specific options, some programs may need to be - slightly - patched. An interesting solution similar to DCE with fewer limitations seems to be VMSimInt: instead of reimplementing the POSIX layer as DCE does, it delegates the virtualization to a discretized version of QEMU [324]. We are not aware of any related user apart from its creators but the potential is real.

The tendency seems to let simulators handle link-level simulations as wifi or LTE communications since this is unavailable in host OSs and coordinate the scheduling of packets accordingly in some virtualization layer: original for DCE or outsourced (QEMU) for VMSimInt.

More elaborate features can emerge from combining some of these techniques. For instance, a smartphone user may enable both LTE and WiFi to benefit from the mobility advantage and at the same time limit the cellular throughput to save some battery or because WiFi is actually cheaper. Some other user may choose to trade some of the aggregation benefit in exchange for higher confidentiality.

Information such as the RTT or the packet sequence number are critical to mitigate these problems and are already available at the transport layer. While the application layer could provide a similar or even better service, having a standard multipath transport protocol allows to mutualize the knowledge and should ease multipath communications deployment.

7.3 An MPTCP implementation in NS-3

A few MPTCP implementations already exist, some used in production environments such as Apple's voice recognition system Siri. Among the implementations, Linux[7] is the oldest one with some impressive achievements (Fastest TCP connection [227]). Work is also done to improve the MPTCP support on other operating systems such as Solaris[8] and FreeBSD[9], Hence asking why developing a MPTCP simulator is a legitimate question. In this section we describe our motivations and the technical aspects of the implementation. We also present a few tools we developed to ease testing and analysis of related MPTCP traces.

7.3.1 Why a simulator ?

Simulation traditionally comes handy for two reasons:

1. Running experiments in a simulated testbed allows for faster reproducibility avoiding hardware costs.

2. Focusing on the algorithmic part rather than implementation complexity. Implementation details can have an impact on the overall fidelity

[7] `http://multipath-tcp.org`.

[8] `https://mailarchive.ietf.org/arch/msg/multipathtcp/ugMIu566McQMn8YCju-CTjW9beY`.

[9] `http://caia.swin.edu.au/urp/newtcp/mptcp/`.

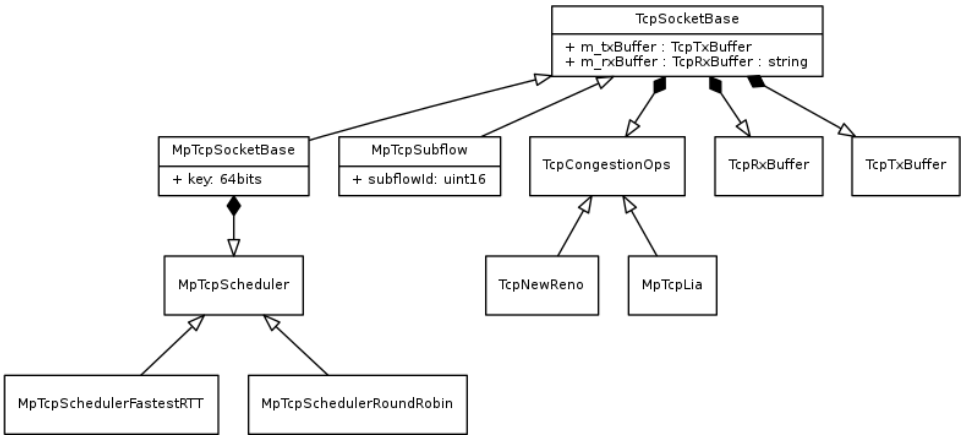

Figure 7.1: Implementation structure in NS-3 code.

of the model. Hence being able to compare with a simpler model beforehand can help find out if the difference in performance stems from implementation details or from the algorithm.

Experimenting with MPTCP in the real world can be complex depending on the scenario. Mobility is a major use case and usually requires access to LTE and wifi. Not only does it have a cost but LTE is not ubiquitous and experiments involving wireless channels are time consuming because of the variability and care their setup require. Other experiments rely on accurate time measurements (e.g. to measure one-way delays as in [266]), which can prove challenging in real setups but are straightforward in discrete time simulators.

Simulations can help find and solve a problem before the real test and result in a huge time gain. Running an experiment in a time discrete event simulator such as NS-3 can also be faster than running its real time equivalent.

Point 1) alone does not justify yet another implementation since testing can also be realized through alternative means. In simple cases, container-based simulations such as Mininet testbeds can be enough but at higher throughput, hardware limits (e.g. processor speed) can spoil the results: switching to discrete time solution such as DCE (see section 7.2.2) makes sense in that case.

The point 2) can be considered as the stronger motivation, especially when looking back at the number of use cases described in section 3.5.

Table 7.2: Comparison between NS-3 MPTCP simulators.

Features	Chihani et al. [325]	Kheirkhah et al. [326]	Our implement.
Option serialization	Partial	Partial	Full
Standard compliance	Connection phase	Connection phase	Full
Backward Compatibility	No	No	Yes
Ack-aware buffer mgnt	No	No	Yes
Comparison to OS implem.	No	No	Yes

Implementing such solutions into current operating systems usually means adding the features into the kernel. While simulation results may lose fidelity compared to a reasonable kernel implementation, we argue that kernel development complexity can generate bad implementations that can not be easily verified and may not be representative of expected results/analytical models. In those cases, developing a simulation model beforehand is reasonably faster and can help realize problems ahead of time.

As a side effect, we also think the implementation can serve for education purposes since the model only deals with MPTCP essentials, thus reducing the learning complexity.

7.3.2 Related work

We found one advertised MPTCP public model developed for NS-2 but NS-2 is an obsolete simulator and the model was lacking in many ways. We have been able to access two previous MPTCP implementations, [325] and [326], both done using NS-3 as well. These two implementations are similar in many aspects and are compared with ours in table 7.2.

Recent developments in NS-3 such as TCP option support and generic packet serialization in a wire format made it possible for NS-3 to communicate with real stacks. Contrary to previous NS-3 implementations that support a subset of the options, ours support full (de)serialization of MPTCP options, which means it can handle a higher variety in options (e.g., 32 and 64 bits encoding for DSNs).

To allow the communication with an external stack such as the linux one, we also implemented standard compliant connection and closing phases, which is another differentiating point from [325] and [326]. Thus our implementation is capable of generating valid tokens based on the sha1 hash of a random key, and closing a connection requires the sending and acknowledgment of a DSS with the data fin bit. While the implementation is not robust enough yet to handle all cases, it managed to exchange a file with an external linux MPTCP stack with the use of DCE as reported hereafter.

Contrary to [325], [326], our implementation is backward compatible with existing NS-3 TCP scripts, following MPTCP's spirit. Thus in our implementation, the connection phase starts with a legacy TCP socket (more precisely a 'TcpSocketBase' see fig. 7.1) and only once an MPTCP option is received it evolves into an MPTCP socket (see 'MpTcpTcpSocketBase' in Figure 7.1). This allows for better integration with the general framework, and adds the additional benefit of allowing the MPTCP connection to fall back to TCP. Our hope is to be able to upstream this implementation so that improvements can then be added incrementally: we have started steps in that direction in January 2016 but this requires many changes to the core and is particularly hard to iron out details to obtain a consensus on how to ingrate this work in NS-3.

We also respected an aspect of the specification that could affect the simulation fidelity, i.e., data can not be removed from the subflow sockets until it is acknowledged at both the TCP and MPTCP levels.

Finally, our implementation is also the first to our knowledge to be evaluated against an operating system stack in comparable conditions as described later in section 7.4.

7.3.3 Supported and missing features

The implementation was developed in ns-3.23 while giving care to performance and algorithmic aspects. As such, the fallback capabilities (MP_FAIL option, infinite mapping and checksums) of the protocol have not been implemented with the exception of the initial fallback, when the server does not answer with an MP_CAPABLE option, i.e. it does not support MPTCP and the client falls back to legacy TCP. This was made possible by extending the existing NS-3 code infrastructure; for instance in Figure 7.1, only the structures starting with "MpTcp " were added. It also spares some resources

Table 7.3: List of supported and missing features.

SHA1 support	We added an optional SHA1 support in NS-3 to generate valid MPTCP tokens and initial DSNs. This allows to communicate with a real stack and also proved necessary for wireshark to be able to analyze the communication.
Scheduling	The fastest RTT and round robin schedulers are available.
Congestion control	Subflows can be configured to run TCP ones such as NewReno or LIA.
Mappings	As in the standard, data is kept in-buffer as long as the full mapping is received. This is necessary when checksums are used, otherwise this can be disabled to forward the data faster.
Subflow handling	It is done directly by the application that can choose to advertise/remove/initiate/close a subflow at anytime if it is permitted by the protocol.
Packet (de)serialization	Packets generated along with MPTCP options can be read-/written to a wire, allowing an NS-3 MPTCP stack to interact with other MPTCP stacks, such as a linux one.
Fallback	If the server does not answer with an MP_CAPABLE option, the client falls back to legacy TCP. Other failures are not handled, e.g. infinite mapping or MP_FAIL handling as simulating these features is of little interest.
Buffer space	Buffer space is not shared between subflows, data is replicated between the subflow and the meta send/receive buffers rather than moved.
Path management	We drifted away from the specifications in order to be able to identify a subflow specifically, i.e., we associate a subflow id to the combination of the IP and the TCP port. Nevertheless the implementation is modular so it is possible to replace the subflow id allocation with a standard scheme.

during the simulation. Indeed the ability to enable dynamically MPTCP on a per connection basis means that our implementation works with all the other TCP scripts. We focused our work on implementing the aspects that could have an impact on the performance such as how data is freed from the buffers: MPTCP requires the full mapping to be received before being able to free the buffer. We established a list of the key features of our implementation in table 7.3.

Compared to the linux implementation, a major shortcoming of the NS-3 mptcp implementation is the lack of the penalization mechanism reducing the window of a subflow that blocks the MPTCP window and the opportunistic retransmission feature. Like the Linux implementation and contrary to the two previous models, our scheduler follows the object oriented pro-

gramming principles, i.e., the user can develop and use its own scheduler without modifying NS-3. This is aligned with our intention to ease testing.

Also contrary to linux that generates DSS mappings just in time to be able to adapt to network conditions, we designed the scheduler to be able to delay the decision until the last minute or to create mappings in advance. Creating mappings in advance has the advantage of being able to generate mappings that cover several packets. While the throughput gain is negligible, it can spare some of the scarce TCP option space.

7.4 Evaluation

We chose not to run quantitative tests with the previous NS-3 implementations since they are based on NS-3 versions that date back from late 2009 for [325] (ns-3.6) and December 2013 for [326] (ns-3.19). This gap in versions make the practical evaluation a challenge as well as the interpretation of results as the NS-3 TCP implementation evolved a lot in the meantime. Conversely, we compare the linux version to our NS-3 stack in a simple but we believe fair setup. We looked for tools that would allow for seamless testing and analysis between the kernel and NS-3 stacks to lighten the analysis workload but could not find any. Hence we did some more development to unify the linux and NS-3 evaluation, leveraging on the standardized “pcap ”format. We first introduce these tools in 7.4.1 and 7.4.2.

7.4.1 Semantic analysis of MPTCP packet captures

As far as MPTCP signaling and data analysis is concerned, there is currently little choice with only one tool we are aware of: MPTCPTRACE [327]. MPTCPTRACE is interesting for bulk analysis but we wanted to be able to look at the packet level to ease debugging. Thus we chose to improve the MPTCP support of wireshark [226], which specializes in packet-level network protocol analysis. A capture is visible in fig. 7.2. We mainly added the following features:

- MPTCP connection identification: ability to map TCP subflows together based on the key and tokens respectively sent in the MP_CAPABLE and MP_JOIN options.
- Verification of the initial DSN based on the MPTCP key.

- Display relative DSN, i.e. the first MPTCP sequence number sent being considered as 0.
- Computation of the latency between the arrival of new data throughout all subflows.
- Detection of DSS mappings spanning several packets.
- Detected retransmissions across subflows.

We want to construct a data structure so that we can efficiently retrieve all intervals (i.e., DSS) overlapping another interval or point, to map TCP sequence numbers to DSNs (valid also for DACKs). Thus we have introduced augmented interval trees support in wireshark backend. Given a set of n intervals, queries require $O(\log n+m)$ time, with n being the total number of intervals and m being the number of reported results. Construction requires $O(nlogn)$ time, and storage requires $O(n)$ space.

Wireshark analyzes packets on the fly, as they are captured, hence instead of being able to count the number of packets beforehand and allocate the correct amount of memory we build the tree in the worst possible manner as packets mostly arrive in order and force successive rotations in order

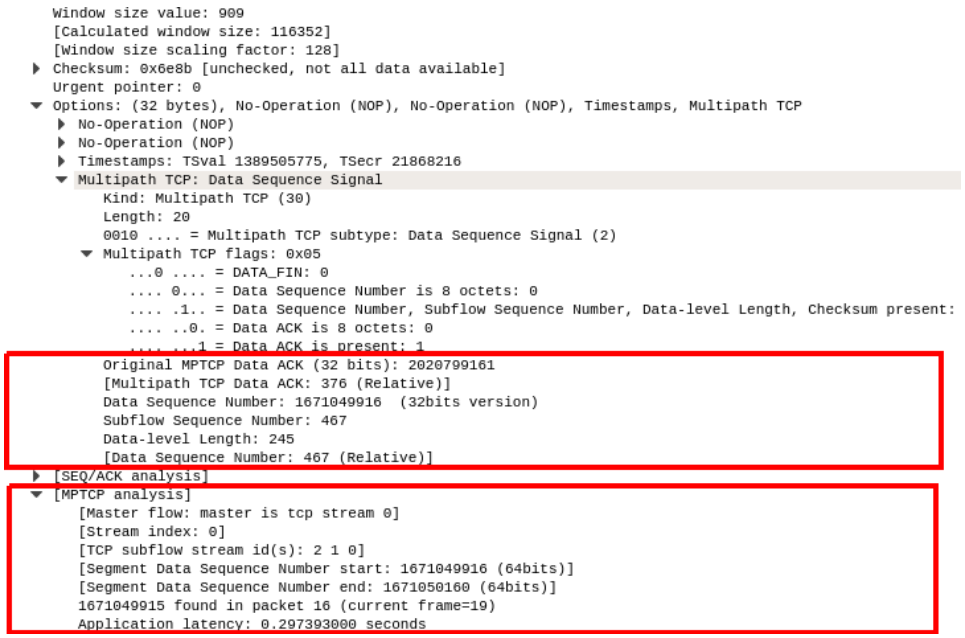

Figure 7.2: The new WIRESHARK MPTCP analysis section. Framed in red some of our additions.

to balance the tree. Compared to a linked-list, it means the structure construction is more expensive but the search should be faster.

7.4.2 Presentation of mptcpanalyzer

The linux kernel has a mechanism named OR composed of a static reinjection policy: it retransmits (i.e., *reinjects*) a segment on another subflow only if the connection is receive-window limited. This reinjected segment (several can be reinjected as well) acknowledgment may arrive before the initial one (because of a loss on the initial subflow or because the reinjections were made on a faster subflow), thus the sender may receive an ACK with a bigger advertised window sooner than without the reinjection. As a consequence, this reduces HoL occurrence in exchange of a little overhead. Reinjections occur for several reasons: in case of handover, excessive losses over one subflow or because the Opportunistic Retransmission (OR) [328] kicks in.

Improving the reinjection policy might be one of the most sensible way to improve MPTCP performance since it does not require any protocol change and directly addresses the HoL problem with non-negligible gains: Nikravesh, Guo, Qian, *et al.* [5] reached an average decrease in the download time of 10% when re-injecting segments on a subflow timeout and when the receive window exceeded a 75% threshold (chosen empirically). The paroxysm of reinjection is reached with the redundant scheduler described in Lopez, Aguado, Pinedo, *et al.* [329] which duplicates all segments on all paths to increase reliability. The lack of a tool capable of accurately analyzing the contribution of each subflow limits the research possibilities.

A better characterization of the different types of buffer blocking could help choose the previous threshold or devise when to reinject. Adhari, Dreibholz, Becke, *et al.* [311] established the following taxonomy of buffer blocking issues for SCTP, which also apply to MPTCP:

- Window-Induced Receiver Buffer Blocking corresponds to the phenomenon we just described.
- Reordering-Induced Receiver Buffer Blocking implies that packets remain in the receive buffer because they are out-of-order and can not be forwarded to the application and acknowledged.
- GapAck-Induced Sender Buffer Blocking is related to the Non-Renegotiable Selective Acknowledgments (NR-SACK) feature: NR-SACK is not

standardized for MPTCP even though a IETF draft [330] was proposed. GapAck-Induced Sender Buffer Blocking (GSB) occurs when the sender is send-window limited because of NR-SACK'ed data.

HoL can be a combination of these issues, and in order to investigate the events, we need ways to detect and visualize them, if possible in a way that works across OSs. Otherwise it is difficult to compute the contribution of each subflow to the global throughput as the reinjected bytes would count twice. Programs working on packet captures (i.e., *pcap* files) allow such analysis.

We thus developed a program called MPTCPANALYZER [331] to help in that regard; with MPTCP analysis, automate plotting of various parameters such as DSNs from *pcap* files. Contrary to MPTCPTRACE that does both the analysis and plotting of capture files, MPTCPANALYZER leverages as much as possible the analysis generated by WIRESHARK. As WIRESHARK is a widely used software, its dissection may be more trustworthy in the long term. As far as we are aware, there are very few applications that can measure OWDs, most notably OWAMP (Open source) [332] from the Internet2 project[10]. Yet all these applications do active measurements but we are not aware of any application that can for instance correlate packets of a single TCP connection, between two packet traces, and compute OWDs.

Passive measurements of OWDs allows to measure the parameters with real traffic contrary to OWAMP, which generates its own traffic. It would allow to distinguish useful reinjections from wasted ones as similar analysis is done by tstat [333] for RTOs. Hence we added this capability in MPTCPANALYZER.

We used MPTCPANALYZER to produce the plots presented in the next section 7.4.3.

7.4.3 Comparison with linux MPTCP on a 2-link topology

We present in the following a few simulations to compare the linux kernel running in DCE 1.7 to our NS-3 implementation. In order to minimize the differences due to the environment and for the ease of reproducibility, we chose to compare the linux and NS-3 MPTCP implementations within the DCE framework. This means that nodes, routers and links are created by

[10] `http://www.internet2.edu/`

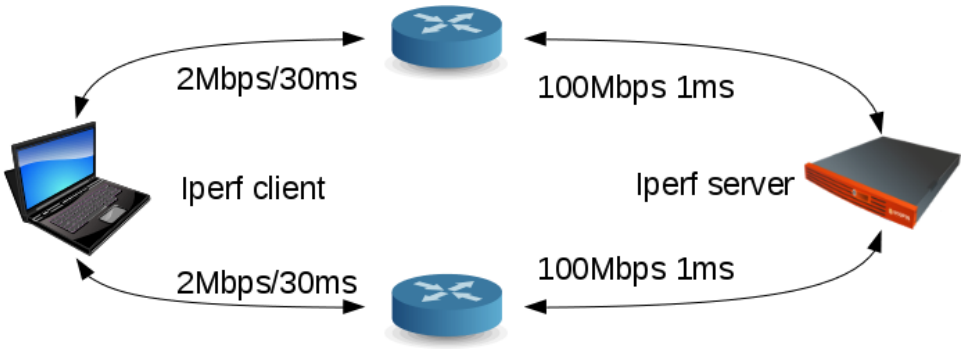

Figure 7.3: The topology used for the simulations. First hop is the 2Mbps bottleneck, with a variable propagation delay.

NS-3. Every node can be configured with a specific network stack. We always install linux stacks in the routers.

When running hybrid simulations, i.e. with both MPTCP Linux kernel and NS-3 stacks, the kernel will drop packets if the checksums are invalid so NS-3 should be configured to generate them or checksum validation should be disabled.

The BDP refers to the number of unacknowledged bytes that can be in flight. It is generally advised to set the BDP higher than $RTT*$bottleneck capacity to account for queuing delays in both the networks and the hosts. Note that in this case, as DCE runs in discrete time, kernel operations are virtually instantaneous if not programmed otherwise so only network latency impact the RTT. On one path with a bottleneck of 2Mbps and a RTT of 60ms, the BDP is 120kbits. We run the experiments with LibOS [320] applied against the linux mptcp kernel v0.89. Moreover:

- Scheduler is set to round robin.
- Number of paths is set to one (Figure 7.4), then two (Figure 7.5).
- Forward and backward one way delays are set to 30ms on each path.
- We launch several runs with different receiver windows.

We ran five seconds iperf2 [11] sessions between the two hosts without any background traffic on the topology of fig. 7.3.

[11] `http://iperf.sourceforge.net/`

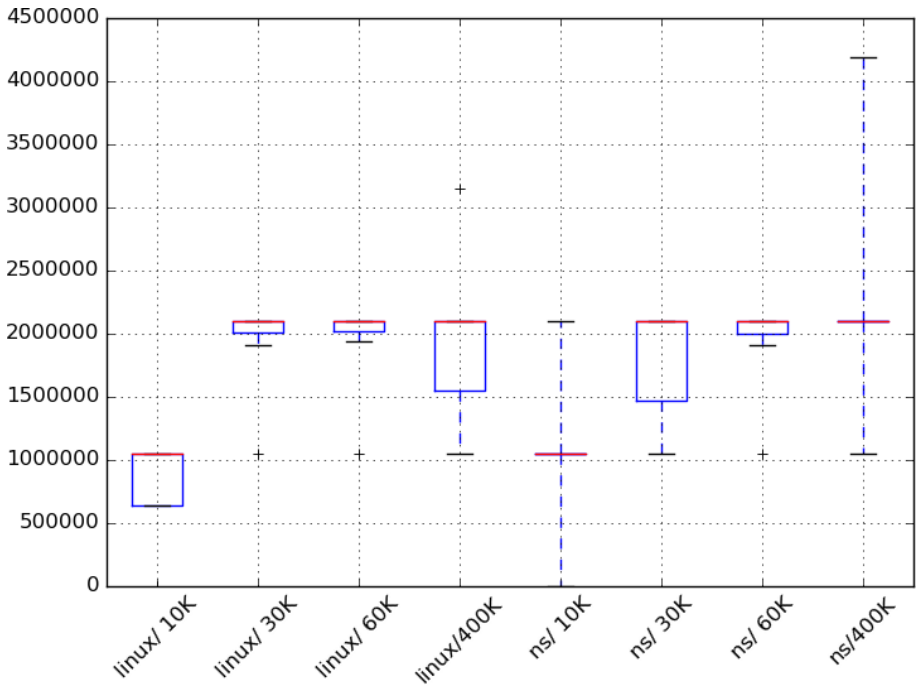

Figure 7.4: Results for the single path topology.

The size of the router buffers is the default one for the Linux version 3.X kernel, i.e., it starts at 87380 bytes up to (with Linux buffer autotuning, it can go up to 4MB).

On fig. 7.4, we notice that both stacks make the maximum use of the paths except when it is window limited as for the 10KB case. We can also notice that the throughput is a little more than the maximum throughput. We believe it is due to iperf2. Compared to the one path case, we can the expected doubling in throughput in the two paths case on fig. 7.5 when we add a path when the window is big enough. It also seems that the NS-3 version is greedier as in the 30KB window.

In order to check the behavior of the scheduler and thanks to mptcp-analyzer [226], we were able to plot the relative MPTCP sequence numbers transmitted on every subflow for a 40KB setup. We establish that DSNs are indeed sent in a round robin manner in both the linux (fig. 7.7) and the NS-3 cases (fig. 7.6). There are more sequence number for the NS-3 case because the throughput was higher for that setup.

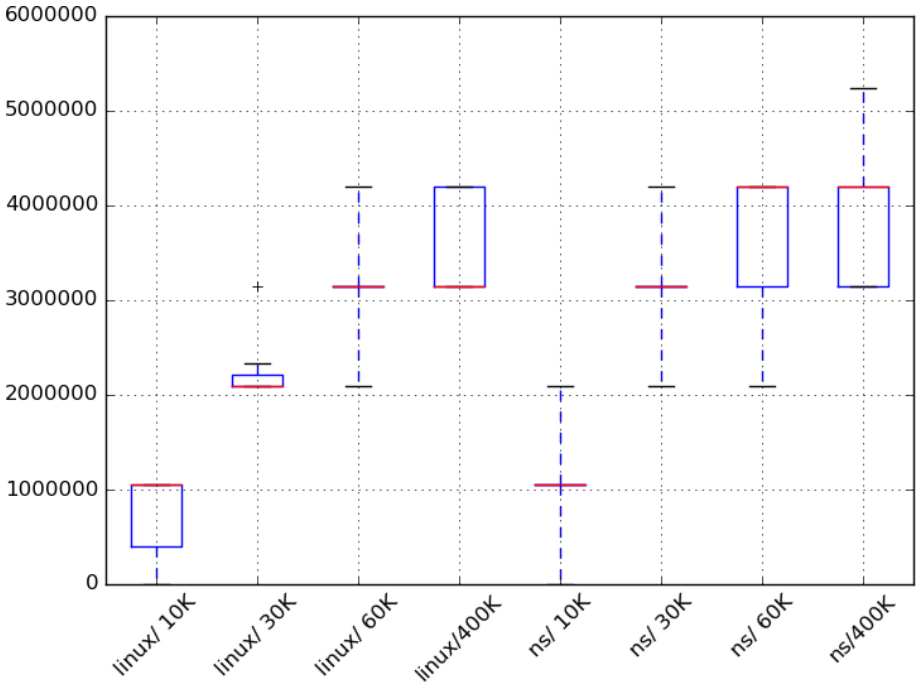

Figure 7.5: MPTCP Linux kernel and NS-3 iperf2 throughputs with two paths.

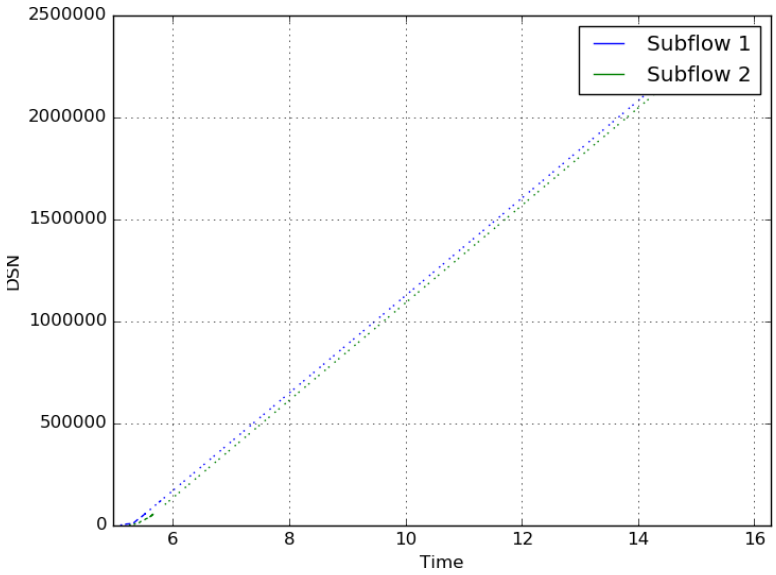

Figure 7.6: Repartition of sequence numbers across two subflows with the NS-3 round robin scheduler.

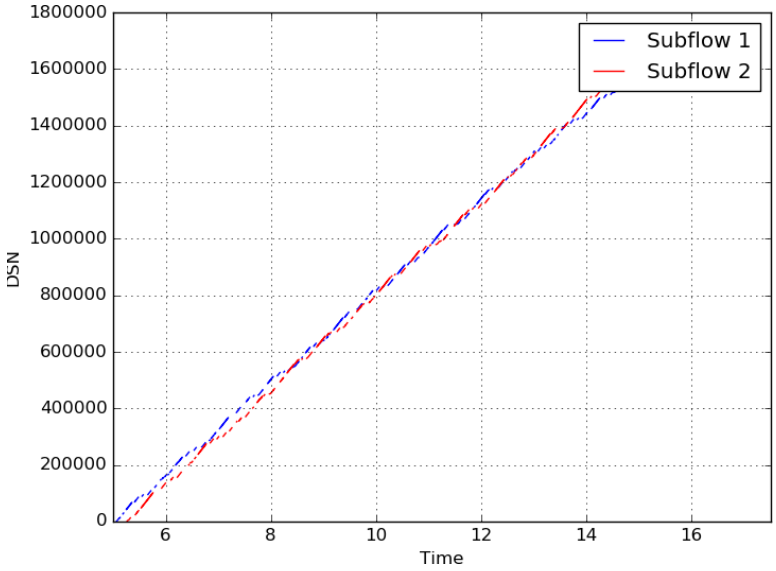

Figure 7.7: Repartition of sequence numbers across two subflows with the **linux** round robin scheduler.

7.4.4 Open Problems

The current buffer handling in NS-3 currently copies data back and forth between the subflows and the meta socket instead of sharing a pool of memory. This is the main difference with other implementations and could impact the simulation fidelity in tight buffer simulations. It is a problem we chose to notify to the NS-3 development team rather than fixing it ourselves as it implied many modifications across the NS-3 source code. One promising solution is NR-SACK [272]; sadly the source is not available and this would require NS-3 to implement SACK first.

7.5 Summary

We presented the MPTCP protocol and its new implementation in the NS-3 that conforms to many of the features described by the standard. The source code is available at `https://github.com/lip6-mptcp` and seemingly raised interest in the community considering the number of questions asked. Sharing with the research community also helped in finding and fixing bugs. We hope our effort will allow to develop new schemes in an easier way to improve or find new ways of using a multipath communication. MPTCP represents a subset of how multipath protocols could improve our future

communications and may represent a bridge between TCP and SCTP for instance. Indeed some developers may want to modify their application to better leverage MPTCP features, hopefully instead of doing so via a direct use of the MPTCP API, they could use the Transport Area Protocol Services (TAPS) API to painlessly switch afterwards to possibly more powerful multipath transport protocols, for instance SCTP.

The profusion of simulation frameworks and emulators results in a lot of duplicated work. Working with real stacks require an expertise and is time consuming as it involves low level details annex to the researcher consideration such as kernel *workqueues*, which explains why researchers may divert their attention to simpler models such as the ones in typical simulators. Nevertheless results obtained from real measurements are necessary to validate results obtained from models. Hence any solution that can help researchers quickly prototype and deploy their solutions is welcome. DCE or VMSimInt [321] are promising solutions in that regard. Another approach that is increasingly popular as more and more researchers work on the network stack is to move it from kernel to user space. While this approach would, there are concerns that it would favor the emergence of greedy congestion controls and endanger the Internet stability [12]. Reproducibility is paramount to good research and the high pace of development of network stacks may hinder this because of a change in the default configuration for instance or simply because of an error, hence it seems important to do inter-version testing for either simulators or kernels. Paasch et al. made a valuable contribution in applying experimental design [106] to test the Linux stack over a large combination of configurations (buffer size, delay, loss, etc) with MPTCP in Mininet. We hope these experiment could be ported to DCE, which would remove the bias seen for high loads in Mininet and also take into consideration the kernel version as a parameter.

[12] `https://lwn.net/Articles/691887/`.

Chapter 8

Conclusion and Perspectives

Network research is at the crossroad between different disciplines; mathematics, physics, computer science, politics. Its distributed nature makes backward compatibility a real challenge leading to suboptimal solutions in the technical sense. The scientist must reconcile these different approaches in a perspective similar to engineering. As an example of this irony, MPTCP was created due to middlebox interference and now, the technology itself has become the source of new middleboxes in order to benefit better from the path diversity. In the following, we critically revise our work and highlight future work that may derive from it.

8.1 Recollections

The initial goal of the thesis was to look for ways to improve cloud access with multipath protocols. Upon first analysis, TRILL, LISP, SCTP and MPTCP appeared as protocols easier to work with, thanks to an open standardization process at the IETF, contrary to for instance SPB. TRILL success seemed assured with constructors backing it up and a trusted creator Radia Perlman; LAN based protocols allowed for local testing and combined with Cloud environment raise interesting challenges in terms of latency or scaling but TRILL had no open source implementation (Gandi[1] now released a version to upstream in the Linux kernel). On the other hand, OpenLISP and LISPmob were available along with MPTCP Linux kernel. Hence we realized the LISP and MPTCP architecture described in Chapter 4. This

[1] `http://www.gandi.net/`

proved instructive in a sense that it made clear how hard crosslayer architectures are and why the concept of different layers was successful. It became clear that the gain should be high to justify a crosslayer implementation, and that it would be possible only in highly skilled environments. The LISP infrastructure was not very stable and this made our experiments difficult. Hopefully our LISP router was co-hosted with client hence we could to some extent understand what was going wrong but it also made clear that in a proper context with LISP running at the edge, it would be even more difficult. From a deployment perspective, it was also unclear how successful the protocol could be when major constructors refused to adopt it. In the end, we decided to focus on MPTCP since it seemed with better odds of being successful. Also the Pareto-optimality problem seemed also like a problem worth solving. In retrospective, it seems like a good choice as several companies have shown their interest in MPTCP for bandwidth aggregation mostly (first Proximus, then OVH, Ericsson...) The help from skilled researchers at Université Catholique de Louvain (UCL) also reassured researchers hesitating to study MPTCP and helped creating a community around the protocol. After the experiments 109, we chose to rely on simulators to do our work. The ability to have a deterministic outcome with the same inputs is really helpful in the prototyping phase. This can be done at the expense of realism though, which is the reason why the DCE approach is interesting: it reconciles both worlds and makes the transition between prototyping and experimentation easier.

8.1.1 Future Work

Concurrent multipath communications are challenging in many ways. There is room for improvements at several levels. Network coding is an active area of research, which could improve MPTCP characteristics [334], yet network coding can also be the source of issue: additional latency because of the computational and throughput overhead. Its performance depends on problems similar to what already affect the raw protocol: traffic pattern and scheduling. Hence even if network coding is promising, improvements have to be done at the transport layer as well. During our studies, while working on OWD estimation we got sidetracked at some point by the simulation of clock skew, which we would like to resume in the future as explained in Section 8.1.2. A strong reason for our work initial interest was to analyze

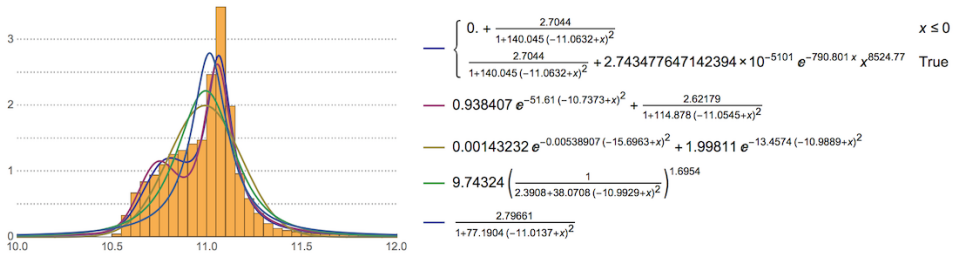

Figure 8.1: Mixture distributions approximations of a 1-month dataset of OWDs between two RIPE probes.

analytically the performance. This requires a deep understanding of the protocol inner working and data to choose the correct models and hypotheses . We present the reasons why we have not succeeded yet in Section 8.1.4.

8.1.2 Support of Time Distribution Protocols Scenarios in DCE

While working on Chapter 5, we realized the technique could also apply to time distribution protocols under certain conditions, specifically with multihomed servers. Network Time Protocol (NTP) as a protocol is mildly complex but the official NTP client is very complex, resorting both to mathematical theory mixed with computer science and specific techniques to improve the clock precision. Despite these efforts, in a 1997 survey [335], out of the 20.000 surveyed peers, 1200 were considered unsynchronized (with a phase offset superior 128ms). The synchronized set exhibited a median of 7,45m and a mean of 15,87ms. We tried to obtain detailed data in order to model OWD distribution: we could only find the Test Traffic Measurement (TTM) project which closed in 2014[2]. Through email exchanges, we have been able to write a script to download four months of raw data representing OWD measurements between dozens of pair of probes (16GBytes of data). We were able to clean the data for a month and approximate the OWD distribution with a few mixture distribution proposed by a configured Mathematica as shown in: A common hypothesis is to model latencies as a shifted gamma distribution [336], [337], yet the approximations in Figure 8.1 are not, further analysis of this data was then given up as it could justify a

[2] `https://www.ripe.net/analyse/archived-projects/ttm/ttm-data`

PhD by itself.

. When a NTP client receives a timestamp T1, it should adjust it to add the propagation delay and set its clock to $T1 + f1$, but it can not measure it and assumes $f1 = 0.5 * RTT$ hence the NTP client updates its clock with the value $T1 + \frac{RTT}{2/}$ (the real algorithm runs many algorithms, this is a simplified approach).

Under the assumption that packets are sent at the same time from the NTP peer, intra-subflow RTT variations could be compared with the inter-subflow Δ_{OWD} variations to deduce if the variations in RTT were due to the forward or backward congestion. For instance if both NTP hosts are connected via two paths and the RTT of the fast path increases while the slow path RTT remains constant as well as their $\Delta_F OWD$, then it means the congestion happened on the backward path and the NTP client, instead of assuming setting the estimation of the forward propagation delay to 40% of the RTT instead of the presently static 50%.

Time distribution experiments are very demanding since you can only measure your error with the precision of your best clock, most likely a hardware clock. This hardware clock will most likely need to be instrumented and connected to a computer hence care must be taken not to lose precision because of the physical channel or because of the software.

Hence we though it could be of interest to have perfectly reproducible NTP experiments via the use of discrete time simulators. Such simulations require that nodes can run their own clock, which for instance is not the case in NS-3 where the virtual time is exactly the same for each node.

As a consequence we added per node clocks in NS-3 which raised some interest in the IoT community. Indeed in order to save energy, the IoT community was looking forward to schedule wake up timers before putting nodes to sleep. On real hardware, clock skew generate problems since nodes may wake up at different times even with their alarms set to the same time. Without our modifications, upstream NS-3 would wake up the nodes at the exact same time, hiding the problem.

As a second step, we proceeded to let NTP run in DCE. The interaction between the two softwares exhibited a few interesting aspects of NTP. For instance, in order to deduce the resolution of the host clock, the ntp client would request several times in a loop until some precision can be deduced. As syscalls in DCE are virtually instantaneous, this check would fail and the

NTP client abort. It also exhibited the limit of the DCE approach consisting in reimplementing kernel interfaces: the NTP client client expected information conveyed by the linux kernel that DCE did not disclose. The final step to allow for NTP simulations would have been to add the ntp-related syscalls to DCE, such as *ntp_adjtime* or *adjtimex*. The obvious choice is to forward those calls to a real kernel not to duplicate code. The linux kernel maintains a 'jiffy' counter which is currently kept in synchronization with the simulator time via DCE. One of the challenge was to change this direct update of the counter to let the syscalls update it as it is done on real systems. To achieve this, DCE mainly needed to mimic two linux components responsible for timekeeping: Linux maintains a counter called *jiffies* that is governed by two entities: a hardware clock called *clocksource* in charge of remembering the epoch, and a *clockevent* device in charge of generating regular interrupts to update the jiffie counter. The jiffie counter used to have a 10ms precision but in recent linux kernels, the default is set to 1ns. Vanilla DCE bypasses those previous mechanisms to keep the linux jiffie counter perfectly synchronized with the NS-3 clock, which prevents any skew to propagate to the kernel. Our proposition is to let the linux jiffie counter be updated by the conventional means so that NTP adjustments can also affect the jiffie counter via calls to the kernel function *xtime_update*, function in charge of applying the NTP correction. One of the problems though is that *xtime_update* accepts as input the elapsed jiffies since the last update and if it exceeds a certain threshold then the nominal NTP algorithms are not used. One solution we did not implement should be to call this function several times with a number of jiffies inferior to the threshold.

Even if we could not run NTP with the NTP kernel disciplines, we published the work accomplished thus far in [338] and hope to complete this final missing block later.

8.1.3 Improving sockets Application Programming Interface

As engineers and researchers try to optimize applications, there is a clear demand on having better information on the traffic patterns: is the application latency rather than throughput sensitive ? Should the service occupy only the unused bandwidth (lower than best effort as depicted in Low Extra Delay Background Transport (LEDBAT) [339]) ? Is there any interface preference ? Is the connection short lived or long lived ?

A list of suggested characterizations has been proposed by Schmidt, Enghardt, Khalili, *et al.* [340] under the name of 'socket intents'. If we reminisce the problems mentioned throughout the thesis, it is easy to understand how to exploit this information. One of the MPTCP problem is to distinguish between short lived and long lived subflows to prevent MPTCP from wasting resources on unused subflows (especially relevant with cellular interfaces as shown in [5]). For latency sensitive applications, it would allow to configure the MPTCP stack with more aggressive reinjection policies, or even allow for full duplication over several subflows of the DSN. Information on interface preferences could be fed into MPTCPNUMERICS to help with traffic scheduling.

Such information is currently obtained either via predefined configuration policies (Full Qualified Domain Name (FQDN)/port lists for instance) or reactive solutions based on heuristics such as mahout [242] but it has become very relevant in multiple access connectivity environments.

Technologies such as SCTP, websockets, MPTCP extend the set of available transport services. However application programmers often face difficulties when they use protocols other than TCP or UDP. An IETF group has been formed around to tackle this problem: the Transport Area Protocol Services (TAPS) working group [341] intends to make the use of alternate transport protocols easier.

This is very likely to require application programmers to use a richer API than the Berkeley socket-like APIs exposed by every major OS. Berkeley sockets have been little changed since the time they were devised, in the nineties. Thinking of a wrapper or extensions to the API to take into account the effective outbreak of multipath communications seems reasonable but choosing the correct abstractions is crucial and challenging.

8.1.4 Better Theoretical Models

This is one of the most important point in our opinion since experimentation can not prove but only confirm or refute hypotheses. The variants of TCP are numerous, each variant performing strongly in an environment but rarely in an optimal way, thus very few are used in practice. The proliferation of schedulers and congestion control is a possibility for MPTCP as the space of states is even bigger. We believe it is important to compare the new scheduler or congestion controls over the full states and pay attention to

the Pareto-optimality aspect. There has been relatively few attempts at modeling the throughput of concurrent multipath transport protocol. We mainly list Wallace and Shami [342] who models the CMT-SCTP throughput via 2 methods: a Markov model and renewable theory.

Still for SCTP, Sarkar [343] propose a Markov model to estimate the expected window size on each path depending on a loss rate, then deduces the datarate and sums up the datarates across paths to compute the communication throughput. As for MPTCP, [344] studies the impact of the choice in the first subflow over the rest of the communications in the 2-subflow case and compares the result with Mininet. Peng, Walid, and Low [345] use fluid modeling to model the behavior of MPTCP and test the results against NS-2, which has a wrong MPTCP model, basically running several TCP flows in parallel with no mapping or coupled congestion control.

The difficulty of modeling CMT can be a reason for having little literature. TCP modeling already is complex to model as can be seen in the seminal paper [304]. A good model needs to take into account fast retransmissions, RTOs, and the maximum possible window for throughput modeling. Depending on the lossrate of the link, some hypotheses become invalid (a lossrate close to 0 is simplified in several models) and require another approach. The model is also dependent on the chosen congestion control. Taking into account the RTT variance is another challenge. To analyze short connections, models have to consider the slow start instead or complementary to the congestion avoidance state as is done in Mellia, Stoica, and Zhang [346].

With CMT, there are even more variables like the number of subflows, the scheduling strategy, the coupling between windows. Previous works compute the throughput but as shown in [5], [102], it is clear that modeling reinjections or even NR-SACK [272] matters hence one should model goodput rather than throughput to find ways to optimize the duplication of segments for MPTCP. It might be interesting to devise more metrics in order to compare the different variants of CMTs, for instance to improve the stability of the throughput when a subflow is lost.

Even as CMT modeling progresses, experimentation remains mandatory and approaches such as the experimental design applied in [106] allows to cover many networks configurations and see the influence of each variable on the final result.

8.2 Conclusion

People directly or indirectly concerned by the maintenance of the Internet have recently been facing an increasing number of problems: the transition from Internet Protocol version 4 (IPv4) to IPv6, the ever growing number of routes exchanged in the Domain Free Zone (DFZ), the bufferbloat,... The urgency of some problems has started to overcome the reluctance of addressing these problems, due to the fear of breaking a complex and unpredictable network such as the Internet. For instance, the bufferbloat project [3] has lead to the successful deployment of the Codel queuing discipline in millions of Linux hosts, Apple has enabled ECN by default on its systems. The ossification of the Network does not seem as a fatality anymore. The experience gained along the years allowed the networking community to realize, advertise and partially or fully address problems related to multipath communications. The deployment problem known with SCTP helped design MPTCP in a more backward compatible way.

The Happy Eyeballs mechanism thought initially for the IPv4 to IPv6 transition has also been applied to SCTP as a solution to probe for the peer SCTP support and fallback or legacy TCP for instance. Using disjoint paths is an interesting possibility for CMT, with a direct benefit on reliability and confidentiality, yet it can not be achieved without some sort of cooperation with the network as hosts ignore the path diversity between them. Some practical propositions exist to distribute hints about the path diversity via DHCP or some oracle but even so endhosts have little control over the forwarding. Hence choosing the number of subflow with the hope of creating disjoint subflows remains an open problem. This decision is made even harder when taking into consideration buffer constraints. Indeed TCP flow control puts a limit on the number of subflows that can be used concurrently to send data. This limit is hard to find as network characteristics evolve, and depend on many parameters such as the traffic pattern and the scheduler. Traffic pattern is indeed valuable information, even more than in single path communications since you have more possibilities to deal with different patterns. A latency-sensitive duplication could choose to fully or partially duplicate segments on several paths. While heuristics exist to find this information, having this explicit could be an improvement, especially

[3] `https://www.bufferbloat.net/`

for short flows. The drawback is that it most likely requires an upgrade of applications but this can also be considered as an opportunity to improve applications even further. In this regard, the TAPS IETF working group proposes to abstract network services via a new socket APIs. The application would detail a few networking-related characteristic and the underlying mechanism would choose the appropriate network protocol and protocol options that fit the features. TAPS would allow to improve network performance while shifting most of the burden off the application developers to the OS, making the evolution of services easier and globally customizable. A host could configure Happy Eyeballs (HE) or tunneling according to the policy in place. Also MPTCP might become widely adopted considering the current industrial enthusiasm, but it is a subset of SCTP feature-wise with an emphasis on ease of deployment. Relaxing some constraints such as the ordered delivery makes sense for bulk transfers and relying on a TAPS API could provide a smoother transition from single path communications to CMT.

Own publications

International Journals with peer review

- M. Li, A. Lukyanenko, Z. Ou, A. Yla-Jaaski, S. Tarkoma, M. Coudron, S. Secci, "Multipath Transmission for the Internet: A Survey", *IEEE Communications Survey & Tutorials*, Vol. 18, No. 4, pp: 2887-2925, Dec. 2016.

International conferences with peer review

- M. Coudron, S. Secci, G. Pujolle, P. Raad, P. Gallard, "Cross-layer Cooperation to Boost Multipath TCP Performance in Cloud Networks", in *Proc. of 2013 IEEE Int. Conference in Cloud Networking (CLOUDNET 2013)*, Nov. 11-13, 2013, San Francisco, USA.

- M. Coudron, S. Secci, G. Maier, G. Pujolle, A. Pattavina, "Boosting Cloud Communications Through A Crosslayer Multipath Protocol Architecture", in *Proc. of 2013 IEEE Worskshop on Software Defined Networks for Future Networking Services (IEEE SDN4FNS 2013)*, Nov. 11-13, 2013, Trento, Italy.

- M. Coudron, S. Secci, G. Pujolle, "Augmented Multipath TCP Communications", in *Proc. of 2013 IEEE Int. Conference on Network Protocols (ICNP 2013)*, Poster session, Oct. 7-11, 2013, Gottingen, Germany. Peer review.

- M. Coudron, G. Pujolle, S. Secci, "Differentiated Pacing on Multiple Paths to Improve One Way Delay Estimations", in *Proc. of 2015 IEEE/IFIP International Symposium on Integrated Network Management (IEEE/IFIP IM 2015)*, May 11-15, 2015, Ottawa, Canada.

- M. Coudron, S. Secci, "Per Node Clocks to Simulate Clock Desynchronization in ns-3", *2016 Workshop on Network Simulator 3 (WNS3 2016)*, June 15-16, 2016, Seattle, USA.

- M. Coudron, D. Nguyen, S. Secci, "On buffer and window management for Multipath TCP", *Proc. of 2016 Int. Conference on the Network of the Future (NOF 2016)*, Buzios, Brazil, Nov. 14-16, 2016.

Submitted

- M. Coudron, S. Secci, "MPTCP in NS3: Implementation Evaluation", submitted to Computer Networks, major revision.

Software contributions

To paraphrase the Mininet authors "a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures." [315]. This is key to reaffirm trust in scientific results. The access to datasets is also very valuable but there are many understandable reasons, among which confidentiality, which make it more difficult to share. We faced early on the problems of replicating experiments. After studying some articles with interesting results, we wanted to extend or improve the mechanisms at work. Yet often, the source code is not shared and re-implementing from scratch represents too high a cost.

Too often we wanted to build upon the work of others either to examine how they processed their data or simply to compare our solution with previous solutions and were unable to. In order not to create a similar frustration, we open sourced all the source code used in our experiments with the hope they get refined further and ease the work of other researchers. Over the past years, we have invested a lot of effort to upstream our changes in their original software. This increases reproducibility as well as democratize the analysis to a broader scope then academic research.

Our main contributions either merged in their original codebase or available as standalone:

- Semantic support of Multipath TCP in WIRESHARK (`www.wireshark.org`), i.e., it makes wireshark capable of mapping TCP subflows to their respective MPTCP connections.

- MPTCPANALYZER: a program to generate plots based on MPTCP packet capture files (`https://github.com/lip6-mptcp/mptcpanalyzer`).

- Our work on MPTCP support in NS-3 is available `https://github.com/lip6-mptcp/ns3mptcp` and is being reworked for inclusion in the main simulator.
- The entire source code used for the experimentation in **??** is available at `https://github.com/lip6-mptcp/mptcpnetlink`. This includes modifications made to the LISPmob router, the kernel modifications, the kernel module acting as path-manager and finally the userspace daemon in charge of communicating with the LISP router.
- MPTCPNUMERICS generates an ILP from a network configuration and solves it to find optimal congestion window targets or buffer sizes.

We have also several pending contributions made to the software either to fix bugs (as in the libnl netlink library) or to provide additional capabilities.

Bibliography

[1] S. Agarwal, C. N. Chuah, and R. H. Katz, "OPCA: Robust interdomain policy routing and traffic control", *2003 IEEE Conference on Open Architectures and Network Programming Proceedings, OPENARCH 2003*, pp. 55–64, 2003.

[2] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From Paris to Tokyo", in *Proceedings of the 2013 conference on Internet measurement conference - IMC '13*, 2013, pp. 427–432.

[3] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 address allocation and the BGP routing table evolution", *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, p. 71, 2005.

[4] R. Khalili, N. Gast, M. Popovic, and J.-y. Le Boudec, "MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution", *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1651–1665, 2013.

[5] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An in-depth understanding of multipath TCP on mobile devices", in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking - MobiCom '16*, New York, New York, USA: ACM Press, 2016, pp. 189–201.

[6] M. Kun, Y. Jingdong, and R. Zhi, "The research and simulation of multipath-OLSR for mobile ad hoc network", in *IEEE International Symposium on Communications and Information Technology (ISCIT)*, IEEE, vol. 1, 2005, pp. 540–543.

[7] X. Huang and Y. Fang, "Multiconstrained QoS multipath routing in wireless sensor networks", *Wireless Networks*, vol. 14, no. 4, pp. 465–478, 2008.

[8] I. Van Beijnum, J. Crowcroft, F. Valera, and M. Bagnulo, "Loop-freeness in multipath BGP through propagating the longest path", in *IEEE International Conference on Communications (ICC) Workshops*, 2009, pp. 1–6.

[9] K. Sha, J. Gehlot, and R. Greve, "Multipath Routing Techniques in Wireless Sensor Networks: A Survey", *Wireless Personal Communications*, vol. 70, no. 2, pp. 807–829, 2013.

[10] J. Qadir, A. Ali, Y. Kok-Lim, A. Sathiaseelan, and J. Crowcroft, "Exploiting the power of multiplicity: a holistic survey of network-layer multipath", *IEEE Communications Surveys Tutorials*, 2015.

[11] M. Li, A. Lukyanenko, Z. Ou, A. Yla-Jaaski, S. Tarkoma, M. Coudron, and S. Secci, "Multipath Transmission for the Internet: A Survey", *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2016.

[12] K. Leonard, "An early history of the Internet ", *IEEE Communications Magazine*, vol. 48, no. 8, pp. 26–36, 2010.

[13] "IEEE Standard for Local and Metropolitan Area Networks - Link Aggregation", *IEEE Std 802.1AX-2008*, pp. c1–145, 2008.

[14] "Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks–amendment 20: Shortest path bridging", *IEEE Std 802.1aq-2012*, pp. 1–340, 2012.

[15] C. Huitema, "Multi-homed TCP", *IETF Internet-Draft (Expired)*, 1995.

[16] A Ford, C Raiciu, M Handley, S Barre, and J Iyengar, "Architectural guidelines for multipath {TCP} development", *RFC6182 (March 2011), www. ietf. ort/rfc/6182*, 2011.

[17] D. Farinacci, D. Lewis, D. Meyer, and V. Fuller, "The locator/ID separation protocol (LISP)", *IETF RFC 6830*, 2013.

[18] D Eastlake, M Zhang, A Ghanwani, V Manral, and A Banerjee, "Transparent Interconnection of Lots of Links (TRILL): Clarifications, Corrections, and Updates", *IETF RFC 7180, May*, 2014.

[19] W. Lei, W. Zhang, and S. Liu, "A Framework of Multipath Transport System Based on Application-Level Relay (MPTS-AR)", *IETF Internet-Draft (Experimental), July*, 2015.

[20] M. Menth, A. Stockmayer, and M. Schmidt, "LISP Hybrid Access", *Draft-menth-lisp-ha-00 (Experimental), Internet-Draft, IETF*, 2015.

[21] P. Amer, M. Becke, T. Dreibholz, N. Ekiz, J. Iyengar, P. Natarajan, R. Stewart, and M. Tuexen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)", *Draft-tuexen-tsvwg-sctp-multipath-10 (Experimental), Internet-Draft, IETF*, 2015.

[22] M. Cullen, N. Leymann, C. Heidemann, M. Boucadair, H. Deng, and B. Sarikaya, "Problem Statement: Bandwidth Aggregation for Internet Access", *Draft-zhang-banana-problem-statement-01 (Informational), Internet-Draft, IETF*, 2015.

[23] Cisco, "EtherChannels", 2003, `http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3550/software/release/12-1_13_ea1/configuration/guide/3550scg/swethchl.html` [Available Online].

[24] Juniper, "Aggregated Ethernet", 2014, `http://www.juniper.net/documentation/en_US/junos14.1/topics/task/configuration/link-aggregation-cli.html` [Available Online].

[25] AVAYA, " Multi-Link Trunking", 2011, `https://downloads.avaya.com/css/P8/documents/100134063` [Available Online].

[26] Deutsche Telekom, "Hybrid Access - a promising approach to increase bandwidth of DSL-lines", 2014, `http://www.laboratories.telekom.com/public/english/newsroom/news/pages/hybrid-access.aspx` [Available Online].

[27] "Hybrid Internet Access Bonding", `http://www.tessares.net` [Available Online].

[28] OVH company, "Overthebox", 2015, `https://www.ovhtelecom.fr/overthebox` [Available Online].

[29] "iOS: Multipath TCP Support in iOS 7", 2015, `https://support.apple.com/en-us/HT201373` [Available Online].

[30] O Bonaventure, "In Korean, Multipath TCP is pronounced GIGA Path", 2015, `http://blog.multipath-tcp.org/blog/html/2015/07/24/korea.html` [Available Online].

[31] S. Singh, T. Das, and A. Jukan, "A Survey on Internet Multipath Routing and Provisioning", *IEEE Communications Surveys Tutorials*, 2015.

[32] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, "On load distribution over multipath networks", *IEEE Communications Surveys & Tutorials*, vol. 14, no. 3, pp. 662–680, 2012.

[33] A. L. Ramaboli, O. E. Falowo, and A. H. Chan, "Bandwidth aggregation in heterogeneous wireless networks: A survey of current approaches and issues", *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1674–1690, 2012.

[34] J. Domżał, Z. Duliński, M. Kantor, J. Rzasa, R. Stankiewicz, K. Wajda, and R. Wójcik, "A survey on methods to provide multipath transmission in wired packet networks", *Computer Networks*, vol. 77, pp. 18–41, 2015.

[35] S. Addepalli, H. G. Schulzrinne, A. Singh, and G. Ormazabal, "Heterogeneous Access: Survey and Design Considerations", *Columbia University Academic Commons, Technical Report*, 2013, `http://dx.doi.org/10.7916/D8QJ7F8P`.

[36] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP Multilink Protocol (MP)", *IETF RFC 1990*, 1996.

[37] A. C. Snoeren, "Adaptive inverse multiplexing for wide-area wireless networks", in *Proceedings of the Global Telecommunications Conference (GLOBECOM '99)*, IEEE, vol. 3, 1999, pp. 1665–1672.

[38] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol", in *Proceedings of the Applications, technologies, architectures, and protocols for computer communications*, ACM, vol. 26, 1996, pp. 131–141.

[39] A. Hari, G. Varghese, and G. Parulkar, "An Architecture for Packet-striping Protocols", *ACM Transactions on Computer Systems (TOCS)*, vol. 17, no. 4, pp. 249–287, 1999.

[40] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi, "FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput", in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, vol. 8, 2008, pp. 89–104.

[41] T. Nguyen-Duc, H. Tran-Viet, K. Nguyen, Q. T. Minh, S. H. Ngo, and S. Yamada, "Investigating the Performance of Link Aggregation on OpenFlow Switches", in *Testbeds and Research Infrastructure: Development of Networks and Communities*, Springer, 2014, pp. 194–202.

[42] T. N. Subedi, K. K. Nguyen, and M. Cheriet, "OpenFlow-based in-network Layer-2 adaptive multipath aggregation in data centers", *Computer Communications*, vol. 61, pp. 58–69, 2015.

[43] D Fedyk, P Ashwood-Smith, D Allan, A Bragg, and P Unbehagen, "IS-IS Extensions Supporting IEEE 802.1 aq Shortest Path Bridging", *IETF RFC 6329*, 2012.

[44] D. S. Phatak and T. Goff, "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments", in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02)*, vol. 2, 2002, pp. 773–781.

[45] D. S. Phatak, T. Goff, and J. Plusquellic, "IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping", *Computer Networks*, vol. 43, no. 6, pp. 787–804, 2003.

[46] K. Chebrolu and R. R. Rao, "Bandwidth aggregation for real-time applications in heterogeneous wireless networks", *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 388–403, 2006.

[47] K. Chebrolu, B. Raman, and R. R. Rao, "A network layer approach to enable TCP over multiple interfaces", *Wireless Networks*, vol. 11, no. 5, pp. 637–650, 2005.

[48] K.-H. Kim and K. G. Shin, "Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts", in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, ACM, 2005, pp. 107–120.

[49] ——, "PRISM: improving the performance of inverse-multiplexed TCP in wireless networks", *IEEE Transactions on Mobile Computing*, vol. 6, no. 12, pp. 1297–1312, 2007.

[50] K.-c. Lan and C.-Y. Li, "Improving TCP performance over an on-board multi-homed network", in *IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2012, pp. 2961–2966.

[51] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee, "MAR: A commuter router infrastructure for the mobile Internet", in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, ACM, 2004, pp. 217–230.

[52] K. Manousakis and D. Famolari, "INTELiCON: A Framework for the Simultaneous Utilization of Multiple Interfaces and its Application on TCP", in *International Wireless Communications and Mobile Computing Conference, 2008 (IWCMC '08)*, 2008, pp. 976–981.

[53] K. Evensen, D. Kaspar, P. Engelstad, A. F. Hansen, C. Griwodz, and P. Halvorsen, "A network-layer proxy for bandwidth aggregation and reduction of IP packet reordering", in *Proceedings of the IEEE 34th Conference on Local Computer Networks (LCN)*, IEEE, 2009, pp. 585–592.

[54] S. Pierrel, P. Jokela, and J. Melen, "Simultaneous Multi-Access extension to the Host Identity Protocol", *Draft-pierrel-hip-sima-00 (work in process), Internet-Draft, IETF*, 2006.

[55] A. Gurtov and T. Polishchuk, "Secure multipath transport for legacy Internet applications", in *Proceedings of the 6th International Conference on Broadband Communications, Networks, and Systems*, 2009, pp. 1–8.

[56] T. Polishchuk and A. Gurtov, "Improving TCP-friendliness and Fairness for mHIP", *Inforcommunications Journal*, vol. 3, no. 1, 2011.

[57] J. Sun, Y. Wen, and L. Zheng, "On file-based content distribution over wireless networks via multiple paths: Coding and delay tradeoff", in *Proceedings of the IEEE INFOCOM*, 2011, pp. 381–385.

[58] K. Habak, K. A. Harras, and M. Youssef, "OSCAR: A Collaborative Bandwidth Aggregation System", *ArXiv preprint arXiv:1401.1258*, 2014.

[59] A. Argyriou and V. Madisetti, "Bandwidth aggregation with SCTP", in *Proceedings of the 2003 IEEE Global Telecommunications Conference (GLOBECOM '03)*, IEEE, vol. 7, 2003, pp. 3716–3721.

[60] C Casetti and W Gaiotto, "Westwood SCTP: load balancing over multipaths using bandwidth-aware source scheduling", in *Proceedings of the 2004 Vehicular Technology Conference (VTC '04)*, IEEE, vol. 4, 2004, pp. 3025–3029.

[61] J. Iyengar, K Shah, P Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming", in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '04)*, 2004.

[62] J. R. Iyengar, P. D. Amer, and R. Stewart, "Retransmission policies for concurrent multipath transfer using SCTP multihoming", in *Proceedings of the 12th IEEE International Conference on Networks (ICON '04)*, IEEE, vol. 2, 2004, pp. 713–719.

[63] ——, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths", *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, 2006.

[64] J. Liu, H. Zou, J. Dou, and Y. Gao, "Rethinking Retransmission Policy In Concurrent Multipath Transfer", in *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP '08)*, IEEE, 2008, pp. 1005–1008.

[65] T. Dreibholz, M. Becke, E. P. Rathgeb, and M Tuxen, "On the use of concurrent multipath transfer over asymmetric paths", in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '10)*, 2010, pp. 1–6.

[66] T. Dreibholz, R Seggelmann, M Tuexen, and E. Rathgeb, "Transmission scheduling optimizations for concurrent multipath transfer", in *Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT '10)*, vol. 8, 2010.

[67] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Applying TCP-friendly congestion control to Concurrent Multipath Transfer", in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10)*, IEEE, 2010, pp. 312–319.

[68] H. Adhari, T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "Evaluation of concurrent multipath transfer over dissimilar paths", in *IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA '11)*, 2011, pp. 708–714.

[69] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, "On the impact of congestion control for Concurrent Multipath Transfer on the transport layer", in *Proceedings of the 11th IEEE International Conference on Telecommunications (ConTEL '11)*, 2011, pp. 397–404.

[70] S. Shailendra, R Bhattacharjee, and S. K. Bose, "Improving congestion control for Concurrent Multipath Transfer through bandwidth estimation based resource pooling", in *Proceedings of the 8th International Conference on Information, Communications and Signal Processing (ICICS '11)*, IEEE, 2011, pp. 1–5.

[71] A. Abd El Al, T. Saadawi, and M. Lee, "LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol", *Computer Communications*, vol. 27, no. 10, pp. 1012–1024, 2004.

[72] A. Abd, T. Saadawi, and M. Lee, "Improving throughput and reliability in mobile wireless networks via transport layer bandwidth aggregation", *Computer Networks*, vol. 46, no. 5, pp. 635–649, 2004.

[73] D. Sarkar, "A Concurrent Multipath TCP and Its Markov Model", in *Proceedings of the IEEE International Conference on Communications (ICC '06)*, 2006, pp. 615–620.

[74] C.-M. Huang and C.-H. Tsai, "WiMP-SCTP: Multi-path transmission using stream control transmission protocol (SCTP) in wireless networks", in *Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications Workshops (AINAW '07)*, vol. 1, 2007, pp. 209–214.

[75] J. Liao, J. Wang, and X. Zhu, "cmpSCTP: An extension of SCTP to support concurrent multi-path transfer", in *Proceedings of the IEEE International Conference on Communications (ICC '08)*, 2008, pp. 5762–5766.

[76] L. Budzisz, R. Ferrús, F. Casadevall, and P. Amer, "On concurrent multipath transfer in SCTP-based handover scenarios", in *Proceedings of the IEEE International Conference on Communications (ICC '09)*, 2009, pp. 1–6.

[77] F. H. Mirani, N. Boukhatem, and M. A. Tran, "A data-scheduling mechanism for multi-homed mobile terminals with disparate link latencies", in *Proceedings of the 72nd IEEE Vehicular Technology Conference Fall (VTC '10)*, IEEE, 2010, pp. 1–5.

[78] L. Magalhaes and R. Kravets, "Transport level mechanisms for bandwidth aggregation on mobile hosts", in *Proceedings of the 9th IEEE International Conference on Network Protocols*, 2001, pp. 165–171.

[79] Y. Lee, I. Park, and Y. Choi, "Improving TCP performance in multipath packet forwarding networks", *Journal of Communications and Networks*, vol. 4, pp. 148–157, 2002.

[80] H.-Y. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts", in *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MobiCom '02)*, ACM, 2002, pp. 83–94.

[81] ——, "pTCP: An end-to-end transport layer protocol for striped connections", in *Proceedings of the 10th IEEE International Conference on Network Protocols*, 2002, pp. 24–33.

[82] ——, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts", *Wireless Networks*, vol. 11, no. 1-2, pp. 99–114, 2005.

[83] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar, "A Receiver-centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces", in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom '03)*, 2003, pp. 1–15.

[84] K.-H. Kim, Y. Zhu, R. Sivakumar, and H.-Y. Hsieh, "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces", *Wireless Networks*, vol. 11, no. 4, pp. 363–382, 2005.

[85] C. Cetinkaya and E. W. Knightly, "Opportunistic traffic scheduling over multiple network paths", in *Proceedings of the 23td Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*, IEEE, vol. 3, 2004, pp. 1928–1937.

[86] M. Zhang, J. Lai, A. Krishnamurthy, L. L. Peterson, and R. Y. Wang, "A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths", in *Proceedings of the General Track: USENIX Annual Technical Conference*, 2004, pp. 99–112.

[87] J. Chen, K. Xu, and M. Gerla, "Multipath tcp in lossy wireless environment", in *Proceedings of IFIP 3rd Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net '04)*, 2004, pp. 263–270.

[88] K. Rojviboonchai and A. Hitoshi, "An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes", *IEICE transactions on communications*, vol. 87, no. 9, pp. 2699–2707, 2004.

[89] K. Rojviboonchai, T. Osuga, and H. Aida, "RM/TCP: protocol for reliable multi-path transport over the Internet", in *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA '05)*, vol. 1, 2005, pp. 801–806.

[90] Y. Dong, N. Pissinou, and J. Wang, "Concurrency Handling in TCP", in *Proceedings of the 5th Annual Conference on Communication Networks and Services Research (CNSR '07)*, 2007, pp. 255–262.

[91] V. Sharma, S. Kalyanaraman, K. Kar, K. Ramakrishnan, and V. Subramanian, "MPLOT: A transport protocol exploiting multipath diversity using erasure codes", in *Proceedings of the 27th IEEE Conference on Computer Communications (INFOCOM '08)*, 2008, pp. 121–125.

[92] V. Sharma, K. Kar, K. Ramakrishnan, and S. Kalyanaraman, "A transport protocol to exploit multipath diversity in wireless networks", *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 4, pp. 1024–1039, 2012.

[93] X. Wang, Z. Feng, D. Fan, Y. Xue, and V. Le, "A Segment-Based Adaptive Joint Session Scheduling Mechanism in Heterogeneous Wireless Networks", in *Proceedings of the 70th IEEE Vehicular Technology Conference Fall (VTC '09)*, IEEE, 2009, pp. 1–5.

[94] C.-L. Tsao and R. Sivakumar, "On effectively exploiting multiple wireless interfaces in mobile hosts", in *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT '09)*, ACM, 2009, pp. 337–348.

[95] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multipath congestion control for shared bottleneck", in *Proceedings of Workshop on Protocols for Fast Long-Distance Networks (PFLDNeT 09 ')*, 2009, pp. 19–24.

[96] C. Raiciu, D. Wischik, and M. Handley, "Practical congestion control for multipath transport protocols", *University College of London Technical Report*, 2009.

[97] C. Raiciu, M. Handley, and D. Wischik, "Coupled Multipath-Aware Congestion Control", *IETF Internet-Draft*, 2009, `https://tools.ietf.org/html/draft-raiciu-mptcp-congestion-00`.

[98] A. Ford, C. Raiciu, M. Handley, and S. Barre, "TCP Extensions for Multipath Operation with Multiple Addresses", *IETF Internet-Draft*, 2009, `https://tools.ietf.org/html/draft-ford-mptcp-multiaddressed-00`.

[99] C. Raiciu, C. Pluntke, S. Barré, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath TCP", in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, p. 10.

[100] S. Barré, C. Paasch, and O. Bonaventure, "MultiPath TCP: From Theory to Practice", in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I*, ser. NETWORKING '11, Springer-Verlag, 2011, pp. 444–457.

[101] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic window coupling for multipath congestion control", in *Proceedings of the 19th IEEE International Conference on Network Protocols (ICNP '11)*, IEEE, 2011, pp. 341–352.

[102] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp", in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA: USENIX, 2012, pp. 399–412.

[103] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is Not Pareto-optimal: Performance Issues and a Possible Solution", in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, ACM, 2012, pp. 1–12.

[104] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution", *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1651–1665, 2013.

[105] M. Li, L. Andrey, S. Tarkoma, and A. Ylä-Jääski, "The Delayed ACK evolution in MPTCP", in *Global Communications Conference (GLOBECOM), 2013 IEEE*, IEEE, 2013, pp. 2282–2288.

[106] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath TCP", in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT '13*, New York, New York, USA: ACM Press, 2013, pp. 393–398.

[107] A. Singh, M. Xiang, A. Konsgen, C. Goerg, and Y. Zaki, "Enhancing fairness and congestion control in multipath TCP", in *Proceedings of the 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC '13)*, 2013, pp. 1–8.

[108] T.-A. Le and L. X. Bui, "Forward Delay-based Packet Scheduling Algorithm for Multipath TCP", *ArXiv preprint arXiv:1501.03196*, 2015.

[109] H. A. Kim, B. hwan Oh, and J. Lee, "Improvement of MPTCP Performance in heterogeneous network using packet scheduling mechanism", in *Proceedings of the 18th Asia-Pacific Conference on Communications (APCC '12)*, 2012, pp. 842–847.

[110] M. Li, L. Andrey, and Y. Cui, "Network coding based multipath TCP", in *2012 IEEE Conference on Computer Communications Workshops (INFOCOM workshop)*, 2012, pp. 25–30.

[111] Y. Cui, X. Wang, H. Wang, G. Pan, and Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol", in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, IEEE, 2012, pp. 366–375.

[112] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol", *IEEE/ACM Transactions on Networking (TON)*, 2014.

[113] C. Diop, G. Dugue, C. Chassot, and E. Exposito, "QoS-oriented MPTCP Extensions for Multimedia Multi-homed Systems", in *Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA '12)*, 2012, pp. 1119–1124.

[114] Dizhi Zhou, Wei Song, and Minghui Shi, "Goodput improvement for multipath TCP by congestion window adaptation in multi-radio devices", in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, IEEE, 2013, pp. 508–514.

[115] R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. van Adrichem, and F. Kuipers, "Experiences with MPTCP in an intercontinental OpenFlow network", in *Proceedings of the 29th TERENA Network Conference (TNC '13)*, 2013.

[116] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP Algorithms: Theory and Design", *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 305–316, 2013.

[117] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP: Analysis, Design and Implementation", *CoRR*, vol. abs/1308.3119, 2013.

[118] M. Coudron, S. Secci, G. Pujolle, P. Raad, and P. Gallard, "Cross-layer cooperation to boost multipath TCP performance in cloud networks", in *Proceedings of the 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet '13)*, 2013, pp. 58–66.

[119] F. Yang and P. Amer, "Non-renegable Selective Acknowledgments (NR-SACKs) for MPTCP", in *In proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2013, pp. 1113–1118.

[120] M. Li, L. Andrey, S. Tarkoma, Y. Cui, and A. Ylä-Jääski, "Tolerating path heterogeneity in multipath TCP with bounded receive buffers", *Computer Networks*, vol. 64, pp. 1–14, 2014.

[121] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and K.-W. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices", in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, IEEE, 2014, pp. 1815–1823.

[122] M. Li, L. Andrey, S. Tarkoma, and A. Ylä-Jääski, "MPTCP incast in data center networks", *Communications, China*, vol. 11, no. 4, pp. 25–37, 2014.

[123] F. Yang and P. Amer, "Work in progress: Using one-way communication delay for in-order arrival MPTCP scheduling", in *Proceedings of the 9th International Conference on Communications and Networking in China (CHINACOM)*, 2014, pp. 122–125.

[124] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?", in *Proceedings of IEEE Global Communications Conference (GLOBECOM '14)*, 2014, pp. 4807–4813.

[125] M. Coudron, S. Secci, and G. Pujolle, "Differentiated pacing on multiple paths to improve one-way delay estimations", in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 672–678.

[126] M. Allman, H. Kruse, and S. Ostermann, "An application-level solution to TCPâĂŹs satellite inefficiencies", in *Proceedings of the 1st International Workshop on Satellite-based Information Services (WOSBIS '96)*, 1996.

[127] H. Sivakumar, S. Bailey, and R. L. Grossman, "PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks", in *Proceedings of the ACM/IEEE*

conference on Supercomputing (CDROM '00), IEEE Computer Society, 2000, p. 37.

[128] Jason Lee and Dan Gunter and Brian Tierney and Bill Allcock and Joe Bester and John Bresnahan and Steve Tuecke, "Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks", in *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, 2001.

[129] P. Rodriguez and E. W. Biersack, "Dynamic parallel access to replicated content in the Internet", *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 4, pp. 455–465, 2002.

[130] Y. Hasegawa, I. Yamaguchi, T. Hama, H. Shimonishi, and T. Murase, "Improved data distribution for multipath TCP communication", in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '05)*, 2005.

[131] ——, "Deployable multipath communication scheme with sufficient performance data distribution method", *Computer Communications*, vol. 30, no. 17, pp. 3285–3292, 2007.

[132] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: Video Streaming with WWAN Striping", in *Proceedings of the 14th annual ACM international conference on Multimedia*, ACM, 2006, pp. 327–336.

[133] H. Sakakibara, M. Saito, and H. Tokuda, "Design and implementation of a socket-level bandwidth aggregation mechanism for wireless networks", in *Proceedings of the 2nd annual international workshop on Wireless internet*, ACM, 2006, p. 11.

[134] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath Live Streaming via TCP: Scheme, Performance and Benefits", in *Proceedings of the 3rd ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT '07)*, New York, New York: ACM, 2007, 11:1–11:12.

[135] ——, "Multipath live streaming via TCP: scheme, performance and benefits", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP '09)*, vol. 5, no. 3, p. 25, 2009.

[136] S. Tullimas, T. Nguyen, R. Edgecomb, and S.-c. Cheung, "Multimedia streaming using multiple TCP connections", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 4, no. 2, p. 12, 2008.

[137] A. Baldini, L. De Carli, and F. Risso, "Increasing performances of TCP data transfers through multiple parallel connections", in *Proceedings of IEEE Symposium on Computers and Communications (ISCC)*, 2009, pp. 630–636.

[138] D. Kaspar, K. Evensen, P. Engelstad, A. F. Hansen, P. Halvorsen, and C. Griwodz, "Enhancing video-on-demand playout over multiple heterogeneous access networks", in *Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC '10)*, IEEE, 2010, pp. 1–5.

[139] D. Kaspar, K. Evensen, P. Engelstad, and A. F. Hansen, "Using HTTP pipelining to improve progressive download over multiple heterogeneous interfaces", in *Proceedings of the IEEE International Conference on Communications (ICC '10)*, IEEE, 2010, pp. 1–5.

[140] K. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, and C. Griwodz, "Quality-adaptive scheduling for live streaming over multiple access networks", in *Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video*, ACM, 2010, pp. 21–26.

[141] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, and P. Engelstad, "Improving the Performance of Quality-Adaptive Video Streaming over Multiple Heterogeneous Access Networks", in *Proceedings of the 2nd annual ACM conference on Multimedia systems*, ACM, 2011, pp. 57–68.

[142] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad, "Using bandwidth aggregation to improve the performance of quality-adaptive streaming", *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 312–328, 2012.

[143] E. Miyazaki and M. Oguchi, "Evaluation of Middleware for Bandwidth Aggregation using Multiple Interface in Wireless Communication", *International Journal On Advances in Networks and Services*, vol. 4, no. 3 and 4, pp. 343–352, 2012.

[144] K. Habak, M. Youssef, and K. A. Harras, "DBAS: A Deployable Bandwidth Aggregation System", in *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS '12)*, IEEE, 2012, pp. 1–6.

[145] ——, "An optimal deployable bandwidth aggregation system", *Computer Networks*, vol. 57, no. 15, pp. 3067–3080, 2013.

[146] K. Habak, M. Youssef, and K. Harras, "G-DBAS: A Green and Deployable Bandwidth Aggregation System", in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '12)*, 2012, pp. 3290–3295.

[147] K. Habak, K. A. Harras, and M. Youssef, "OPERETTA: An optimal energy efficient bandwidth aggregation system", in *Proceedings of the 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '12)*, IEEE, 2012, pp. 121–129.

[148] W. Zhang, W. Lei, S. Liu, and G. Li, "A general framework of multipath transport system based on application-level relay", *Computer Communications*, vol. 51, pp. 70–80, 2014.

[149] K. Habak, K. A. Harras, and M. Youssef, "Bandwidth aggregation techniques in heterogeneous multi-homed devices: A survey", *ArXiv preprint arXiv:1309.0542*, 2013.

[150] Y. Wen and V. Chan, "Ultra-reliable Communication over Vulnerable All-Optical Networks via Lightpath Diversity", *IEEE Journal on Selected Areas in Communications, Optical Communications and Networking*, vol. 23, no. 8, pp. 1572–1587, 2005.

[151] X. Chen, A. Jukan, A. Drummond, and N. da Fonseca, "A multipath routing mechanism in optical networks with extremely high bandwidth requests", in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, 2009, pp. 1–6.

[152] N. F. Maxemchuk, "DISPERSITY ROUTING IN STORE-AND-FORWARD NETWORKS", Available as `http://repository.upenn.edu/dissertations/AAI7524101`, PhD thesis, University of Pennsylvania, 1975.

[153] P. Key, L. Massoulié, and D. Towsley, "Combining multipath routing and congestion control for robustness", in *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, IEEE, 2006, pp. 345–350.

[154] S. Shakkottai, E. Altman, and A. Kumar, "The Case for Non-Cooperative Multihoming of Users to Access Points in IEEE 802.11 WLANs", in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, 2006.

[155] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 47–52, 2008.

[156] IEEE, "IEEE 802.ad Link Aggregation Task Force", 2000, `http://www.ieee802.org/3/ad/` [Available Online].

[157] W. Simpson, "The Point-to-Point Protocol (PPP)", *IETF RFC 1661*, 1994.

[158] T. Consortium *et al.*, *Openflow switch specification/version 1.1. 0*, 2011.

[159] E. Blanton and M. Allman, "On making TCP more robust to packet reordering", *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, pp. 20–30, 2002.

[160] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: a reordering-robust TCP with DSACK", in *Proceedings of the 11th IEEE International Conference on Network Protocols*, 2003, pp. 95–106.

[161] C. Perkins, "IP Encapsulation within IP", *IETF RFC 2003*, 1996.

[162] C. Perkins, "IP Mobility Support for IPv4, Revised", *IETF RFC 5944*, 2010.

[163] C. E. Perkins, "Mobile IP", *IEEE Communications Magazine*, vol. 35, no. 5, pp. 84–99, 1997.

[164] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host identity protocol", *IETF RFC 5201*, 2008.

[165] E. Nordmark and M. Bagnulo, "Shim6: Level 3 multihoming shim protocol for IPv6", *IETF RFC 5533*, 2009.

[166] P. Jokela, "Using the encapsulating security payload (ESP) transport format with the host identity protocol (HIP)", *IETF RFC 5202*, 2008.

[167] J. Milbrandt, K. Humm, and M. Menth, "Adaptive bandwidth allocation: impact of routing and load balancing on tunnel capacity requirements", in *Next Generation Internet Design and Engineering (NGI '06)*, IEEE, 2006, 8–pp.

[168] R. Stewart, "Stream control transmission protocol", *IETF RFC 4960*, 2007.

[169] R. Hamilton, J. Iyengar, I. Sweet, and A. Wilk, *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*, 2016.

[170] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, *TCP Fast Open*, 2014.

[171] S. Ha and I. Rhee, "CUBIC : A New TCP-Friendly High-Speed TCP Variant", *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel*, vol. 42, no. 5, pp. 64–74, 2008.

[172] N. Cardwell, Y. Cheng, S. C. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR Congestion-Base Congestion Control", *ACM queue*, vol. 14, no. December, pp. 24–30, 2016.

[173] R. Stewart, *Stream Control Transmission Protocol*, 2007.

[174] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options", Tech. Rep., 1996.

[175] S. Mascolo, L. A. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli, "Performance evaluation of Westwood+ TCP congestion control", *Performance Evaluation*, vol. 55, no. 1, pp. 93–111, 2004.

[176] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A Measurement Study on the Impact of Routing Events on End-to-end Internet Path Performance", in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '06)*, ACM, 2006, pp. 375–386.

[177] A. Pathak, H. Pucha, Y. Zhang, Y. Hu, and Z. Mao, "A Measurement Study of Internet Delay Asymmetry", in *PASSIVE AND ACTIVE NETWORK MEASUREMENT*, ser. Lecture Notes in Computer Science, vol. 4979, Springer, 2008, pp. 182–191.

[178] S. Keshav, "A control-theoretic approach to flow control", *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, pp. 188–201, 1995.

[179] K Rojviboonchai, N Watanabe, and H Aida, "One-way-trip time (OWTT) measurement and retransmission policy for congestion control in M/TCP", in *Proceedings of the Annual Conference of IPSJ*, 2002.

[180] K Ramakrishnan, S. Floyd, D Black, *et al.*, "The addition of explicit congestion notification (ECN) to IP", *IETF RFC 3168*, 2001.

[181] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCPâĂŹs fast recovery algorithm", *IETF RFC 6582*, 2012.

[182] C Raiciu, M Handley, and D Wischik, "Coupled congestion control for multipath transport protocols", *IETF RFC 6356*, 2011.

[183] A Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", 2013.

[184] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP", in *Proceedings of the ACM SIGCOMM conference*, vol. 41, 2011, pp. 266–277.

[185] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage", in *Proceedings of the ACM/IEEE conference on Supercomputing*, IEEE Computer Society, 2004, p. 53.

[186] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems", in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, vol. 8, 2008, pp. 1–14.

[187] A. S.-W. Tam, K. Xi, Y. Xu, and H. J. Chao, "Preventing TCP incast throughput collapse at the initiation, continuation, and termination", in *Proceedings of the 20th IEEE International Workshop on Quality of Service*, IEEE Press, 2012, p. 29.

[188] T. Das and K. M. Sivalingam, "TCP improvements for data center networks", in *Proceedings of the 5th International Conference on Communication Systems and Networks (COMSNETS '13)*, IEEE, 2013, pp. 1–10.

[189] P. D. Amer, C. Chassot, T. J. Connolly, M. Diaz, and P. Conrad, "Partial-order transport service for multimedia and other applications", *IEEE/ACM Transactions on Networking (TON)*, vol. 2, no. 5, pp. 440–456, 1994.

[190] S. C. Nguyen and T. M. T. Nguyen, "Evaluation of multipath TCP load sharing with coupled congestion control option in heterogeneous networks", in *Proceedings of the Global Information Infrastructure Symposium (GIIS '11)*, 2011, pp. 1–5.

[191] S. C. Nguyen, X. Zhang, T. M. T. Nguyen, and G. Pujolle, "Evaluation of throughput optimization and load sharing of multipath tcp in heterogeneous networks", in *Proceedings of the 8th International Conference on Wireless and Optical Communications Networks (WOCN '11)*, IEEE, 2011, pp. 1–5.

[192] Y.-C. Chen, Y. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A Measurement-based Study of Multipath TCP Performance over Wireless Networks", in *Proceedings of the 2013 ACM Conference on Internet Measurement Conference (IMC '13)*, 2013, pp. 455–468.

[193] Z. Shamszaman, S. Ara, and I. Chong, "Feasibility considerations of multipath TCP in dealing with big data application", in *International Conference on Information Networking (ICOIN '13)*, 2013, pp. 708–713.

[194] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers", in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, ACM, 2014, pp. 27–32.

[195] B. Arzani, A Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of Path Characteristics and Scheduling Policies on MPTCP Performance", in *Proceedings of the 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2014, pp. 743–748.

[196] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure, "Are TCP extensions middlebox-proof?", in *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, 2013, pp. 37–42.

[197] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?", in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, ACM, 2011, pp. 181–194.

[198] G. Detal, C. Paasch, and O. Bonaventure, "Multipath in the Middle(Box)", in *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMiddlebox '13)*, 2013, pp. 1–6.

[199] C. Nicutar, C. Paasch, M. Bagnulo, and C. Raiciu, "Evolving the Internet with connection acrobatics", in *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, ACM, 2013, pp. 7–12.

[200] M. Becke, H. Adhari, E. P. Rathgeb, F. Fa, X. Yang, and X. Zhou, "Comparison of Multipath TCP and CMT-SCTP based on Intercontinental Measurements", in *Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM '13)*, IEEE, 2013, pp. 1360–1366.

[201] J. Postel and J. Reynolds, "File transfer protocol", *IETF RFC 959*, 1985.

[202] T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network", in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '02)*, IEEE, 2002, pp. 434–443.

[203] L. Golubchik, J. C. S. Lui, T. F. Tung, A. L. Chow, W.-J. Lee, G. Franceschinis, and C. Anglano, "Multi-path continuous media streaming: What are the benefits?", *Performance Evaluation*, vol. 49, no. 1, pp. 429–449, 2002.

[204] S. Mao, D. Bushmitch, S. Narayanan, and S. S. Panwar, "MRTP: a multiflow real-time transport protocol for ad hoc networks", *IEEE Transactions on Multimedia*, vol. 8, no. 2, pp. 356–369, 2006.

[205] H. Schulzrinne, S. Casner, R Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", *IETF RFC 3550*, 2003.

[206] D. Bushmitch, S. Panwar, and A. Pal, "Thinning, striping and shuffling: traffic shaping and transport techniques for variable bit rate video", in *Proceedings of the 2002 IEEE Global Telecommunications Conference (GLOBECOM '02)*, vol. 2, 2002, pp. 1485–1491.

[207] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext transfer protocol–HTTP/1.1*, 1999.

[208] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross-layer design", *IEEE Wireless Communications*, vol. 12, no. 1, pp. 3–11, 2005.

[209] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?", in *2014 IEEE Global Communications Conference*, IEEE, 2014, pp. 4807–4813.

[210] O. Bonaventure. (2012). Multipath TCP, Tutorial, IEEE CloudNet 2012, [Online]. Available: `http://www-phare.lip6.fr/cloudnet12/Multipath-TCP-tutorial-cloudnet.pptx`.

[211] A Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "RFC6182 Architectural guidelines for Multipath TCP Development", *RFC6182*, 2011.

[212] M. Scharf and A. Ford, *Multipath TCP (MPTCP) Application Interface Considerations*, RFC 6897 (Informational), Internet Engineering Task Force, Mar. 2013.

[213] M. Coudron, S. Secci, G. Pujolle, P. Raad, and P. Gallard, "Cross-layer cooperation to boost multipath TCP performance in cloud networks", in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, IEEE, 2013, pp. 58–66.

[214] R. van der Pol, S. Boele, F. Dijkstra, A. Barczyk, G. van Malenstein, J. H. Chen, and J. Mambretti, "Multipathing with MPTCP and OpenFlow", in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, IEEE, 2012, pp. 1617–1624.

[215] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic Window Coupling for multipath congestion control", in *2011 19th IEEE International Conference on Network Protocols*, IEEE, 2011, pp. 341–352.

[216] S. Ferlin, O. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection", in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, IEEE, 2016, pp. 1–9.

[217] H. Oda, H. Hisamatsu, and H. Noborio, "Design, Implementation and Evaluation of Congestion Control Mechanism for Video Streaming", *International Journal of Computer Networks & Communications*, 2011.

[218] T. A. Le, C. S. Hong, and S. Lee, "MPCubic: An extended cubic TCP for multiple paths over high bandwidth-delay networks", in *ICTC 2011*, IEEE, 2011, pp. 34–39.

[219] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP", *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, 2012.

[220] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks", in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT '13*, New York, New York, USA: ACM Press, 2013, pp. 73–84.

[221] C. Paasch, "Improving Multipath TCP", PhD thesis, 2014.

[222] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of MultiPath TCP performance over wireless networks", *Proceedings of the 2013 conference on Internet measurement conference - IMC '13*, pp. 455–468, 2013.

[223] R. Barik, M. Welzl, S. Ferlin, and O. Alay, "LISA: A linked slow-start algorithm for MPTCP", in *2016 IEEE International Conference on Communications (ICC)*, IEEE, 2016, pp. 1–7. arXiv: `arXiv:1011.1669v3`.

[224] Ming Li, A. Lukyanenko, S. Tarkoma, and A. Yla-Jaaski, "The Delayed ACK evolution in MPTCP", in *2013 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2013, pp. 2282–2288.

[225] F. H. Mirani and N. Boukhatem, "Evaluation of forward prediction scheduling in heterogeneous access networks", in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2012, pp. 1811–1816.

[226] M. Coudron. (). MPTCP simulator source, [Online]. Available: `https://github.com/lip6-mptcp`.

[227] C. Paasch, G. Detal, S. Barré, F. Duchêne, and O. Bonaventure. (2013). The fastest TCP connection with Multipath TCP, [Online]. Available: `http://multipath-tcp.org/pmwiki.php?n=Main.50Gbps`.

[228] (). The Application Layer Traffic Optimization (ALTO), [Online]. Available: `http://datatracker.ietf.org/wg/alto/charter/`.

[229] M. Scharf, G. Wilfong, and L. Zhang, "Sparsifying network topologies for application guidance", in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IEEE, 2015, pp. 234–242.

[230] S. Secci, G. Pujolle, T. M. T. Nguyen, and S. C. Nguyen, "Performance-Cost Trade-Off Strategic Evaluation of Multipath TCP Communications", *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 250–263, 2014.

[231] H. Adhari, S. Werner, T. Dreibholz, and E. P. Rathgeb, "LEDBAT-MP – On the Application of Lower-than-Best-Effort for Concurrent Multipath Transfer", in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, IEEE, 2014, pp. 765–771.

[232] . (2016), [Online]. Available: `https://www.ietfjournal.org/multipath-tcp-deployments/`.

[233] M. Coudron, S. Secci, G. Maier, G. Pujolle, and A. Pattavina, "Boosting Cloud Communications through a Crosslayer Multipath Protocol Architecture", in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, IEEE, 2013, pp. 1–8.

[234] A. Akella, B. Maggs, S. Seshan, and A. Shaikh, "On the Performance Benefits of Multihoming Route Control", *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 91–104, 2008.

[235] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, *The Locator/ID Separation Protocol*, 2013.

[236] R. Perlman, "Rbridges: transparent routing", in *IEEE INFOCOM 2004*, vol. 2, IEEE, 2004, pp. 1211–1218.

[237] A. Atlas, T. Nadeau, and D. Ward, *Problem Statement for the Interface to the Routing System*, 2016.

[238] A. Atlas, J. Halpern, S. Hares, D. Ward, and T. Nadeau, *An Architecture for the Interface to the Routing System This*, 2016.

[239] D. Farinacci, D. Meyer, and J. Snijders, *LISP Canonical Address Format (LCAF)*, 2016.

[240] P. Raad, S. Secci, D. C. Phung, A. Cianfrani, P. Gallard, and G. Pujolle, "Achieving Sub-Second Downtimes in Large-Scale Virtual Machine Migrations with LISP", *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 133–143, 2014.

[241] A. Farrel, J. P. Vasseur, and J Ash, "A Path Computation Element (PCE)-Based Architecture", pp. 1–40, 2006.

[242] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection", in *2011 Proceedings IEEE INFOCOM*, IEEE, 2011, pp. 1629–1637.

[243] L. Dunbar, D. Eastlake, R. Perlman, and I. Gashinsky, "Directory Assistance Problem and High-Level Design Proposal", pp. 1–15, 2013.

[244] T. Li and H. Smit, "IS-IS Extensions for Traffic Engineering Status", pp. 1–17, 2008.

[245] D. Saucez, B. Donnet, and O. Bonaventure, "Implementation and preliminary evaluation of an ISP-driven informed path selection", in *Proceedings of the 2007 ACM CoNEXT conference on - CoNEXT '07*, New York, New York, USA: ACM Press, 2007, p. 1.

[246] D. Farinacci, M. Kowal, and P. Lahiri, *LISP Traffic Engineering Use-Cases*, 2017.

[247] R. Perlman, H. Fangwei, D. Eastlake, K. Krupakaran, and T. Liao, "TRILL Smart Endnodes", pp. 1–16, 2017.

[248] (). A linux kernel TRILL implementation., [Online]. Available: `https://github.com/Gandi/ktrill`.

[249] K. Sriganesh, ""Multipath Transmission Control Protocol Proxy"WIPO Patent No. 2012049631. 20 Apr. 2012.",

[250] G. Detal, C. Paasch, S. van der Linden, P. Mérindol, G. Avoine, and O. Bonaventure, "Revisiting flow-based load balancing: Stateless path selection in data center networks", *Computer Networks*, vol. 57, no. 5, pp. 1204–1216, 2013.

[251] D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, D. Fedyk, M. Ouellete, M. Seaman, and P. Unbehagen, "Shortest path bridging: Efficient control of larger ethernet networks", *IEEE Communications Magazine*, vol. 48, no. 10, pp. 128–135, 2010.

[252] (2013). "NU@GE project ", [Online]. Available: `http://www.nuage-france.fr`.

[253] (2016). LISPmob open source LISP node, [Online]. Available: `{http://www.lispmob.org}`.

[254] L. Iannone, D. Saucez, and O. Bonaventure, "Implementing the Locator/ID Separation Protocol: Design and experience", *Computer Networks*, vol. 55, no. 4, pp. 948–958, 2011.

[255] D. Phung, S. Secci, D. Saucez, and L. Iannone, "The OpenLISP control plane architecture", *IEEE Network*, vol. 28, no. 2, pp. 34–40, 2014.

[256] Farinacci, D. and Meyer, D., *"The LISP Internet Groper"RFC 6835*, 2016.

[257] ——, (2016). The lisp internet groper (lig) open source version, [Online]. Available: `https://github.com/davidmeyer/lig`.

[258] *No Title*.

[259] R. Touihri, P. Raad, N. Turpault, F. Cachereul, and S. Secci, "Unifying LISP and TRILL control-planes for distributed data-center networking", in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, pp. 925–930.

[260] R. V. D. Pol and M. Bredel, "Experiences with MPTCP in an intercontinental multipathed OpenFlow network", *... Education Networking ...*, no. November 2012, 2013.

[261] D. Acemoglu, R. Johari, and A. Ozdaglar, "Partially Optimal Routing", *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 6, pp. 1148–1160, 2007.

[262] B. Hesmans, G. Detal, S. Barré, R. Bauduin, and O. Bonaventure, "SMAPP : Towards Smart Multipath TCP-enabled APPlications", in *CoNEXT '15*, 2015.

[263] B. Hesmans and O. Bonaventure, "An enhanced socket API for Multipath TCP", in *Proceedings of the 2016 workshop on Applied Networking Research Workshop - ANRW 16*, New York, New York, USA: ACM Press, 2016, pp. 1–6.

[264] M. Boucadair, C. Jacquenet, D. Behaghel, stefano.secci@lip6.fr, W. Henderickx, R. Skog, O. Bonaventure, S. Vinapamula, S. Seo, W. Cloetens, U. Meyer, L. M. Contreras, and B. Peirens, "An MPTCP Option for Network-Assisted MPTCP", Internet Engineering Task Force, Internet-Draft draft-boucadair-mptcp-plain-mode-09, Oct. 2016, Work in Progress, 23 pp.

[265] M. Boucadair, C. Jacquenet, and T. Reddy, *DHCP Options for Network-Assisted Multipath TCP*, 2016.

[266] M. Coudron, S. Secci, and G. Pujolle, "Differentiated pacing on multiple paths to improve one-way delay estimations", in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IEEE, 2015, pp. 672–678.

[267] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", in *Conference on Communications architectures, protocols and applications (SIGCOMM)*, 1994.

[268] M. Aylene and C. Weigle, "Investigating the Use of Synchronized Clocks in TCP Congestion Control", PhD thesis, University of North Carolina at Chapel Hill, 2003.

[269] S. Floyd, A. Arcia, D. Ros, and J. Iyengar, *Adding Acknowledgement Congestion Control to TCP Abstract*, 2010.

[270] D. Kaspar, "Multipath Aggregation of Heterogeneous Access Networks", PhD thesis, Faculty of Mathematics and Natural Sciences at the University of Oslo, 2012.

[271] F. Yang and P. Amer, "Using One-way Communication Delay for In-order Arrival MPTCP Scheduling", in *Proceedings of Chinacom*, 2014.

[272] Fan Yang and P. Amer, "Non-renegable Selective Acknowledgments (NR-SACKs) for MPTCP", in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, IEEE, 2013, pp. 1113–1118.

[273] V. Paxson, "On calibrating measurements of packet transit times", in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, 1998, pp. 11–21.

[274] *Global positioning system standard positioning service performance standard 4*, 2008.

[275] *Precision Time Protocol (IEEE 1588)*.

[276] D. Mills, U. Delaware, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", *RFC5905*, pp. 1–110, 2010.

[277] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A measurement study of internet delay asymmetry", in *Lecture Notes in Computer Science*, vol. 4979 LNCS, 2008, pp. 182–191.

[278] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end internet path performance", *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2006.

[279] V Paxson, "Measurements and analysis of end-to-end Internet dynamics", PhD thesis, University of California at Berkeley, 1998.

[280] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance", *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[281] S. Liu, T. BaÅ§ar, and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks", *Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[282] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "Compound TCP : A Scalable and TCP-Friendly Congestion Control for High-speed Networks", *Proceedings 25th Conference on Computer Communications (InfoCom 2006)*, pp. 23–29, 2006.

[283] V Jacobson, R. Bradeb, and D. Borman, "TCP Extensions for High Performance", *RFC1323*, 1992.

[284] R. Scheffenegger, M. Kuehlewind, and B. Trammell, "Additional negotiation in the TCP Timestamp Option field during the TCP handshake", *Draft-scheffenegger-tcpm-timestamp-negotiation-05*, 2013.

[285] J.-H.C.J.-H. Choi and C. Y. C. Yoo, "Analytic end-to-end estimation for the one-way delay and its variation", in *Second Consumer Communications and Networking Conference (CCNC)*, IEEE, 2005.

[286] O. Gurewitz and M. Sidi, "Estimating one-way delays from cyclic-path delay measurements", in *Conference on Computer Communications (INFOCOM)*, IEEE, 2001.

[287] S. Biaz and N. H. Vaidya, "Is the round-trip time correlated with the number of packets in flight?", in *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement - IMC '03*, New York, New York, USA: ACM Press, 2003, p. 273.

[288] S. Rewaskar, J. Kaur, and D. Smith, "Why Don't Delay-based Congestion Estimators Work in the Real-world?", 2006.

[289] J. Martin, A. Nilsson, and Injong Rhee, "Delay-based congestion avoidance for TCP", *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 356–369, 2003.

[290] G McCullagh and D. Leith, "Delay-based congestion control: Sampling and correlation issues revisited", ... *University of Ireland, Maynooth, Tech. Rep*, pp. 1–12, 2008.

[291] Google, "BBR: Congestion-Based Congestion Control",

[292] E. P. Ribeiro and V. C. M. Leung, "Asymmetric path delay optimization in mobile multi-homed SCTP multimedia transport", *Proceedings of the 1st ACM workshop on Wireless multimedia networking and performance modeling - WMuNeP*, 2005.

[293] F. Song, H. Zhang, S. Zhang, F. Ramos, and J. Crowcroft, "An estimator of forward and backward delay for multipath transport", *Technical report, University of Cambridge*, no. 747, 2009.

[294] D. Zhou, H. Li, and J. Li, "Analysis of re-sequencing buffer overflow probability based on stochastic delay characteristics", *IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2013.

[295] J. Nagle, *Congestion Control in IP/TCP Internetworks*, RFC 896, Internet Engineering Task Force, Jan. 1984.

[296] R. Braden, *Requirements for Internet Hosts - Communication Layers*, RFC 1122 (INTERNET STANDARD), Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864, Internet Engineering Task Force, Oct. 1989.

[297] *Ns3 official website.*

[298] *Ns3 simulation source code.*

[299] M. Coudron, N. Ho, D. Duy, and S. Secci, "On buffer and window management for Multipath TCP", in *Network of the Future (NoF)*, 2016.

[300] M. Li, A. Lukyanenko, Z. Ou, A. Yla-Jaaski, S. Tarkoma, M. Coudron, and S. Secci, "Multipath Transmission for the Internet: A Survey", *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2016.

[301] M. Coudron. (2016). mptcpnumerics, [Online]. Available: `https://github.com/lip6-mptcp/mptcpnumerics`.

[302] M. Coudron and S. Secci, "Multipath TCP in ns-3: Implementation Evaluation", *HAL report number <hal-01382907>*, 2016.

[303] Json Development Team. (2016). Introducing json, [Online]. Available: `http://www.json.org`.

[304] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation", *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, 2000.

[305] SymPy Development Team. (2016). Sympy: Python library for symbolic mathematics, [Online]. Available: `http://www.sympy.org`.

[306] S. Mitchell, S. M. Consulting, and I. Dunning. (2011). Pulp: A linear programming toolkit for python, [Online]. Available: `https://github.com/coin-or/pulp`.

[307] T. Schoenemann, "Computing optimal alignments for the ibm-3 translation model", in *Proceedings of the Fourteenth Conference on Computational Natural Language Learning, CoNLL 2010, Uppsala, Sweden, July 15-16, 2010*, 2010, pp. 98–106.

[308] *Transmission Control Protocol RFC 793*, 1981.

[309] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, 1988.

[310] B.-H. Oh and J. Lee, "Constraint-based proactive scheduling for MPTCP in wireless networks", *Computer Networks*, vol. 91, pp. 548–563, 2015.

[311] H. Adhari, T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "Evaluation of Concurrent Multipath Transfer over Dissimilar Paths", in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, IEEE, 2011, pp. 708–714.

[312] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab", *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, p. 3, 2003.

[313] (). Global Environment for Network Innovations., [Online]. Available: `http://www.geni.net/`.

[314] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop", in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, New York, New York, USA: ACM Press, 2010, pp. 1–6.

[315] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation", *Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT '12*, p. 253, 2012.

[316] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct Code Execution : Revisiting Library OS Architecture for Reproducible Network Experiments", *COnference on*

Emerging networking experiments and technologies (CONext), pp. 217–228, 2013.

[317] J. Yan and D. Jin, "A Virtual Time System for Linux-container-based Emulation of Software-defined Networks", in *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation - SIGSIM-PADS '15*, New York, New York, USA: ACM Press, 2015, pp. 235–246.

[318] (). Link modeling using ns 3, [Online]. Available: `https://github.com/mininet/mininet/wiki`.

[319] S. Jansen and A. McGregor, "Simulation with Real World Network Stacks", in *Proceedings of the Winter Simulation Conference, 2005.*, IEEE, pp. 2454–2463.

[320] T. Hajime, R. Nakamura, and Y. Sekiya, "Library Operating System with Mainline Linux Network Stack", in *Netdev0.1*, 2015.

[321] T. Werthmann, M. Kaschub, M. Kühlewind, S. Scholz, and D. Wagner, "VMSimInt: A Network Simulation Tool Supporting Integration of Arbitrary Kernels and Applications", in *Proceedings of the Seventh International Conference on Simulation Tools and Techniques*, ICST, 2014.

[322] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker, "To infinity and beyond", in *Proceedings of the twentieth ACM symposium on Operating systems principles - SOSP '05*, New York, New York, USA: ACM Press, 2005, p. 1.

[323] (). OMNet++, [Online]. Available: `https://omnetpp.org/`.

[324] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator", in *USENIX*.

[325] B. Chihani and D. Collange, "A Multipath TCP model for ns-3 simulator", in *Workshop on ns-3 held in conjunction with SIMUTools*, 2011. arXiv: `1112.1932`.

[326] M. Kheirkhah, I. Wakeman, and G. Parisis, "Multipath-TCP in ns-3", in *Ns3 workshop*, 2014.

[327] B. Hesmans and O. Bonaventure, "Tracing multipath TCP connections", in *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, New York, New York, USA: ACM Press, 2014, pp. 361–362.

[328] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O Bonaventure, M. Handley, U. P. Bucuresti, and U. C. D. Louvain, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", no. 1, 1995.

[329] I. Lopez, M. Aguado, C. Pinedo, and E. Jacob, "SCADA Systems in the Railway Domain: Enhancing Reliability through Redundant MultipathTCP", in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, vol. 2015-Octob, IEEE, 2015, pp. 2305–2310.

[330] Z. Deng, *Non-Renegable Selective Acknowledgements (NR-SACKs) for MPTCP*, 2013.

[331] M. Coudron. (2016). mptcpanalyzer: mptcpanalyzer: a multipath TCP analysis tool, [Online]. Available: `http://dx.doi.org/10.5281/zenodo.55288`.

[332] S. Shalunov and B. Teitelbaum, *One-way Active Measurement Protocol (OWAMP) Requirements*, RFC 3763 (Informational), Internet Engineering Task Force, Apr. 2004.

[333] JTelecommunication Networks Group - Politecnico di Torino. (). Tstat: Tcp statistic and analysis tool, [Online]. Available: `http://tstat.polito.it/`.

[334] Ming Li, A. Lukyanenko, and Yong Cui, "Network coding based multipath TCP", in *2012 Proceedings IEEE INFOCOM Workshops*, IEEE, 2012, pp. 25–30.

[335] D. L. Mills, *NTP Performance Analysis*, 2004.

[336] V. Paxson and L. Berkeley, "End-to-End Internet Packet Dynamics",

[337] Sunggon Kim, Ju Yong Lee, and Dan Keun Sung, "A shifted gamma distribution model for long-range dependent Internet traffic", *IEEE Communications Letters*, vol. 7, no. 3, pp. 124–126, 2003.

[338] M. Coudron and S. Secci, “Per node clocks to simulate time desynchronization in networks”, in *Workshop on ns-3 (WNS3)*, Seattle, 2016, p. 6.

[339] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlwind, *Low Extra Delay Background Transport (LEDBAT)*, 2012.

[340] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann, “Socket intents”, in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT ’13*, New York, New York, USA: ACM Press, 2013, pp. 295–300.

[341] *Transport Services (TAPS).*

[342] T. D. Wallace and A. Shami, “Concurrent Multipath Transfer using SCTP: Modelling and Congestion Window Management”, *IEEE Transactions on Mobile Computing*, vol. 1233, no. c, pp. 1–1, 2014.

[343] D. Sarkar, “A Concurrent Multipath TCP and Its Markov Model”, vol. 00, no. c, pp. 615–620, 2006.

[344] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, “Deconstructing MPTCP Performance”, in *22nd International Conference on Network Protocols (ICNP)*, 2014, pp. 269–274.

[345] Q. Peng, A. Walid, and S. H. Low, “Multipath TCP: Analysis and Design”, pp. 1–14,

[346] M. Mellia, I. Stoica, and H. Zhang, “TCP Model for Short Lived Flows”, vol. 6, no. 2, pp. 85–87, 2002.

UNIVERSITÉ PIERRE ET MARIE CURIE
LABORATOIRE D'INFORMATIQUE DE PARIS 6
4 PLACE JUSSIEU, 75005 PARIS