UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Definition of new WAN paradigms enabled by smart measurements

Francesco Ciaccia

Definition of new WAN paradigms enabled by smart measurements

Francesco Ciaccia

Directors:
Dr. René Serral-Gracià
Dr. Mario Nemirovsky

Computer Architecture Department
Universitat Politecnica de Catalunya

This dissertation is submitted for the degree of
Doctor of Philosophy

October 2020

A Maria e Salvatore, il nucleo della famiglia nucleare.

Acknowledgements

Writing this dissertation was not just the result of effort and research, but a long journey during which I met a lot of incredible people: that is what I value the most of this experience.

This thesis would have not been possible without the support given by my supervisors, Dr. Mario Nemirovsky and Dr. René Serral-Gracià. Thank you for the opportunity you gave me, the unconditional support, the knowledge, the mentoring. My appreciation of you goes beyond the professional and academic side.

A special acknowledgement goes to Dr. Ivan Romero, as one of the main advisors and contributors to this work. Our endless conversations, investigations, and collaborations made this thesis possible. Thank you.

To all the people at Clevernet, for their feedback, cooperation, and friendship. Especially I would like to thank: Dr. Rodolfo Milito for his mentoring; Dr. Oriol Arcas Abella for his precious feedback and help in development; Dr. Diego Montero for his great efforts and methodic contributions; Genís Riera Pérez for his continuous support to my research and close friendship. And to everybody else in the office who made my day-to-day lighter, funnier, better (with the risk of forgetting somebody): Roberto Barreda, Arnau Verdaguer, Lluis Castillo, Fernando Stecconi, Francesco Carrella, David Rodríguez, Judit Gonzáles, Tugberk Arkose. I learnt a lot from each one of you; working at Clevernet has been one of the best experiences of my life.

I want to give a special thank to the people that have been very close to me and supported my work throughout all these years: to Alberto, for being the best friend somebody could ever aspire to have. You supported me mentally and physically in many, many occasions; thank you for always being there. To Raquel, for having been such an important part of my life during this journey. To Josue, for his great advise, friendship, and for sharing the joys and pains of pursuing a Ph.D. To my Bros: Doc, Toyo, Teddy. Our friendship does not know distance limits.

Finally, my biggest thank goes to my parents, Maria and Salvatore. Everything I am, I owe it to you. I love you.

This work was supported by the Industrial PhD grant 2015DI023 of AGAUR and Gencat and the project Efficient Smart Multi Connected Networks co-financed by the Spanish Ministry of Ciencia Innovacioon y Universidades with reference RTC-2017-6655-7, The Spanish Agenda Estatal de Investigacion and the European Regional Development Fund (FEDER).

Abstract

Nowadays massive amounts of data are being moved over the Internet thanks to data-hungry applications, Big Data, and multimedia content. Combined with a reduction in cost and augmented reliability for high-speed broadband access, the whole Internet infrastructure is facing new challenges especially when information crosses long geographical distances. That is the case for Wide Area Networks (WANs), which are typically traversed in enterprises with multi-site deployments. When a connection is established between end-points that are geographically distant with high latency and high bandwidth, data is flowing over a so-called Long Fat Network. Currently, transport protocols in end-points are not able to exploit the resources of such links, notably the most common Transmission Control Protocol (TCP) implementations still stuffer from design flaws that limit their efficiency. More recent developments still suffer from low fairness in resource sharing and lack of global visibility.

We identify SD-WAN as an SDN use-case that can enable new transport protocols adoption, improving traffic behavior over WANs, without the need to modify the end-points. In this Ph.D. thesis, we explore new approaches to network measurements that will enable both end-points and SD-WAN edge routers, to gain visibility over the end-to-end network status. Such additional visibility promotes the development of smarter control mechanisms for network traffic, improving resource utilization.

The preliminary study carried on comprises TCP behavior over WANs and existing methodologies to control its traffic patterns and enforce rate throttling. We also identify a specific use case that poses challenges for WAN scenarios: the Split TCP connections in a Performance Enhancing Proxy (PEP).

New control mechanisms to improve resource utilization and fairness are defined in this project. Specifically, we propose a new approach called Receive Window Modulation (RWM) that allows edge-routers to control the sending rate of a TCP connection by modifying the window advertised by the receiver. We prove that such a controller can improve TCP efficiency and fairness by leveraging local information and additional contextual information obtained from network measurements. It also provides a lossless throttling mechanism, allowing for policy enforcement without hindering TCP throughput. We validate RWM in a real experimental scenario, showing improvements of up to 70% in TCP throughput when coupled with loss-based congestion controls. Bufferbloat is also mitigated, reducing the end-to-end TCP latency measured almost three-fold in some scenarios.

Another contribution of this research project includes a new method to estimate network available bandwidth from TCP passive probings based on the statistical analysis of the Inter-Packet arrival time (SABES). The methodology is based on the packet dispersion model and takes advantage of state-of-the-art machine learning techniques to improve its accuracy, including Deep Neural Networks and Kernel Density Estimation. We validate the model in both simulations and real-world experiments, obtaining a median of the mean absolute error distribution of less than 10% of the network capacity.

We also study network capacity estimation and bottleneck detection with an innovative active probing approach called HIRE. We propose a new packet dispersion model that takes into account the packet pairs delay, allowing for precise end-to-end capacity estimation. HIRE also introduces the concept of Hidden packets Red-shift Effect, which consists of injecting TTL expiring packets in between probing pairs at a specific rate. This technique allows locating the narrow link position along the path and even estimating the capacity of some of the other links located before the narrow link. We validate the model in simulations obtaining an estimation error of less than 6% in all scenarios, even when reducing the probing traffic considerably.

All these contributions constitute the building blocks of a Stateful Edge Router Architecture, SERA. Such architecture is presented in the final part of the dissertation, preparing the ground for future developments.

Table of contents

List of figures

List of tables

Glossary

Acronyms / Abbreviations

ANN	Artificial Neural Network
AQM	Active Queue Management
BDP	Bandwidth-Delay Product
DNN	Deep Neural Network
ECN	Explicit Congestion Notification
HIRE	Hidden Packets Red-shift Effect
IXP	Internet Exchange Point
LFN	Long Fat Network
ML	Machine Learning
MPLS	Multiprotocol Label Switching
MPTCP	Multipath TCP
MTU	Maximum Transmission Unit
NIC	Network Interface Card
OS	Operating System
PEP	Performance Enhancing Proxy

QDISC	Queueing Discipline
RED	Random Early Detection
RWM	Receive Window Modulation
RWND	Receive Window
SABES	Statistical Available Bandwidth Estimation
SD-WAN	Software Defined Wide Area Network
SDN	Software Defined Networking
SERA	Stateful Edge Router Architecture
SLA	Service Level Agreement
TBF	Token Bucket Filter
TC	Traffic Control
TCP	Transmission Control Protocol
ToS	Type of Service
VPN	Virtual Private Network
WAN	Wide Area Network
XDP	eXpress Data-Path

Chapter 1

Introduction

Over the last decade, the interest in Software Defined Networking (SDN) has increased both in academia and in industry. The main idea behind SDN is leveraging software to increase network devices programmability, and ease deployment and management. In the industry, the most relevant use-case for SDN that surged during the years is represented by the Software Defined Wide Area Network, SD-WAN. WANs are computer networks which are geographically extended. In enterprises, WANs usually connect branch offices and head-quarters where centralized services, databases, or private clouds are provided. Branch-to-branch topologies are also common. Over such long distances, Internet performance is usually unstable, given that the traffic could cross multiple administrative domains and exchange points; in remote deployments, last mile connections could also be very poor in terms of reliability, connectivity and throughput.

MPLS dedicated connections are the usual choice to provide bounded performance in terms of latency and bandwidth for critical use-cases, according to specific Service Level Agreements (SLA). However, their cost is usually prohibitive, especially when the bandwidth grows. MPLS connections are often encrypted and allow for the establishment of Virtual Private Networks (VPNs). SD-WAN solutions objective, between others, is to allow companies to migrate from expensive MPLS connections, to the use of legacy Internet, while retaining most of the MPLS features (namely ease of deployment, security and SLA compliance).

1.1 Motivation

While most SD-WAN solutions in the market address the problems of enterprise networks management and deployment over WAN effectively, providing bounded performance guarantees over the legacy Internet still represents a challenge. One of the main reasons is its higher congestion level than the one of a dedicated MPLS link, that, combined with the typical high latency of WANs, translates in poor data transfer throughput, often hindering mission critical applications response time. This is particularly true when using the Transmission Control Protocol (TCP), because of its design principles. While other transport protocol such as QUIC are becoming more popular, TCP still moves most of the bytes transmitted over the Internet [50].

Legacy Internet access links do not provide guarantees in terms of availability either, which could cause complete business disruptions and considerable economic losses. To compensate such scenarios, SD-WAN solutions try to increase availability by leveraging multiple Internet access links at the same time and seamlessly migrating traffic in case of connectivity disruption. More sophisticated systems are also able to detect brownouts, where connectivity is still provided but the Quality of Service in terms of specific metrics is not met; however their visibility over the network status is limited to conventional metrics such as jitter, latency and packet loss.

1.2 Objectives

The objective of this research project is to investigate innovative solutions to address SD-WANs criticalities in terms of performance and network visibility. To achieve it, we identify current flaws in the TCP transport protocols and propose a new scheme to improve its performance behavior that can be deployed in an edge router (which represent the typical topological deployment of an SD-WAN solution). Furthermore, new methodologies to estimate the network congestion level are proposed, introducing state of the art machine learning algorithms in processing well known passive measurements such as inter-packet time to extrapolate the end-to-end available bandwidth. Another technique based on active probing is exposed, establishing a new paradigm in the interpretation of packet-pairs dispersion that allows for precise narrow link capacity estimation and location on path. All this additional visibility

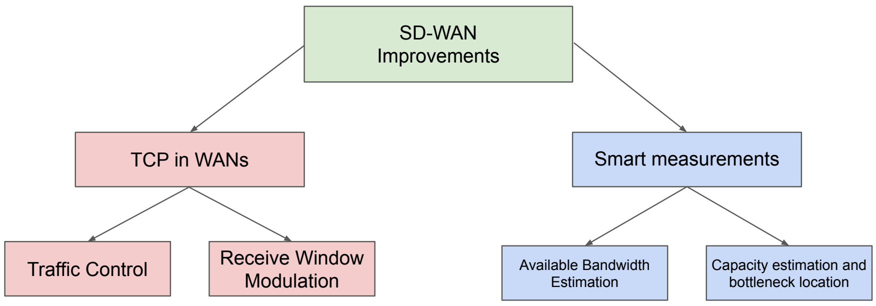

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 1.1 The research project areas covered and contributions.

is finally exploited in combination with the previously described TCP controller to achieve resource sharing and performance optimization over the traffic handled by an SD-WAN device.

1.3 Contributions

Following the motivation and objectives described, the research project focused on different areas in computer networks, with the final intention of designing innovative SD-WAN devices that can leverage additional end-to-end visibility of the network to improve network resources utilization. The project preliminary work included the study of TCP in WAN scenarios and other related topics, and resulted in a granted patent. One of the two core phases of this PhD explored new mechanisms in TCP control that can be implemented in an edge router, resulting in a conference paper. Smart measurements were the focus of the third phase of the project, providing material for two conference papers and a journal. Finally, an architecture for an SD-WAN device based on the rest of the work is proposed as the last part of the project, which was included as part of a second patent filing. Documentation distributed internally at the hosting company was also part of the project output and were used to elaborate this dissertation. Figure 1.1 provides a visual overview of the areas investigated in the project and their inter-dependencies.

1.3.1 Preliminary Work

The preliminary work carried out for the project resulted in two main documents:

[**PAT1**] Nemirovsky, M., Serral-Gracià, R., **Ciaccia, F.**, & Romero, I. (2017). Intelligent adaptive transport layer to enhance performance using multiple channels. U.S. Patent Application No. 15/626,130. Patent Granted. [52]

[**INT1**] Arcas Abella, O., **Ciaccia, F.**, & Montero, D. (2018). TCP Insights Analysis and Operative Recommendations from the Technology Group. Clevernet Internal Documentation.

The topics in analysis include the exploration of standard TCP and Multi-Path TCP (MPTCP) protocols behavior in WAN scenarios, and the study of new WAN measurements to identify path diversity and exploit multi-connectivity. To fully understand the advances in MPTCP development, an extensive study about TCP and its behavior in Linux system was carried on internally at the company and resulted in [**INT1**]. The MPTCP advances resulted in the development of a prototypal MPTCP path-manager for the Linux Kernel. Contributions were provided to the patent filing [**PAT1**], specifically in specifying innovative traceroute techniques. A framework for flexible network assessment through active probing was implemented and integrated in the company product in this first part of the project. An analysis tool leveraging eBPF technology to collect detailed TCP statistics from the Linux kernel was developed and will be open-sourced in the future.

1.3.2 TCP Control

A considerable part of the thesis focuses on TCP; this part of the research project generated the following documents:

[**INT2**] **Ciaccia, F.** (2018). Linux Token Bucket Filter Implementation analysis. Clevernet Internal Documentation.

[**PAP1**] **Ciaccia, F.**, Arcas-Abella, O., Montero, D., Romero, I., Milito, R., Serral-Gracia, R., & Nemirovsky, M. (2019, July). Improving TCP Performance and Reducing Self-Induced Congestion with Receive Window Modulation. In 2019 28th International Conference on Computer Communication and Networks (ICCCN) (pp. 1-6). IEEE. [11]

From the preliminary work studies and needs of the company, surged the idea to research established and new mechanisms to limit TCP self-induced congestion to improve throughput and reduce bufferbloat. An analysis was carried on state of the art mechanisms implemented in the Linux kernel to control flow burstiness and resulted in [**INT2**] . A new approach to mitigate TCP self-induced congestion in WANs was presented in [**PAP1**], representing one of the main contributions of this research project. The Receive Window Modulation controller proposed also represents an effective mechanism to throttle TCP flows without inducing packet loss.

1.3.3 Smart Measurements for Wide Area Networks

The other main area that was covered during the project development was network measurements, resulting in the following publications:

[**PAP2**] **Ciaccia, F.**, Romero, I., Arcas-Abella, O., Montero, D., Serral-Gracià, R., & Nemirovsky, M. (2020). SABES: Statistical Available Bandwidth EStimation from passive TCP measurements. IFIP Networking 2020. [12]

[**PAP3**] **Ciaccia, F.**, Romero, I., Serral-Gracià, R., & Nemirovsky, M. (2020). HIRE: Hidden Inter-packet Red-shift Effect. 2020 IEEE Global Communications Conference. [13]

[**PAP4**] **Ciaccia, F.**, Romero, I., Serral-Gracia, R., & Nemirovsky, M. (2020). AI-enhanced End-to-end Network Assessment. To be submitted to Q1 Journal.

A series of innovative approaches and techniques were presented in [**PAP2**], [**PAP3**], and [**PAP4**], focusing on end-to-end available bandwidth assessment and end-to-end capacity estimation. The information provided by such smart measurements can be exploited effectively to improve the behavior of the controller developed during the first part of the project.

1.3.4 Network Control

The mechanisms described in the smart measurements publications were included in the patent:

[**PAT2**] Romero, I., **Ciaccia, F.**, Nemirovsky, M., & Serral-Gracià, R. (2020). Automatic Communication Network Control. PCT/US Patent Application submitted 05/2020. Patent Pending. [60]

The patent [**PAT2**] also prepared the ground describing coordination mechanisms to achieve a better bandwidth management in a controlled network. An architecture for such an integrated controller is presented in this thesis dissertation and will be the subject of a future publication.

1.4 Methodology

The research methodology employed in the research project encompasses a wide variety of tools. The TCP behavior study and the proposed mitigation scheme have a system design approach, with a qualitative and quantitative evaluation achieved by prototyping the controller and testing it in a real world scenario.

The available bandwidth estimation proposal focuses on a data driven approach, with massive amount of different scenarios generated through simulation, which were then used to train a neural network model whose generalization has been validated both in simulations and real world scenarios.

The narrow link active-probing based tool called HIRE, is based on an analytical model which is validated in simulations.

1.5 Thesis Structure

Table 1.1 shows how the research project contributions map to this dissertation chapters.

The remainder of this manuscript is structured as follows.

Chapter 2 outlines the behavior of modern TCP implementations in WAN scenarios, and exposes the main problem this work addresses.

Chapter 3 shows state-of-the-art solutions in the Linux networking stack that mitigate some TCP flaws, and poses the background to better understand the solutions proposed subsequently.

Chapter	**Title**	**Contributions**
Chapter 2	TCP over Wide Area Network	[**INT1**]
Chapter 3	Traffic Control in Linux	[**INT2**]
Chapter 4	Receive Window Modulation for TCP over WAN	[**PAP1**], [**PAT2**]
Chapter 5	Packet-pair dispersion models and measurement	[**INT2**]
Chapter 6	Available Bandwidth Estimation	[**PAP2**], [**PAT2**]
Chapter 7	Deep Neural Networks for AvBw estimation	[**PAP2**], [**PAT2**], [**PAP4**]
Chapter 8	Narrow link estimation and location	[**PAP3**], [**PAT2**], [**PAP4**]
Chapter 9	Towards a Stateful SD-WAN traffic controller	[**PAT2**]

Table 1.1 Mapping of document contributions to the thesis chapters.

Chapter 4 highlights a proposal to mitigate TCP self-induced congestion and throttle TCP flows losslessly. The system exploits the protocol flow control mechanisms, in a scheme called Receive Window Modulation (RWM).

Chapter 5 discusses network measurements of interest and the challenges posed by their collection in modern virtualized environments, with emphasis on packet dispersion.

Chapter 6 evaluates a proposal to estimate end-to-end available bandwidth from inter-packet time, called SABES. It is based on the packet dispersion model and implemented with a heuristic approach.

Chapter 7 extends the heuristic developed in 6 by means of Deep Neural Networks, improving the available bandwidth estimation obtained.

Chapter 8 delve deeper in the end-to-end capacity estimation problem describing an approach based on active probing and exposing a new model for packet-pairs dispersion analysis, called HIRE.

Chapter 9 summarize the main contributions of the project, and proposes an architecture for a stateful software edge router based on the technology presented in the rest of the dissertation. It also sets the discussion for future developments with some final remarks.

Chapter 2

TCP over Wide Area Networks

The Internet is a packet-switched network of networks that interconnects the world. Instead of a centralized management, it uses relatively simple algorithms in autonomous nodes that have emergent properties as a whole. In contrast to other types of networks, it does not intrinsically provide some guarantees to the users, such as integrity and reliability of the communications. The TCP/IP suite of protocols provides these guarantees, in particular error-free, ordered and reliable delivery of the data. Specifically, TCP is a transport protocol that uses an Automatic Repeat Request mechanisms to retransmit data that is corrupt or lost, and sliding windows to control how much data to send to not saturate the receiver (Flow Control) or the network (Congestion Control). A more comprehensive and detailed overview about generic TCP concepts is provided in Appendix A.

In this chapter we will analyse how classic TCP implementations can cause self-induced congestion in WAN scenarios, also causing a phenomenon called bufferbloat.

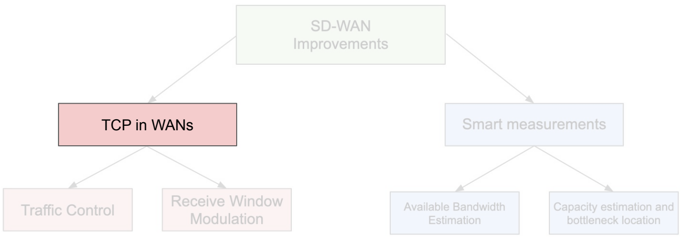

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 2.1 Chapter contribution to the dissertation.

Opposed to loss based TCP congestion control algorithms, we also review more modern approaches providing better network resources estimation. Finally we present a specific scenario of interest called Split TCP, which presents challenges that we address with the proposal presented in Chapter 4.

2.1 TCP Control and Self Induced Congestion

TCP is the transport protocol of choice for many of the distributed application being developed. It provides guarantees in terms of data integrity and delivery; it controls and modulates the sending rate according to estimated network conditions by means of two mechanisms: i) congestion control and ii) flow control. The former is a sender mechanism aimed at adjusting the sending rate, according to congestion events in the network as estimated by the congestion control algorithm. The latter provides the receiver with a mechanism to signal to the sender the amount of data it can receive. In fact, TCP flow control relies on explicitly signaling the available receive window in the protocol header. It was designed considering slow receivers, which were not able to process all the received data because of constrained computational resources. However, TCP flow control is rarely involved in a normal TCP connection in modern Internet era, as processing power has increased dramatically since the protocol definition, preventing the receiver to become the bottleneck. That is generally true, except for cases where the maximum receive window advertised is smaller than the path Bandwidth-Delay Product (**BDP**). This can happen even in modern operating systems that are equipped with receive window scaling options and receive window auto-scaling mechanisms such as Linux, Microsoft Windows, Android, and macOS. In these cases, the maximum buffer allocated by the operating system binds the maximum advertised TCP receive window, causing flows than could generate higher throughput to be flow-controlled. However, this is easily overcome by a correct setup of the maximum TCP allocated buffer, and is configurable in most operating systems. To explain the phenomenology of TCP self-induced congestion we will assume that the advertised TCP receive window is big enough to accommodate the flow BDP, so that we are not limited by TCP flow control.

Opposed to that, while network infrastructure has evolved, TCP design choices in congestion control can still represent a limitation in network resources exploitation.

Many congestion control proposals, e.g. CUBIC [28], act too aggressively, contributing to what we define as *self-induced congestion*: intermediate routers start dropping packets causing consistent throughput reduction, especially in the presence of loss-based congestion control algorithms. Then, when big buffers are present along the path, bufferbloat manifests, reducing the responsiveness of latency-sensitive concurrent flows, such as those of interactive applications. In the following sections we will analyze how loss-based congestion controls operate and affect the network behavior; later we will present a recent development in congestion control that follows a better control scheme based on the estimation of the path BDP, called BBR.

2.1.1 Loss Based Congestion Controls

Historically, TCP congestion control algorithms reacted to packet loss interpreting it as a congestion signal. This caused a reduction in the sending rate to alleviate pressure over the network infrastructure. The most common causes for packet losses on the Internet are:

- **congestion** in intermediate router queues,
- routers **Active Queueing Management** (AQM) policies,
- targeted throughput **throttling**,

among others. In the beginning of TCP history, congestion in intermediate routers could be attributed to slow links, high access concurrency, and small buffers in routers queues. In these cases, packet loss represents an actual signal of network congestion, especially when related to tail-drop in small router queue buffers.

TCP congestion control was designed to react to losses as shown in Figure 2.2. To exemplify the different congestion controls response in this figure, we assume that a loss is always triggering a Retransmission Timeout (RTO), causing the algorithm to restart from the Slow Start phase (SS). The consolidated Reno congestion control reduces its congestion window by half when a packet loss is detected, impacting the flow throughput considerably. Its congestion avoidance phase is linear, causing a slow recovery of the actual optimal throughput. More modern loss-based congestion controls such as CUBIC, have increased their efficiency by

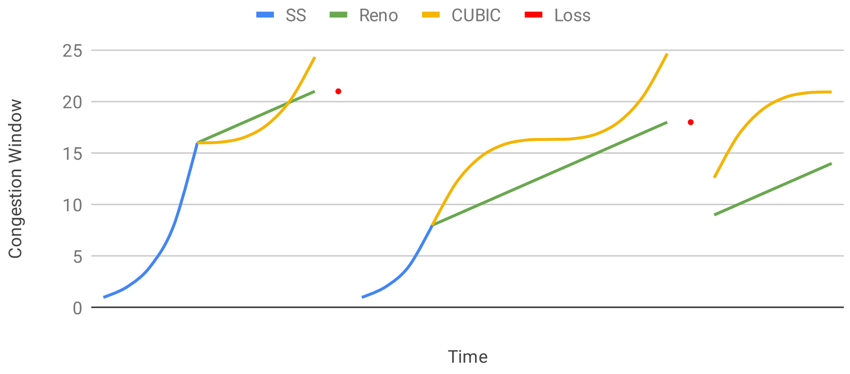


Fig. 2.2 Schematic behavior comparison of Reno and CUBIC loss-based congestion controls.

reducing less the congestion window in case of a loss event and growing faster during their congestion avoidance (following a cubic shape spline function). CUBIC is the default congestion control in the Linux kernel.

Along the years, to cope with higher demand and higher capacity links, and thanks to a reduced price in memory technology, routers manufacturers started increasing the queues buffer size. However, congestion control algorithms did not change, still probing the network for packet loss before reducing their sending rate. In the absence of active queueing management in the routers, packets are tail-dropped once the buffer has been filled. In presence of such bigger buffers, a huge quantity of packets is queued before any actual loss takes place, while the sender is still increasing its sending rate. The excess of packets buffered in the router queue causes other flows crossing the same router, and the congesting flow itself, to perceive a considerable increment in round-trip latency. This phenomenon called **bufferbloat** is disruptive for most interactive and latency-sensible applications, while greatly reducing TCP bulk data throughput because it usually triggers RTOs [25]. Bufferbloat is especially perceived in WAN scenarios characterized by naturally big propagation delays. This is because the TCP acknowledgements take more time to be received by the sender, whose response to congestion is delayed. In Figure 2.3 an example of bufferbloat is shown. A TCP bulk data transfer of a 50MB file using the Reno congestion control is competing for a WAN link of 100Mbps of capacity and a nominal RTT

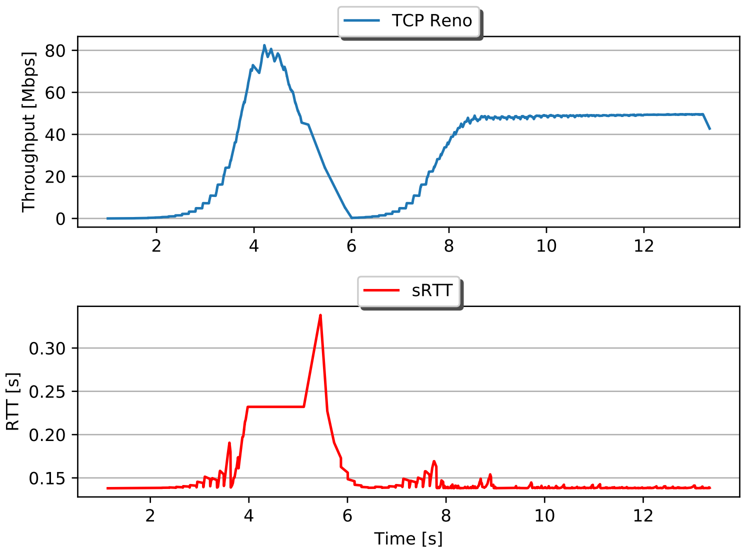


Fig. 2.3 The connection RTT is heavily affected by the Reno congestion control.

of 140ms. Cross-traffic is occupying approximately the 20% of the link capacity. The slow-start growth curve of Reno is aggressive and causes a massive packet loss around second 4. The connection RTT grows up to 350ms, more than doubling the nominal propagation delay. An RTO is triggered because the ACKs notifying the missing packets are delayed considerably. As a consequence the congestion control reduces its congestion window to zero, starting with the slow start process all over again. The network available resources are not efficiently utilized, and the flow takes 13s to complete the file transfer. If the same file would have been transmitted with an average throughput corresponding to the 80Mbps of available bandwidth, it would have finished in approximately 5s, less than half the time.

The key takeaway is that loss-based congestion controls are not able to fully and efficiently exploit WAN network resources as they react to packet loss and not other type of signaling to detect network congestion. Ideally, an efficient congestion control should adapt the sending rate to the path available **BDP**.

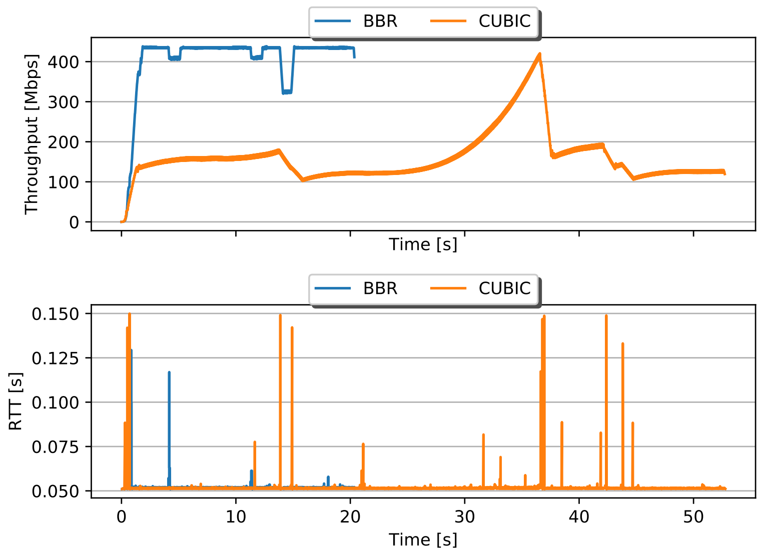


Fig. 2.4 BBR and CUBIC transferring the same file in the same environment.

2.1.2 BBR: Congestion based congestion control

Recently, Google has proposed a new hybrid scheme for congestion control called BBR [9]: its objective is to characterize the current BDP of the connection. It accomplishes so by estimating the minimum RTT and the end-to-end available bandwidth, by periodically inducing queueing and then draining the buffer. BBR operates on the Kleinrock's optimal operating point [44] in which the available bandwidth and the round trip time, RTT, are estimated in order to determine the bandwidth-delay product (BDP). The endpoint probes periodically to estimate the tight link available bandwidth by pacing packets at higher rates than the previous estimation.

Figure 2.4 shows a comparison between CUBIC and BBR in terms if throughput. The same 1GB file is transferred in two separate moments in time, first with CUBIC, then with BBR. The testing environment RTT is approximately 50ms, with an Internet access link nominal capacity of 700Mbps symmetric. BBR terminates the file transfer in less than half the time than CUBIC, with a more consistent behavior. Both congestion control algorithms have impact on the connection RTT. However,

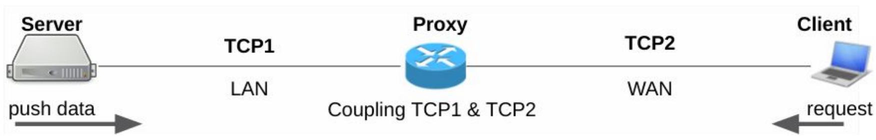


Fig. 2.5 Split TCP connections.

BBR impact is limited in time and corresponds with its bandwidth probing phase, when the queues are filled on purpose. CUBIC on the other hand, is consistently filling the queues with many peaks up to three times the nominal propagation delay especially. While BBR addresses many of the problems of loss-based congestion controls, studies have found a few flaws. BBR has been proven to build long-term standing queues that can cause misleading BDP estimations [30]. This causes the algorithm to often overestimate the BDP while not being fair to other competing flows, especially loss-based congestion control TCP flows [46].

2.2 The Split TCP problem in WANs

A specific use case of interest for Clevernet are Performance Enhancing Proxies (PEPs). PEPs are very useful in mitigating link related degradation as specified in the RFC [5]. Consistent performance improvements over WANs can be achieved by using the Split TCP connections mechanism. In a Split TCP connection scenario, a proxy located close to a receiving or sending end-point, terminates the connection of that end-point and opens another connection towards the final connection destination. This allows the proxy to take control over the TCP connection and apply specific optimization such as Window Scaling (when it is not enabled in the endpoint), improved congestion control algorithms, ACK filtering and retransmission, etc.... In the case of PEP being deployed in an WAN edge router (as an SD-WAN device usually is), a connection impairment happens as shown in Figure 2.5.

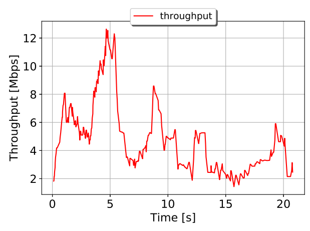
We are particularly interested in the case depicted in Figure 2.5, where there is a WAN between the client and the proxy, while there is a LAN between the proxy and the server. This setup brings up the interactions between the Flow Control and the Congestion Control mechanisms. The Proxy behaves as a slow receiver, struggling to couple two dissimilar TCP connections. While Flow Control dominates the behavior of the (lossless, very low latency) LAN TCP connection, the Congestion Control determines how fast the data is pushed into the WAN.

2.2.1 TCP buffer and windows behavior in Split TCP

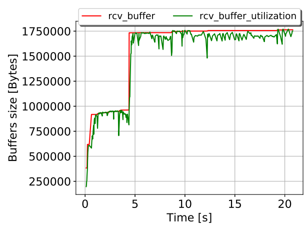
To verify the Split TCP behavior we conducted a test in a controlled environment, replicating the deployment of Figure 2.5. To achieve this, we used two Linux devices acting as server and client of an HTTP file transfer, with a third Linux device acting as a router in between the two. We emulate the desired environment by adding a shaper in the proxy that enforces a rate limiting of 100Mbps with 120ms RTT over the WAN link. The WAN RTT has a variability of $\pm 20ms$ and the link has an additional 0.2% of packet loss, exacerbating the characteristics of a bad WAN link, causing a strong LAN-WAN impairment. The test consists of a download initiated by the client that acts as receiver and provided by the server that acts as a sender. The receive buffers and windows in the Linux systems are tuned so to accommodate the maximum nominal BDP of the connection, so that unnecessary flow control over the WAN will not limit the maximum throughput. The congestion control in use is the Linux default, CUBIC.

We developed a tool that takes advantage of the eBPF technology [23] in the Linux kernel to easily extract information from the operating system, specifically about the network stack status and TCP connections. Figure 2.6 shows the behavior in terms of TCP throughput, receive buffer utilization, TCP windows, and RTT as perceived by the server, the proxy, and the client. The eBPF tool was deployed in all three nodes participating in the connection. We will dig deeper into the Split TCP system by looking at each metric in analysis first in the LAN connection and then in the WAN:

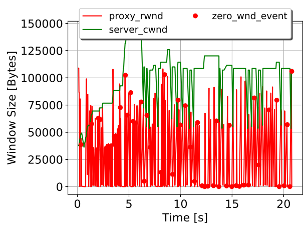
- **Throughput**: from Figures 2.6a and 2.6d, we can see that throughput is loosely coupled between the two connections. This is because the proxy tries to match the amount of data it is receiving from the sender over the LAN with the speed it is able to achieve over the WAN. The bad WAN conditions however, limit the congestion control growth as shown in Figure 2.6f, thus the WAN throughput.



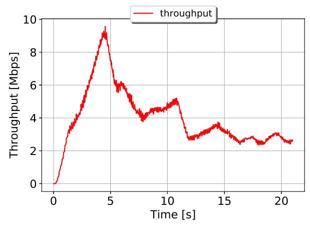
(a) Server throughput - LAN.



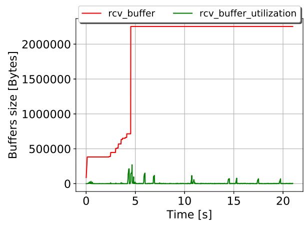
(b) Proxy buffer - LAN.



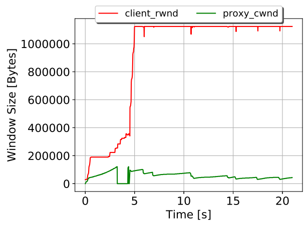
(c) LAN TCP Windows.



(d) Proxy throughput - WAN.



(e) Client buffer - WAN.

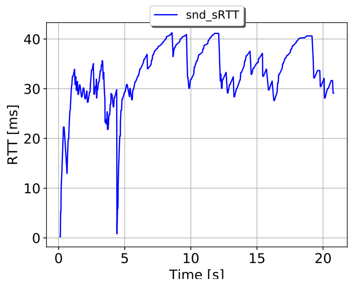


(f) WAN TCP Windows.

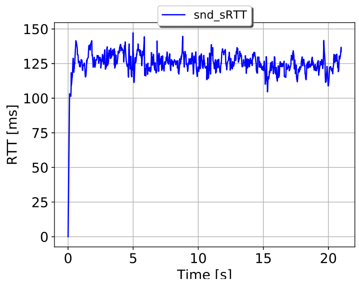
Fig. 2.6 Full Split TCP connections behavior with a proxy close to a sending server.

- **TCP Windows**: as a consequence of the connection impairment, the proxy tries to limit the amount of data the server is delivering over the very low-BDP LAN connection. It does so by activating TCP flow control as shown in Figure 2.6c. The proxy receive window oscillates continuously, advertising a receive window of zero bytes, every time it needs the server to stop sending data. The congestion window of the server is not relevant in the LAN connection as the dominant factor is the proxy flow control (red line) Fig. 2.6c. As already anticipated instead, in the WAN connection between proxy and client, the situation is completely different. The driving factor of the WAN rate is the proxy congestion control (green line), which is always operating far below the client advertised receive window threshold as shown in Figure 2.6f.

- **Buffers**: in the intent of containing the sender rate, the proxy uses flow control at the TCP level. In the meanwhile it is coping with memory pressure on its receive buffer over the LAN, storing as much data as it can while trying to send it to the final client over the WAN slow link as shown in Figure 2.6b: the green line represents the buffer utilization while the red line is the maximum



(a) RTT as seen by server - LAN.



(b) RTT as seen by proxy - WAN.

Fig. 2.7 RTT evolution in the Split TCP scenario.

buffer size, which is auto-tuned by the operating system according to the connection needs. On the other hand, the client receive buffer is barely used as shown in Figure 2.6e, with some occasional spike related to the storage of the out-of-order packets that are stored in case of losses over the WAN, while waiting for retransmission from the proxy.

Finally, we show the RTT recorded for this TCP connection from the server over the LAN (Figure 2.7a) and from the proxy over the WAN (Figure 2.7b). Is notable the behavior shown in the LAN: the back-pressure operated from the proxy over the sender, and the consequent buffering in the proxy, causes a clear case of bufferbloat. The proxy is operating local TCP acknowledgements, so that the sender is not bound to the WAN latency. The propagation delay of this LAN is less than 1ms, but the TCP connection is perceiving up to 40ms RTT between the server and the proxy.

As seen, the split TCP case study presented poses some challenges:

- reducing the consistent memory pressure over the proxy due to the low LAN BDP and slow processing of the proxy;
- mitigating the consequent bufferbloat over the LAN;
- improving the inconsistent performance over the WAN due to lossy links and poor congestion control performance.

2.3 Conclusions

In this chapter we have introduced how modern TCP implementations behave in WANs. We have seen that loss-based congestion controls, which use is still widespread, have some limitations: they can be too aggressive causing bufferbloat and massive tail-drops in routers, which hinder final connection throughput and latency; on the other hand they can be very inefficient in exploiting the available bandwidth in presence of losses, even tough no actual cross-traffic is present (e.g. in case of random losses such as over wireless). While new developments in congestion control seem promising, their short-term wide deployment seems unlikely as they are still affected by a few quirks and require end-points operating systems adoption and migration.

SD-WAN edge devices represent a solution to address most of these problems without the need to modify the end-points, while providing consistent optimizations in traffic engineering and management, for example by using a Performance Enhancing Proxy (PEP). We studied a specific use case for PEPs over WANs, that presented us with different issues. Between them, reducing bufferbloat over low-BDP scenarios in presence of a slow consumer (such as the proxy forwarding over the WAN).

With the objective of addressing such challenges in mind, we investigated already existing mechanisms that could help us, including Active Queueing Management, policing, and throttling techniques as exposed in Chapter 3, before realizing the need to develop a more comprehensive system as exposed in the rest of the dissertation.

Chapter 3

Traffic Control in Linux

The Linux operating system has evolved over the years, at each iteration supporting new features with its networking stack. A Linux-based system is able to cope with almost any networking protocol and task, from switching and routing, up to application level deep packet inspection. It provides a layered system that allows any developer or network administrator to take as much control as needed over any networking-related task. As shown in Figure 3.1, in this chapter we will focus on the traffic control system of the Linux kernel, with special attention on how it interacts with the TCP stack and how it can be used to enforce bandwidth throttling and mitigate bufferbloat. This will provide the needed context to understand the proposal of Chapter 4.

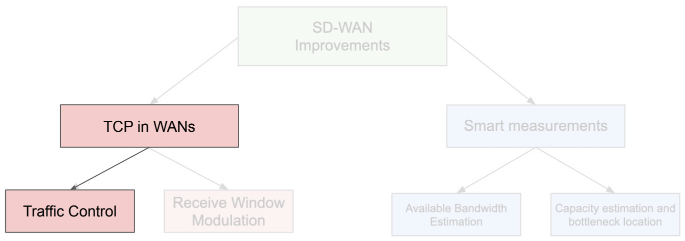

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 3.1 Chapter contribution to the dissertation.

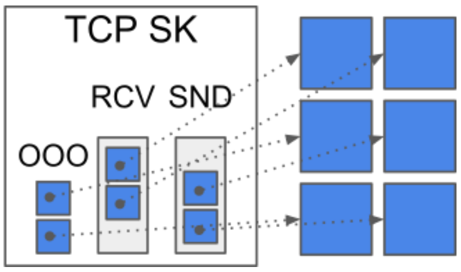


Fig. 3.2 TCP queues in the kernel are collections of socket buffer pointers.

3.1 TCP memory management in the Linux network stack

The memory used by the TCP stack has two important aspects. First, how much of this finite resource is used, and what to do when it becomes scarce. Second, how the memory buffers are related to the protocol's windows, and how their size affect the connection.

For instance, a TCP connection without enough memory will under-perform in a network with big latency. On the other hand, a connection with huge windows may bloat the buffers of the Internet and become too aggressive and unstable, even suffering from unnecessary losses.

The TCP connections have system-wide and per-socket limits. If any of them are reached, or will be soon, the kernel may consider that there is memory pressure and might start performing optimizations, which can consume CPU time and affect the connection throughput. In severe cases, to alleviate memory pressure, packets may be dropped.

In Linux, packets are allocated as Socket Buffers (SKB). Socket Buffers are packets augmented with metadata. SKBs are not moved or copied, except for some cases like de-fragmentation or memory optimization, and pointers to them are pushed and popped from interface or socket queues denoting their ownership. Figure 3.2 shows the main queues of a TCP socket in the Linux kernel: send, receive and out-of-order queues. TCP queues in Linux are colletions of pointers to SKBs. Once allocated, Linux tries to minimize copies and data movements of SKBs, and usually they remain in the same memory location, while only pointers to SKBs are transferred

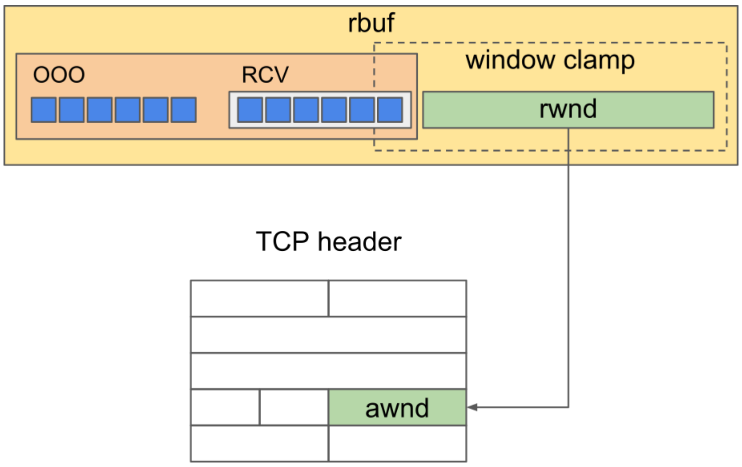


Fig. 3.3 TCP receive buffer in Linux and its relationship with the advertised receive window.

between queues. The total memory used by a TCP connection in Linux can consists of its socket structure and all the SKBs pointed in its queues.

Particularly relevant in the context of this dissertation, is the Linux TCP receive buffer, and how it is implemented and correlated with the TCP receive window advertised in the protocol header. This is because we leverage TCP flow control in a scheme called Receive Window Modulation (RWM) as exposed in Chapter 4. One of the possible RWM implementations uses socket options to modify the TCP receive window advertised on a flow according to the RWM criteria. Another implementation foresees the per-packet modification of the advertised TCP window of a flow, in which case we have to guarantee that the end-point buffers boundaries are respected to avoid breaking TCP semantics and the endpoint memory management. A good understanding of the receive window and buffer mechanisms in Linux is then needed. Figure 3.3 shows a graphical representation of the Linux TCP receive buffer. The TCP receive buffer is shared between the receive and the out-of-order queues of SKBs and the receive window (*rwnd*). The maximum size of the rwnd is denoted by the *window clamp*. The queues may grow and occupy part or all of the rwnd space, which will force the receiver to announce a smaller advertised window (*awnd*) in the packet header (*window update*).

In Linux, and most modern operating systems, a Dynamic Right-Sizing mechanisms is implemented [22]. The idea of dynamic right-sizing is to automatically resize the TCP receive buffer and consequently its advertised receive window, by estimating the connection BDP, without the need to manually tune the TCP buffer size. In this way, the flow control does not represent an unnecessary bottleneck in case of high BDP networks such as Long Fat Networks (LFNs). LFNs are networks characterized by high bandwidth capacity but also very high round trip time, which translates in high Bandwidth Delay Product (BDP). Maximum boundaries set in the kernel default configuration can still represent a limitation in such scenarios.

In the following sections, we will see how the Linux kernel interacts with SKBs to enforce traffic control.

3.2 Traffic Control

The Linux Traffic Control system (TC) consists of a set of tools and queueing systems by which packets are sent and received in a network device [7]. It allows to manipulate packets, control their rate, enforce admission control of specific flows, reorder, and schedule the transmission and reception of packets. The basic control mechanism to achieve such features is the queue. Being packets serialized by design in a stateless packet-switched network such as Internet, queues represent the basic entity to store, delay, and re-organize packets. TC primary operational component are Queueing Disciplines (qdisc).

TC defines six basic actions it can perform over traffic [7]:

- **Shaping**: packets can be delayed in a shaper to control their rate. This is the most common action to be performed to enforce bandwidth throttling in output queues. Shaping can be used to smooth bursty traffic (which is strictly related to the concept of TCP self-induced congestion). The underlying mechanisms of shaping are token and buckets.

- **Scheduling**: schedulers rearrange packets in a queue. The simplest scheduling model is FIFO (First-In First-Out). More sophisticated scheduling discipline try to address specific traffic engineering objectives. For example Stochastic

Fair Queueing (SFQ) tries to prevent a single flow to starve other flows by utilizing all the bandwidth resources.

- **Classifying**: classification is the mechanism by which packets are selected possibly to be treated differently by being queued in different queues. Classes can be defined through filters and are the base for complex shaping hierarchies thanks to the Hierarchical Token Bucket (HTB) qdisc.

- **Policing**: policing consists in measure and limit traffic to a specific queue. Is the basic mechanism to enforce admission control, which means allowing a flow to be established or not (e.g. dropping a new connection establishment because a specific user has reached her quota). Policing foresees packet drop as possible policing action, but other actions are possible (e.g. packet marking). While policing can drop packets, it does not enforce specific bandwidth throttling as in the case of the shaper as it does not delay packets.

- **Dropping**: a mechanism that allow packet drop, according to specific criteria (e.g. dropping can derive from policing a specific class of traffic; packets are also dropped when a shaper queue limit is reached).

- **Marking**: packets can be altered by means of the marking action. For example, it is possible to apply a specific DSCP code in a packet IP header to be used by routers on the path to enforce Differentiated Services (DiffServ).

A qdisc is basically a scheduler. Any TC queue needs an associated qdisc to define how packets are handled. In TC terminology, any qdisc that delays packets such as a shaper, is a non-work-conserving queuing mechanism. The most basic qdisc is FIFO as already described. In Linux the default qdisc for any defined interface is *pfifo_fast*, a slightly modified version of FIFO, that implements basic traffic prioritization following the Type Of Service (ToS) field in the IP header.

Figure 3.4 shows a simplified version of the different levels of the networking stack that a TCP packet goes through in the Linux kernel. Top-down we can see that applications write or read streams of bytes from the TCP socket. Internally the socket manages different queues as list of pointers to SKBs. At the IP level, routing decisions are taken. Before the packets are actually passed or received from the NIC,

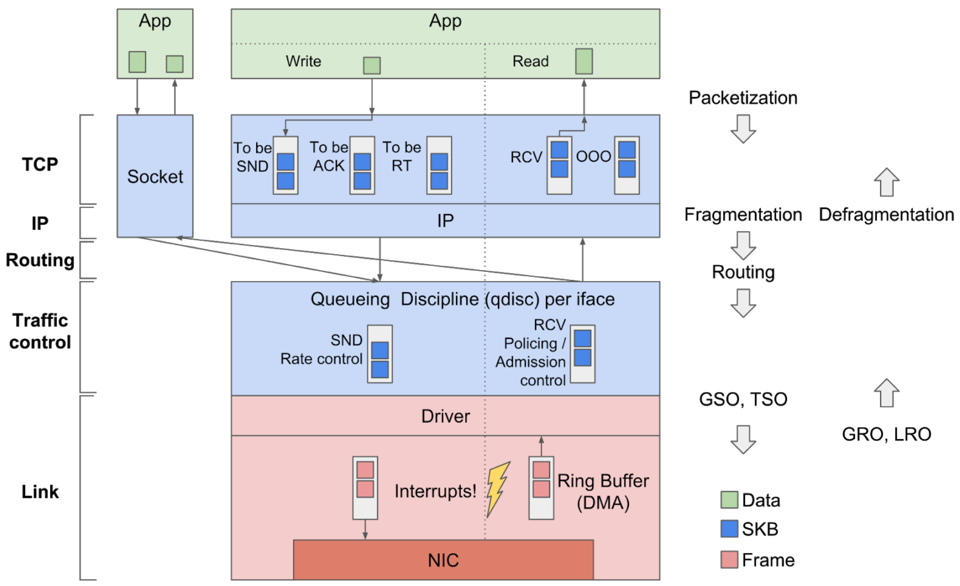


Fig. 3.4 Simplified Linux networking stack up to the transport layer.

packets are queued in the TC qdisc, according to the defined policies. In case of egress traffic, packets can be rate controlled by a shaper, classified, dropped, etc.... Operations on ingress traffic are usually limited to policing/admission control, being that we cannot control how packets are delivered from the network to the device.

At the TC control level, before the packets are actually delivered or received to/from the NIC driver, we find two main mechanisms governing how the application byte stream is split into packets: Generic Segmentation Offload (GSO), Generic Receive Offload (GRO), and TCP Segmentation Offload (TSO). A thorough explanation of these mechanisms and the challenges they pose in per-packet measurements as is exposed in Chapter 5.

In the following sections we will introduce some of the most important Active Queue Management (AQM) schemes aimed at mitigating bufferbloat. Such schemes are implemented as schedulers in the Linux TC system. In obtaining a better control of the traffic to mitigate TCP self-induced congestion, and also to throttle traffic at an arbitrary rate, we studied in detail how the TC shaping mechanism works, focusing on the the token and buckets system it uses, specifically the Token Bucket Filter

(TBF). We will see TBF internals, how it can be used as *pacing* shaper, and which are its limitations.

3.3 Active Queue Management

AQM schemes are used in routers to mitigate network congestion or to improve end-to-end latency. They try to overcome the limitations of a typical tail-drop queuing system, where packet are dropped once buffers are full. Tail-drop can cause TCP global synchronization [64], a phenomena where different TCP loss-based congestion control flows flowing through the same router, loose packets at the same when the router buffer gets full causing a reduction in throughput. This reduction is followed by another exponential increase in throughput from all the senders, causing the buffer to fill again and drop packets causing a bad network resources utilization and increased end-to-end latency.

The most relevant proposal in AQM, as well as the first, is Random Early Detection (RED) [24]. RED is an AQM scheme that drops packets probabilistically before the buffer becomes full by looking at the average queue size. RED assumes a cooperating congestion control at the transport protocol to effectively avoiding dropping all of its packets once the threshold is exceeded. On the other hand it guarantees control on the gateway queue size even in presence of a non-cooperating transport level congestion control. However, dropping packets can be very harmful to loss-based congestion control throughput causing resources under-utilization. Also, RED requires some manual configurations to perform efficiently.

RED can also mark packets once the per-queue quota threshold is exceeded, for example through Explicit Congestion Notification (ECN) [57]. ECN consists of a bit that can be set in the IP header to signal that the router is experiencing congestion. The ECN bit must then be interpreted by the receiving end-point which is then in charge of echoing the ECN bit to the sender in an acknowledgment. ECN biggest limitation is that it needs cooperation from the intermediate and end-points even tough a recent study showed that 70% of the most visited website enabled ECN support by 2017 [50]. However ECN has to be paired with AQM marking mechanisms in the intermediate routers and enabled in the client operating system. ECN is still not enabled by default for outgoing connections in most of the commercial OSes, limiting its effectiveness. Finally, as exposed in [4], ECN "[...] only detects the presence of congestion, not its extent. In the presence of mild congestion, the TCP congestion window is reduced too aggressively, and this unnecessarily reduces the throughput of long flows".

CoDel [53] was designed to counteract bufferbloat effects. It addresses some of RED limitations: the algorithm is *knob-free*, meaning it is self-configuring; average queue size is not considered as an indicator to identify misbehaving queues anymore. CoDel distinguishes between good and bad queues by looking at the amount of time a packet spends in the queue. If the queueing delay increases over a certain threshold the queue is classified as bad, usually implying the formation of a standing queue. This avoids misclassifying bursty queues as bad. CoDel and AQM techniques in general, are advised as best practices for Internet routers in RFC 7567 [3].

Fejes et al. in [21] study of the effect of AQM on recent congestion control schemes. The study focuses especially on DCTCP [4] and BBRv2 [8] (denoted as *scalable* TCP implementations) and their interaction with modern AQM schemes including CSAQM [51] and PI2 [14]. The article shows that coexistence of loss-based and scalable TCP implementation is hard to reach and that modern AQM strategies cannot still provide good results in terms of fairness.

Even tough AQM and ECN are the established proposal for bufferbloat and congestion mitigation in routers, they present limits. Deployment wise, AQM/ECN schemes need support by both the endpoints and the infrastructure. Dropping/marking packets is harmful for loss-based congestion controls and in some scenarios it can cause excessive reduction of the congestion window and bandwidth under-utilization. On the other hand, they are not effective in signaling congestion to scalable TCP implementations such as BBRv2, causing a fairness problem in presence of heterogeneous congestion control scenarios. Most AQM implementations also do not allow to set different classes of traffic and enforce specific throttling policies for each class.

In the following section we dig deeper into the throttling mechanism provided by the Linux Traffic Control system and evaluate its performance and limitations. In Chapter 4 it is exposed a proposal that addresses the limitations of state of the art AQM and throttling mechanisms in a scheme called Receive Window Modulation.

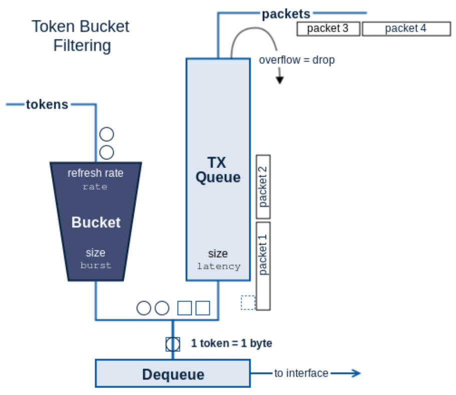


Fig. 3.5 Simplified representation of the Token Bucket Filter. Source: [1].

3.4 Token Bucket Filter

The Token Bucket Filter (TBF) is one of the Linux TC qdisc. TBF is a shaper that enforces a specific rate over traffic being emitted on a network interface (being it virtual or physical). It follows the principles of a token bucket shaper; it also has additional components that allow not only to control the steady state rate but also the peak rate by means of a leaky bucket shaper.

The Linux implementation of all shaping mechanisms relies on tokens and buckets (also for other queuing disciplines). In the Linux kernel implementation, one token corresponds to one byte of data. For TBF the kernel keeps two main structures: a packet buffer that works as bFIFO (bytes FIFO) and a token bucket. Packets are dequeued from the buffer when enough tokens have accumulated in the bucket which is replenished at fixed intervals; this allows to match the desired rate as shown in Figure 3.5.

The token bucket filter can work as policer or a shaper. When working as a policer, packets that arrive to the queue and cannot be dispatched because there are not enough tokens, are dropped. When working as a shaper, packets are accumulated in the buffer waiting for the necessary tokens to be generated.

The main control parameters of this qdisc are:

- **Burst size, B**: the size of the token bucket in bytes.

- **Rate, R**: the rate at which we want to shape the traffic.
- **Limit (Latency)**: limit is the amount of bytes that are allowed to accumulate in the buffer waiting for tokens; it can be expressed also as latency, indicating the time a packet is allowed to wait in the queue for tokens before being discarded. If limit is set to 1 Maximum Transmission Unit (MTU) bytes (or latency to 0 μs), TBF will work as a pure policer, meaning that all non-conforming packets will be dropped.
- **System tick or HZ**: is a hidden parameter that influences the frequency at which the token bucket is replenished of tokens, in systems that cannot take advantage of high resolution timers (i.e. old machines and/or badly compiled kernels). This value is system dependent, e.g. on Intel machines is set to 100Hz, 250Hz, or 1000Hz.
- **Peakrate and second bucket size (mtu)**: these two parameters are needed to achieve perfect shaping (leaky bucket), meaning that no burst is allowed to go over the peakrate value. This is implemented by means of a secondary token bucket with size 1 (mtu).

3.4.1 System tick and high resolution timers

Current versions of the available documentation always refer to the limitation this tool can have due to the system tick parameter. Specifically, the system tick imposes a limitation on the minimum allowed burst/bucket size B to reach a specific rate control; this is because if the shaper is invoked by a fixed system interrupt with a specific frequency f, the maximum rate it can reach is $R_{max} = B \cdot f$. As a consequence, the minimum bucket size necessary to achieve a given rate R is given by: $B_{min} = \frac{R}{f}$.

The bucket size also enforces the maximum burst allowed. In case the shaper has been idle for enough time and the token bucket is full, the next flow of packets will be allowed to be sent at wire speed for a maximum amount of B bytes. The discrete nature of the system tick makes it necessary to set a minimum burst size to enable rate R. It also imposes a limitation to the peakrate control system. This system works with a secondary token bucket, placed right after the first one. This bucket is sized to one MTU worth of tokens. This is needed to obtain precise rate conformance on a millisecond scale, analogous to the metering mechanism of a leaky bucket. For Intel architecture, Linux currently sets the system tick frequency to 250Hz. This does not allow for the leaky bucket to be refilled fast enough, finally limiting the peakrate to no more than 3Mbps with a 1500B MTU.

Even though most of the documentation available on the web regarding this key component behavior (even the very same comments inside the kernel code!) still refer to this limitation, the current implementation relies on a much more modern mechanism that allows TBF to be perfectly usable for today's rate control needs: **high resolution timers** (Linux HR Timers). A thorough investigation of the code brought to a different conclusion, as current implementation relies on high resolution timers allowing for a very high precision also for the peakrate control.

The shaper, once scheduled a packet for dequeue, sets its own watchdog by means of the Linux HR timers which have nanosecond precision. Supposing perfect nanosecond precision, real time process scheduling, and negligible per-packet processing time, a modern CPU with a 4Ghz processor could handle a shaping precision of more than 100Gbps traffic per core (being 5ns the transmission time of a 64B packets at 100Gbps).

Real systems though, are quite different as per-packet processing is not negligible and that in the general case the system in use will not be hard real-time, with many context switches and preemptions on non-atomic operations (see [6]). The picture gets even more complicated when the system is virtualized, as clock precision is generally not good. HR timers also come at expenses of very high CPU consumption according to the number of shapers running on the system, as each one of them will set a nanosecond watchdog timer that will rise an interrupt

3.4.2 TBF evaluation as a pacing shaper

We wanted to evaluate the limitations of the TBF shaper in relation with its bucket size B. This is because, with a bucket size of only one packet the TBF is not only a shaper but also a *pacer*, meaning that it not only enforces bandwidth throttling over a time interval, but it actually schedules each packet departure of the queue at the specified rate. This is of interest to reduce burstiness and self-induced congestion. However,

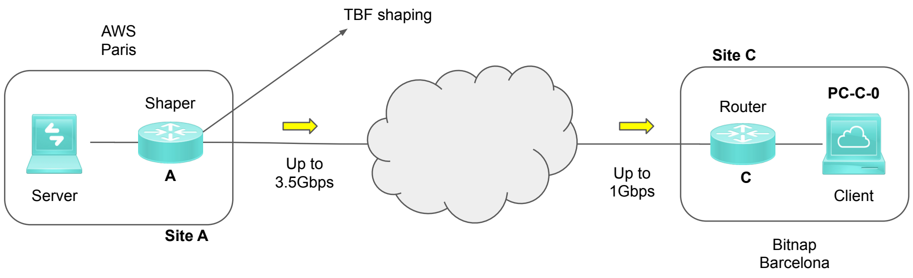


Fig. 3.6 Testing environment for TBF evaluation.

as we found out, a bucket size $B = 1 \cdot MTU$ does not allow to scale efficiently for shaping values higher than 100Mbps, in our virtualized testing environment.

In Figure 3.6 is shown the testbed used to carry on the experiments. The testbed is composed of four VMs, two located in Amazon AWS in the Paris site, and two in Clevernet datacenter in Barcelona. An iperf3 TCP transfer of 10 seconds (fixed time, no fixed size) is executed, being the Barcelona host the client/receiver and the Paris host the server/sender. We enforce TBF shaping on the egress interface of the router/shaper A, while no traffic control is set on any of the other nodes. All TCP related configurations are Linux defaults (default TCP windows and CUBIC congestion control algorithm); they are good enough to guarantee site C ingress link utilization (1Gbps) given the relatively low round trip time between Barcelona and Paris (20ms).

We increased the rate variable R, while keeping the ~1500B (one MTU/packet) value for the burst B to see if it was possible to find the point where, even with high precision timers, it was not possible to achieve the desired maximum throughput. We quickly found that by sizing the burst to one MTU we were already losing precision at 200Mbps. We then increased the burst size to one and a half, two and then three MTUs. Table 3.1 shows the experiments results in terms of actual throughput, given four burst size values with different rates. Results reported are taken from a bandwidth monitoring tool (bmon) measuring throughput on the egress interface of router A. Values are show on standard output and are manually sampled by the operator during the steady state phase of the connection (around 5 seconds from

Rate	**Burst Size [B]**			
	T_{1600B}	T_{2400B}	T_{3200B}	T_{4800B}
100	100	100	100	100
200	192	200	200	200
400	352	400	400	400
600	487	585	592	600
800	606	760	780	800
1000	670	930	960	984

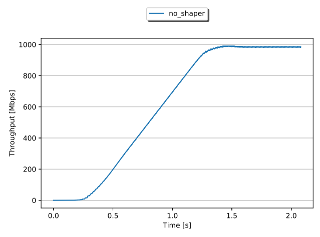
Table 3.1 Throughput achieved by TBF according to the *R* and *B* parameters set.

start). It is possible to observe from table 1 that the burst parameter sized to one MTU is not enough to guarantee sufficient precision for rate control values bigger than 100Mbps. To achieve perfect shaping at all rates we need to increase the burst size to three packets (4800B). When shaping at 1000Mbps we are hitting hardware and/or ISP limitations getting a steady state throughput of 984Mbps for burst size equal to three packets. We tried bigger values than 3MTUs for the burst size *B* without improving the achieved throughput.

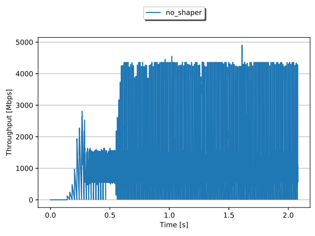
Once we found the minimum token bucket size *B* that allowed for shaping up to 1Gbps of sustained throughput, we wanted to understand if with this configuration we could be able to actually pace packets at that rate on a per-packet level and compare it with a scenario without shaper.

In Figure 3.7 are shown the results for two tests performed in the same environment described previously. We plot just the first two seconds of the trace to focus on the TCP slow start phase, where burstiness can be more evident. To measure the result we run a packet trace capture with tcpdump on the same egress interface where the shaper is set; this implies that we are not measuring actual wire speed but the speed at which packets are passed to the NIC driver ring buffer.

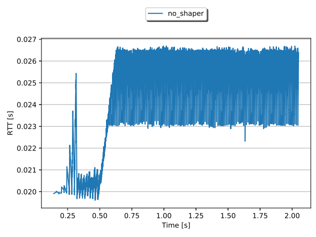
We first run a transfer without any shaping in router A. Figure 3.7a shows the sustained throughput obtained by the file transfer. The sliding window used to compute the throughput is set to $1s$. Already at this scale we can perceive that at the end of the slow start phase there is some burstiness. The phenomena becomes clear when changing scale for the sliding window to $1ms$ as shown in Figure 3.7b.



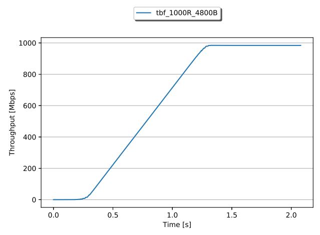
(a) Throughput, 1s, no shaper.



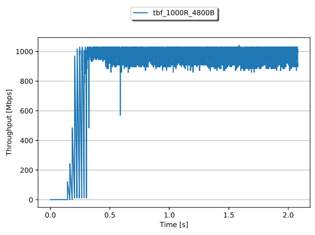
(b) Throughput, 1ms, no shaper.



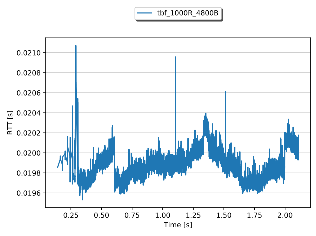
(c) RTT, no shaper.



(d) Throughput, 1s, TBF, 1000R, 4800B.



(e) Throughput, 1ms, TBF, 1000R, 4800B.



(f) RTT, TBF, 1000R, 4800B.

Fig. 3.7 File transfer with and without a TBF shaper.

The sender is actually bursting packets at wire speed (approximately 4Gbps) in some time intervals, followed by intervals without transmission. This causes the building up of a persistent queue in steady state, that impacts the connection perceived RTT by 25% ($+7ms$ compared to the $20ms$ base latency), as shown in Figure 3.7c.

The second experiment places a TBF shaper in router A with an enforced rate R of 1000Mbps (the narrow link capacity of the testbed) and 4800B as burst size B. We can already perceive a more linear growth compared to the non controlled case in the sustained throughput at a $1s$ scale as shown in Figure 3.7d. When looking at a $1ms$ scale we discover that almost perfect shaping is enforced, with no bursty event over the nominal 1Gbps capacity as noticeable in Figure 3.7e. No standing queue builds up as no noticeable additional latency is measured as shown in Figure 3.7f.

3.4.3 TBF limitations

TBF is not building any standing queue in the scenario described in Figure 3.7 because its rate R corresponds exactly to the narrow link capacity of the testbed.

In case we wanted to throttle at values lower than the nominal capacity, the shaper would have had to delay packets in its buffer to match the desired rate, causing the formation of a persistent queue. This happens because being TBF a shaper, it is a non-work-conserving queuing mechanism.

TBF can also be expensive in terms of CPU resources as it acts as the base mechanism for the multi-class token bucket shaper, which in Linux is called Hierarchical Token Bucket (HTB). TBF is agnostic to the type of traffic being throttled, and that is where HTB becomes handy: we can define a hierarchy of traffic classes and their token borrowing policies. Each subclass (or leaf class in HTB terminology) is then throttled by mean of a TBF mechanism. On top of the high amount of interrupt derived from the HR timers of TBF, HTB adds the overhead of its locking mechanism: to guarantee coherency in the shaping enforced, each time a packet traverses the HTB hierarchy it will take the lock over the whole classes tree, causing an higher CPU consumption and making this mechanism less parallelized, as different CPUs that could be handling different flows belonging to different traffic classes, will compete for the same lock. As such, deep HTB hierarchy are usually very performance demanding. Carousel authors present the problem in their paper [61].

3.5 Conclusions

In this chapter we have introduced the Linux networking stack, and dug deep into some of its components.

We have seen how the Linux kernel handles TCP packets in form of Socket Buffers (SKBs) and its TCP memory management with emphasis on the TCP receive buffer. SKBs are the basic unit managed by the Traffic Control system (TC).

We have introduced TC and some scheduler proposals in literature that address TCP bufferbloat and fair sharing. However, these mechanisms are based on the assumptions that TCP reacts to packet losses (which could not be the case in the presence of congestion controls such as BBR). With loss-based congestion control they effectively mitigate bufferbloat, however their way to control traffic is by dropping packets, which is something undesirable as it can hinder throughput unnecessarily. While their configurability degree varies, none of these schedulers can throttle the connection throughput to a desired value, which is one of the objectives of our work.

We then studied in depth the basic shaping mechanism in the Linux kernel, the Token Bucket Filter (TBF) qdisc. We discovered experimentally that TBF takes advantage of high resolution timers, that allow the shaper to also work as a perfect pacer when setting the burst/bucket size to fairly small values, while retaining its ability to scale up to 1Gbps of sustained throughput. Such a solution enables both throttling and burstiness mitigation, at the cost of high resources consumption. Because of its non-work-conserving nature, TBF also cannot prevent traffic from building up persistent queues when we are throttling it at a rate consistently below the sender rate.

This second part of our preliminary study highlighted the limitations of the existing TC mechanisms for TCP as they cannot provide a lossless throttling mechanism that can mitigate bufferbloat at the same time. Existing AQM schemes require cooperation from both the end-points and the infrastructure, while their effectiveness is limited. This motivated us in the development of the Receive Window Modulation scheme exposed in Chapter 4.

Chapter 4

Receive Window Modulation for TCP over WAN

From our study in the interaction between TCP and some of the most relevant traffic control mechanisms provided by the Linux system, we realized that currently there was not a scheme that provided the following features all in a single control mechanism:

- lossless throttling: to guarantee fair usage of a link, current AQM schemes rely on dropping packets to cause a reduction in TCP throughput; a mechanism that allows to signal the sender to reduce its rate without having to drop packets is instead desirable.
- non-delaying shaping: traffic rate control is usually achieved through traffic shapers that delay packets in a queue to control their output rate according to

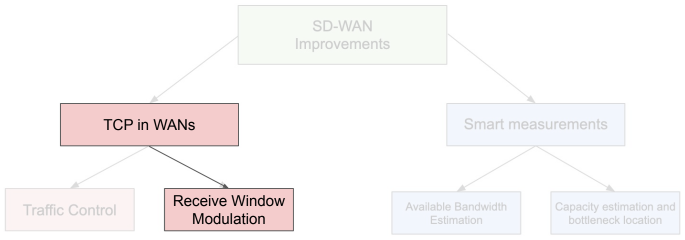

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 4.1 Chapter contribution to the dissertation.

specific policies; this causes the creation of persistent queues that increment end-to-end latency of the connection. We wanted to define a work-conserving shaping mechanism.

- self-induced congestion mitigation: in presence of aggressive senders such as loss-based congestion controls, WANs can get bloated, dropping packets in intermediate routers and reducing the final connection throughput. We wanted to improve TCP congestion control behavior in WANs without the need to change congestion controls in the end-points.

- compatibility with legacy TCP: AQM schemes try to address congestion and bufferbloat mitigation in the network core but often rely on the end-points cooperation (e.g. by supporting ECN); they can also signal congestion to the sender by dropping packets, but such an action can be over-repressive against loss-based congestion controls (e.g. CUBIC, Reno), or completely disregarded by scalable TCP congestion controls (e.g. BBR, DCTCP).

In order to address these problems, we present Receive Window Modulation (**RWM**), a control module for edge routers:

- RWM adjusts the advertised receive window of packets traversing the edge router. This is intended to provide an upper bound to the sender's congestion window growth; the window, which is an upper limit to the connection's bandwidth-delay product (BDP), is based on the router's locally available bandwidth, the estimated connection RTT, and the policies for each class of traffic allowing for the lossless throttling of TCP connections.

- RWM mitigates self-induced congestion and improves end-to-end TCP performance, latency and fairness. Average application goodput is improved up to 70% and latency is reduced by a 2.5x factor in some scenarios.

- This mechanism does not require any modification of the TCP stack, as it is transparent to the end-points, nor connections are terminated by a proxy.

- RWM preserves the characteristics of the sender congestion control deployed in the end-point.

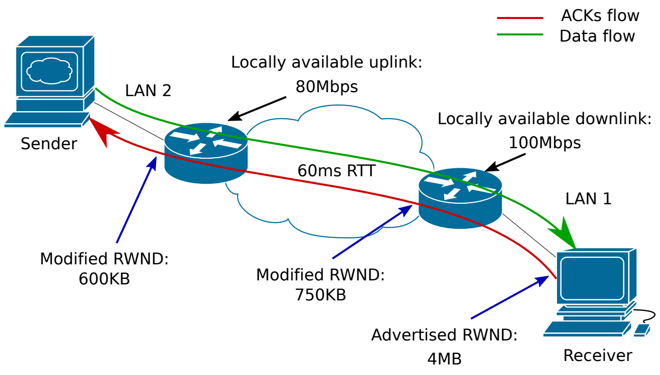


Fig. 4.2 Receive Window Modulation example deploy scheme.

- RWM is compatible with any TCP implementation as it relies on the basic mechanism of TCP flow-control; no cooperation or additional support is required from the end-points.

4.1 Architecture

The main idea behind RWM is to enforce an upper bound to the sending rate of the sender, exploiting TCP flow control. This is achieved by modifying the receiver's advertised window of in-flight acknowledgement packets. This type of control tries to mitigate some limitations of loss-driven congestion control algorithms, which are still extensively used nowadays. Tail drop in intermediate routers can be reduced and bufferbloat avoided if the sending rate in the end-point is adjusted to the path's nominal BDP. The edge deployment vantage point gives the router visibility on all the flows being generated in the Internet access link, which allows a fair share of the edge router resources in terms of bandwidth: the router can throttle flows in case they exceed a specific policy or the local resources of the router without the need to drop packets.

RWM computes the BDP value for each flow based on:

- RTT measurement during connection establishment.

- The router access link available capacity, where capacity can refer to the router network card nominal line rate or a user provided parameter indicating the bandwidth throttling enforced by the ISP on the access link.

Figure 4.2 shows a typical functional schema of a RWM deployment. Both edge routers implement RWM: the controller can be deployed close to both the sender and/or the receiver. In the scenario presented in Figure 4.2 the connection is already established and both edge routers were able to estimate the connection RTT based on the three-way handshake. Each of them transparently modify the receive window advertised by the receiver in the acknowledgement packets, in order to match the locally available bandwidth. The sender will then conform its sending rate to the enforced receive window in case its congestion window exceeds it.

The control logic is triggered on a per-event basis: when a new incoming flow is registered, it is associated a specific traffic profile so that RWM can recompute the optimal value of the receive window for all active flows; a data plane component will then enforce it on the target flow. Particular attention is put in extracting the Window Scaling factor negotiated by both endpoints on connection establishment. Bandwidth allocation is done on a per-class basis, where classes can be defined by the user based on the application port. An upper limit is assigned for the bandwidth of each class. The user is free to allocate bandwidth to the classes within those limits.

Being C the capacity of the edge router link as previously defined; P the allocation policy defined for a specific traffic class j, expressed as percentage of the access link capacity; i the i-th flow for the class j; RWM recomputes the receive window $RWND$ to be assigned to each flow i so that it always respects the following relation:

$$\sum_j \Big(\sum_i \frac{RWND_i}{RTT_i}\Big) * P_j = C \tag{4.1}$$

Equation 4.1 guarantees that the link capacity is distributed equally between flows of the same class and that each class is assigned the user defined share of network resources. The operation of RWM's logic implementing equation 4.1 is described in Algorithm 1. Given the link capacity and the traffic class policies, RWM computes the amount of link share to provide to each class. Then it iterates over all classes and their active flows, and computes the BDP associated to each of them starting from an estimation of the round trip time. It finally computes the receive window to enforce on the acknowledging flow based on the window scaling factor it registered at connection establishment. If the window advertised by the receiver is smaller than the one computed, the adjustment is not applied (see section 4.4.1).

Algorithm 1 RWM operation

```
procedure NEW FLOW REGISTERED
    $C \leftarrow access\_link\_capacity$
    for $c$ in $traffic\_classes$ do
        $P_c \leftarrow class\_policy$
        $C_c \leftarrow C \cdot P_c$
        for $f$ in $flow\_table\_c$ do
            if $WSCALE_f$ not recorded then
                skip to next flow
            end if
            $RTT_f \leftarrow estimated\_flow\_rtt$
            $BDP_f \leftarrow (C_c \cdot RTT_f)/len(flow\_table_c)$
            $RWND_{BDP_f} \leftarrow BDP_f/2^{WSCALE_f}$
            $RWND_f \leftarrow min(RWND_f, RWND_{BDP_f})$
        end for
    end for
end procedure
```

4.1.1 Use Cases

RWM's primary use case targets an enterprise edge gateway or router for WAN links. Our proposed architecture allows for dynamic adaptation of resource utilization, and combined with policy definition and traffic classification, is a powerful tool to be used in corporate networks. RWM is particularly appropriate to improve traffic control at ingress/egress WAN access links, especially when they represent the bottleneck, e.g. in site to site corporate communication. As previously exposed, RWM specifically addresses cases of self-induced congestion; while certain robustness in a real scenario has been proven through experiments (further detailed in section 4.2), its behavior has not been studied in networks where massive amount of cross traffic is the cause of intermediate congested nodes (see section 4.4.3).

4.2 Experiments

A series of experiments were conducted to validate the proposed architecture. The experimental evaluation addresses a scenario where all the traffic flows handled by the router are bulk data transfers belonging to a single traffic class. The objective of the experiments is to validate the benefits obtained when activating the controller in different network scenarios in terms of Flow Completion Time (FCT), goodput, total latency and fairness. In our testbed we emulate different network conditions to generate a range of BDP scenarios. In this evaluation we test the controller between two known locations and deploy one single point of control in the edge router close to the receiver; this edge router has the smallest access link between the two locations. We do not have control over the intermediate hops along the Internet path connecting the two locations. As such the BDP computed in the edge router is only dependent on the local bottleneck and does not take into account possible variability in the inner section of the network. To compensate, we define a $\pm\delta$ around the locally computed BDP value. We've set the value of δ to 10%, in order to study the response of the control when under and over estimating the network conditions.

4.2.1 Testbed

The testbed general configuration is shown in Figure 4.3. The testing environment includes two pairs of nodes in different geographic locations. Each pair is composed of a node that acts as a server or client and another one that acts as router/gateway for the paired instance. The nodes involved are all virtual machines running Debian Linux Stretch. One site has 1Gbps WAN access link, while the other has a 10Gbps one. The typical round-trip time over the Internet between the two sites is 20ms$\pm$0.2ms. In the 1Gbps link site is located the client; it connects to a web server connected through the 10 Gbps link. The virtual machines with the access link of 10Gbps are EC2 Amazon instances. Both instances are m5.xlarge type which are guaranteed to run on Intel Xeon Platinum 8000 series processors, have 4 virtual CPUs assigned and 16GB of RAM. The other two VMs with the 1Gbps access link run in our lab environment; each VM has 4 virtual CPUs and 4GB of RAM assigned. They are

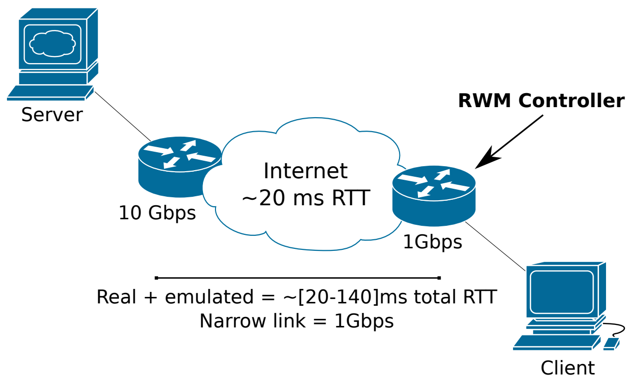


Fig. 4.3 Network topology and configuration for the experiments (RWM only in the client-side router).

hosted on a server blade with 96GB of RAM, running an Intel Xeon CPU E5-2620 v4 with 8 physical cores with hyperthreading.

The only parameters modified at the endpoints are the maximum receive and send buffer size of TCP, which have been tuned to 60MB. This allows the endpoints to reach full theoretical link utilization in all BDP scenarios and avoids the endpoint receive window to become the bottleneck. The congestion control algorithm at the sender is CUBIC (Linux's default). On the router close to the client we use the Linux Traffic Control module called NetEm [29], to change the environment network conditions. We add delay to the link, while keeping the fixed capacity of 1Gbps. Latency introduced has a variability of ±3ms; this variability induces some packet reordering as consecutive packets scheduling can be delayed or anticipated to emulate transient network variability. We do not enforce any artificial loss through NetEm, relying on the intrinsic variability of the Internet between the two sites.

We test with the NetEm standard buffer size of 1000, then with a buffer size of 10000 packets, the former to emulate a small buffer router on path, the latter to study the behavior of the latency induced on the connection by possible bufferbloat. The 10000 packets scenario also allows us to study if there is any smaller queue on the Internet path that causes drops., removing any interference due to the NetEm shaper.

4.2.2 Evaluation

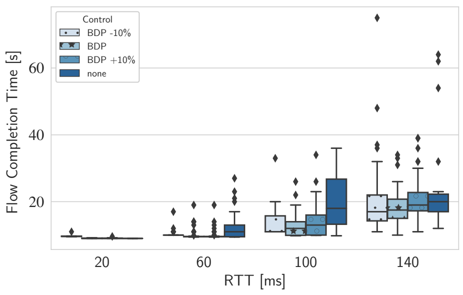
The experiments consist of HTTP GET transfers performed through the client with the `wget` tool. The server provides 1GB files through a `nginx` web server. The client acts as receiver and the server as sender. We combine the following parameters during testing:

- BDP of the path - varied by means of controlling the latency in the client router. Total RTT varies between 20ms and 140ms.
- Client router buffer size: 1000 and 10000 packets.
- Number of concurrent transfers - one and four.
- Controller BDP set: i) exact BDP, ii) exact BDP-10%, iii) exact BDP $+10\%$, and iv) No Controller.

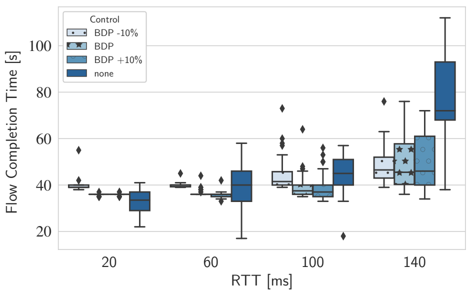
Each combination of parameters is tested 50 times for a total of over 3000 tests. We study the FCT of the downloads. We also recollect statistics from the endpoints by means of an `eBPF` [32] based analysis tool. In particular, in the endpoints we measure:

- The throughput of each download.
- The evolution of the RTT during the data transfer.
- The evolution of the receive window advertised by the receiver and the actual receive window seen by the sender (that could have been modified by our controller).
- The evolution of the congestion window of the sender.

We show the results of FCT using categorical boxplots. Each category represents all the tests performed for a specific value of total RTT for a certain number of concurrent transfers. In each category four boxes are shown: three represent the results when enabling the controller and providing it different values for the BDP estimation as previously described; the fourth box represents the results without the controller.



(a)



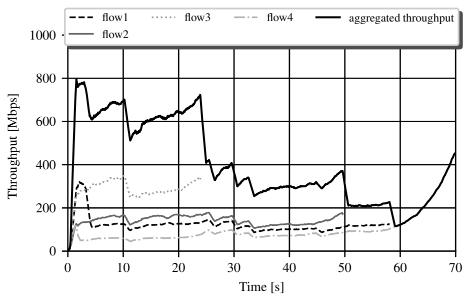
(b)

Fig. 4.4 Flow completion time for one (4.4a) and then four (4.4b) downloads starting simultaneously with or without the controller for different values of RTT.

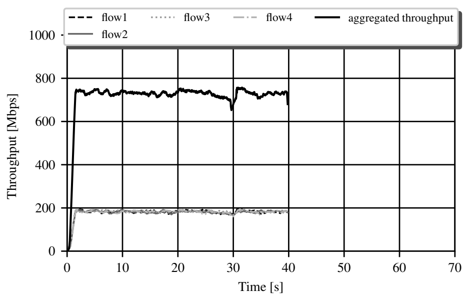
Flow Completion Time

Figure 4.4a depicts the series of tests with one single transfer. We can observe that the range for the non-controlled scenario has considerable variability. The controller improves the behavior of the flows by keeping a more consistent rate during the whole transfer. While improvement is clear for the 60ms and 100ms scenarios, we can see that for 140ms the margin is reduced. Results suggest that, when increasing the RTT, and thus the BDP, the amount of in-flight data injected by the sender is enough to fill one or multiple queues along the path. This holds true even when controlling the receive window to match the nominal BDP. As a matter of fact we start measuring tail drop in our own router's buffer.

On the other hand, we can see from Figure 4.4b that when adding multiple traffic sources the aggressiveness of the congestion control algorithm at the sender is enough to incur in a consistent performance penalty even at lower latencies. At 140ms of RTT is possible to observe that the 50th percentile for the FCT of the uncontrolled scenario is almost 1.5 times higher than the controlled scenario. The difference between the different levels of control applied is marginal with a clear trend: an underestimation of the BDP of 10% brings less variability but average higher values for FCT, while the best results are obtained when controlling at the ideal BDP point; overestimation shows lower 25th percentile values for higher RTT values, but brings more variability to the overall statistic. In this representation each data transfer counts as independent event. We computed the distance between the 50th percentiles of the



(a)



(b)

Fig. 4.5 Throughput of four concurrent transfers. In 4.5a a legacy TCP CUBIC scenario. In 4.5b the same experimental environment is being controlled by RWM.

different categories: RWM improvement in term of FCT in the one transfer scenario is up to 46%, while in the 4 transfers scenario the improvement goes up to 70%. Another relevant observation can be done by looking at the 140ms category in the scenario of Figure 4.4b: the 75th percentile of any of the controlled tests is better than the 25th percentile of the non-controlled case.

Fairness

In figure 4.5 we compare a non controlled experiment using CUBIC in the endpoints, with the same scenario deploying RWM in the receiver's edge router. Four downloads start simultaneously; total RTT is of 60ms. Figure 4.5a shows the throughput of the four connections having a very varied behavior with inconsistent performance. One of the flows is greedier and finishes faster, penalizing the congestion windows of the other three flows. The final FCT for the set is around 70 seconds. In Figure 4.5b the throughput for the test performed using the controller is shown. The throughput is consistent along the whole duration of the data transfer and the congestion windows always works above the level of the receive windows being enforced. The four flows reach fair sharing of the link capacity and all finish at the same time, taking around 40 seconds.

In Figure 4.6 we repropose the results seen in Figure 4.4b but this time considering only the FCT of the slowest of the group of four simultaneously started transfers.

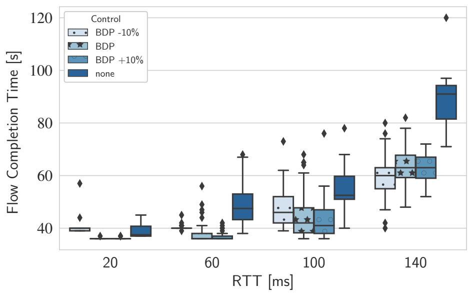


Fig. 4.6 Maximum flow completion time of each set of four concurrent downloads with or without the controller, for different values of RTT.

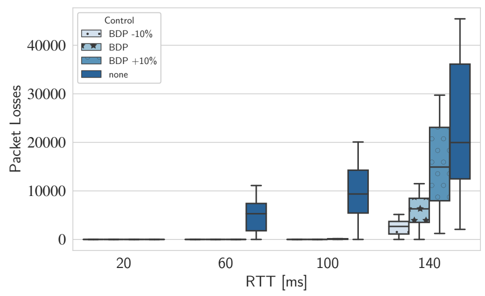
Figure 4.6 confirms the trend seen in the previous section but also provides stronger validation for the fairness properties of RWM.

Latency

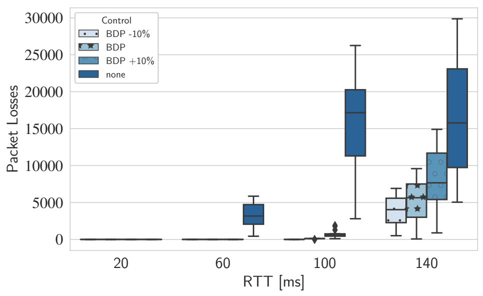
Table 4.1 summarizes the statistics for the RTT of a subset of combinations of the tests performed. In case of lower latencies (20ms case) is possible to observe how the average standing queue induced by a loss-based congestion control is within the order of magnitude of the latency itself. The controller avoids buffering in any of the intermediate nodes. Mean value corresponds to the nominal RTT, and statistical variation is negligible. In all the cases shown, the variability of the non-controlled scenarios exhibits a higher standard deviation.

control	flows	buffer	std	mean	25th	50th	75th	max
none	1	1000	**12.8**	**53.2**	42.8	56.0	64.4	71.2
none	4	1000	**15.9**	**52.8**	41.9	58.5	65.1	71.0
none	1	10000	**12.6**	**55.7**	46.4	59.0	66.1	71.4
none	4	10000	**15.4**	**53.5**	45.5	59.4	65.2	71.1
RWM	1	1000	**0.2**	**21.0**	21.0	21.0	21.0	21.3
RWM	4	1000	**0.2**	**20.4**	20.3	20.4	20.6	25.5
RWM	1	10000	**0.1**	**21.0**	21.0	21.0	21.0	21.1
RWM	4	10000	**0.2**	**20.4**	20.3	20.4	20.5	21.1

Table 4.1 Quartiles and standard deviation for the 20 ms RTT for some combinations of number of concurrent flows and intermediate buffer size (in packets). Statistics for the scenarios controlled at the nominal BDP value are shown.



(a)



(b)

Fig. 4.7 Tail drops caused in the intermediate router queue for one (4.7a) and then four (4.7b) concurrent downloads with or without the controller for different values of RTT.

Self-induced losses

We measured the amount of per-test tail-dropped packets in our intermediate router acting as buffer to increase artificially the latency. This measurement gives an idea of how aggressive are the TCP flows when not controlled or controlled by RWM. Figure 4.7 shows the result when the buffer queue size is set to 1000 packets. Figure 4.7a is a representation of the results obtained by the tests when executing one single TCP transfer at a time. We can see that standard non-controlled CUBIC induces tail-drops in our queue starting at 60ms of induced latency, while the RWM controlled scenario is able to contain the packet loss up to 140ms. In this case, the -10% underestimation of the path BDP reduces considerably the amount of packets dropped. Similar results are obtained with the four concurrent test-set as shown in Figure 4.7b. We start seeing more variation for the RWM controlled scenario already at 100ms, however it is marginal. The absolute number of losses is lower than the single test-set scenario probably due to the augmented concurrency triggering the sender congestion control to reduce the window to a lower threshold. The amount of tail-dropped packets is coherent with the results obtained in terms of FCT as shown in Figure 4.4: the more packet lost, the worse the distribution in terms of throughput, thus longer time to complete the transfer.

4.3 Related Work

In [67] an architecture for bandwidth fair sharing is proposed; it focuses on home gateways and a credit-based resource allocation system. The solution, although, requires some level of interaction with the ISP core network to negotiate the amount of credits the gateway can spend during a congestion period. They also envision a control mechanism based on TCP advertised window modification, but it is not based on BDP estimation. They adapt the TCP receive window by proxying the connection and controlling the receive window at the socket level in the gateway. Instead, RWM proposal for TCP window modification is based on in-flight packet modification.

Other works have been proposed in the past that take advantage of the TCP flow control mechanism to optimize the connection behavior. Explicit Window Adaptation (EWA) [36] also modifies the advertised window of TCP packets as a means of congestion control in intermediate nodes. The goal is to reduce buffer bloat and self-induced congestion due to TCP's window probing. Their testing environment is based on TCP/IP connections over ATM virtual networks. The value of the advertised window is a function of the available buffer space in the ATM router, and the behavior of EWA is compared against a typical Random Early Detection (RED) buffer management mechanism. RWM extends this study in several ways. Window adaptation is a powerful mechanism as demonstrated in EWA with ATM networks, but this technology has been largely superseded by IP-only networks. In this paper, the effects of window adaptation are tested in a more up-to-date environment where RWM is agnostic to the type of network segments the flow will traverse. RWM does not need to be deployed at the bottleneck as its objective is to maximize end-to-end behavior of the TCP flows according to the local resources available to the edge router. The feedback function does not need to be exclusively coupled to the buffer space though, as it can be dynamically derived from the current locally available bandwidth and some user defined traffic policies, computing a BDP value for each connection. In addition, the benefits of window adaptation in this work are not only focused on goodput and buffer utilization, but also a study on the impact on latency is included. In [65] a similar flow control based mechanism is proposed but the scheme is developed with satellite networks in mind and its deployment model foresees the controller to be right before the satellite link bottleneck.

In Chapter 2 we have introduced BBR as a new congestion control algorithm based on BDP estimation. We have also seen its limitations as it lacks of fairness in presence of other flows using loss-based congestion controls and with RTT heterogeneity. RWM operates differently. The receive window is set accordingly to the edge router access link locally available capacity and the measured connection RTT, providing an upper boundary to the sender congestion control. It is not a substitute of the congestion control algorithm deployed in the sender: the response of the sender algorithm to transient network conditions is preserved. Given this, if our system is coupled to senders that use loss-based congestion controls like CUBIC, the fairness of such an algorithm towards other flows is retained. Finally, in contrast to BBR, this system is targeted to routers and does not require modifying the end-points.

4.4 Discussion

4.4.1 RWM Compliance with TCP

TCP is a transport-level protocol, where flow control and congestion control policies are applied by the endpoints. On-route packet processing and modification is not required by design. As a matter of fact, TCP has a checksum mechanism to detect errors that can originate in the network. RWM guarantees TCP data integrity by recomputing the TCP checksum of each modified packet. At the same time the flow control semantic is kept intact: if the original advertised receive window is lower than the value the controller wants to enforce, no changes are applied to the packet. This guarantees that the receiving endpoint can still apply flow control in case of need (e.g if it is not able to process the amount of data received). Another important matter is the preservation of the window scaling factor. The receive window field in the TCP header is a 16-bit word. It allows to signal receive window of up to 65kB, which is very small considered the BDP that modern connections can achieve, especially on LFNs. To overcome this limitation the Window Scaling option has been proposed as part of the standard [34] and it is implemented in most TCP stacks nowadays. Window Scaling is a TCP option which is negotiated at connection establishment and provides a multiplication factor for the receive window value declared in the header allowing for values of up to 2^{30} bytes for the TCP receive window. Being an option negotiated only during the three-way handshake, is necessary to parse this value to correctly interpret the receive window values being advertised on the flow. RWM does not interact with flows for which it was not possible to retrieve the window scaling option on establishment, thus preserving the flow control semantics in all cases.

4.4.2 Path Symmetry

The controller, in order to be able to estimate the connection RTT, to retrieve the window scaling factor and finally to apply the computed receiver window to all packets of the flow, needs to have visibility on packets from both directions of a TCP flow. This is usually the case in edge routers acting as gateways in corporate LANs, which is the typical use case envisioned for RWM as stated in section 4.1.1. On the other hand this limits the applicability of the scheme as routers deeper in the Internet core could not have access to both directions of the flow due to path heterogeneity.

4.4.3 Available Bandwidth Estimation

Currently RWM bases the BDP computation exclusively on its locally available bandwidth and the RTT measured for the connection. In the testing environment presented in section 4.2 the edge router manages the smallest Internet access link between the sender and the receiver. While this schema was shown to be effective in this scenario, there is no guarantee that it would be as effective in presence of consistent cross traffic deep down in the network. In such a case the congestion control of the sender will be triggered by packet losses happening in the network core, not just due to self-induced congestion. Effectiveness of RWM in these scenarios could be improved by developing a BDP estimator that takes into account transient network conditions by employing available bandwidth estimation techniques, predicting variations in network conditions so to control the flow with a more conservative window value before loss events could take place. Chapter 6 addresses the available bandwidth measurement problem and propose an innovative strategy to estimate it, that can be integrated in RWM.

4.4.4 Distributed deployment of points of control

We have developed RWM for the following scenario: traffic across two remote sites traversing the Internet, with RWM deployed in just one of the two sites edge router. This solution does not require the controller deployed in more than one node. Future work includes a study of the convergence properties of multiple sites topologies with an RWM deployed at the edge router of each site.

4.5 Conclusions

In this chapter we presented a novel controller for an edge router that improves end-to-end TCP connections behavior by throttling the flows modifying the TCP receive window advertised: we call it Receive Window Modulation - RWM. RWM computes the window value to enforce by estimating the BDP of the connection based on its locally available bandwidth and the end-to-end RTT. RWM has been proven to enhance TCP goodput, while reducing dramatically the buffering caused in intermediate nodes by TCP loss-based congestion control mechanisms. As a consequence, buffer-bloat is contained and TCP connections behavior in terms of latency improves. The experimental evaluation focuses on a scenario where the traffic is mostly composed of bulk data transfers. In this case RWM shows an improvement for average application goodput of up to 70%, while avoiding buffering in the intermediate nodes and consistently reducing latency in respect to legacy CUBIC TCP connections. The best results have been obtained in presence of multiple concurrent flows where the RWM schema is able to provide high level of fairness when sharing link resources. However, its response to transient network conditions and actual congestion could be improved by additional visibility provided by an available bandwidth estimator. We investigate established models in available bandwidth estimation based on packet-pair dispersion, in Chapter 5. A new algorithm that takes advantage of packet-pair dispersion models applied to passive TCP measurements is then proposed in Chapter 6.

Chapter 5

Packet dispersion model and measurement

In our journey to improve WANs behavior through the definition of new control mechanisms to be deployed in edge devices, we realized the need for a better understanding of the end-to-end network conditions. As exposed in section 4.4.3, our RWM scheme could benefit from an accurate estimation of the network available bandwidth for example. There has been a productive community effort during the last 20 years in defining, developing and testing models that enable end-to-end network bandwidth resources estimation. One of the most relevant is the **packet dispersion model**, which is obtained from measuring the **inter-packet time** of packet pairs or trains.

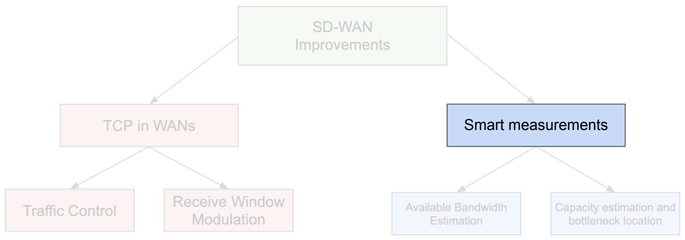

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 5.1 Chapter contribution to the dissertation.

In this Chapter we will provide a definition for the packet dispersion model, show remarkable results obtained by the community in respect to the statistical analysis of packet dispersion distributions, and finally emphasize some challenges involved in measuring inter-packet time effectively in modern NFV environments. Such background is necessary to understand the contributions exposed in Chapters 6 and 8.

5.1 Packet dispersion

The packet dispersion model has been used by the community to study mainly two characteristics of a network: capacity and available bandwidth. The end-to-end capacity of a path is intended as the maximum bandwidth of the link with the least nominal capacity on the path, called the *narrow link*. While capacity is generally a feature which is stable in time (with the exception of wireless scenarios) the available bandwidth of a path is a transient condition that can have continuos variations over small time scale. To measure the available bandwidth of an end-to-end path in a given moment in time we will have to estimate the load of the *tight link*, being this the link with the least amount of available bandwidth, or the most amount of cross-traffic. Even tough the narrow link could easily be the subject of heavy congestion, making it also the tight link, this is not necessarily always the case. For example, an Internet Exchange Point (IXP) with very high nominal capacity, could get very congested making it the tight link of a given end-to-end path.

The main idea behind packet dispersion is to send two packets of equal size back-to-back (i.e. at the maximum rate R achievable by the sender) into the network. Assuming the network in analysis does not have any cross traffic, once the packet pair traverses the narrow link with capacity C_n on path, their relative distance or *dispersion* will increase linearly in respect to C_n. As a consequence, to be able to measure C_n we need $R > C_n$.

Figure 5.2 illustrates packet pair dispersion. Packets of size L are emitted back-to-back by the sender at a rate $R = 1/\Delta_{in}$. When the packet pair traverses a link with

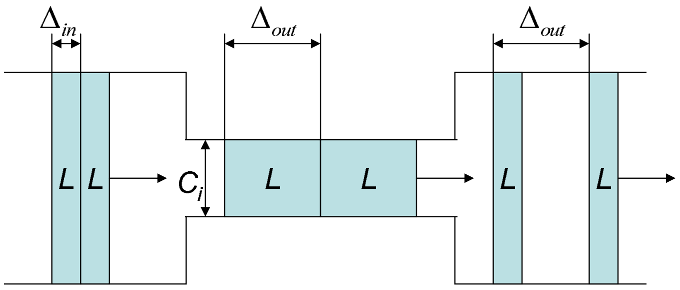


Fig. 5.2 Packet dispersion. Source: [45].

capacity $C_i < R$ their link output dispersion Δ_{out} will be:

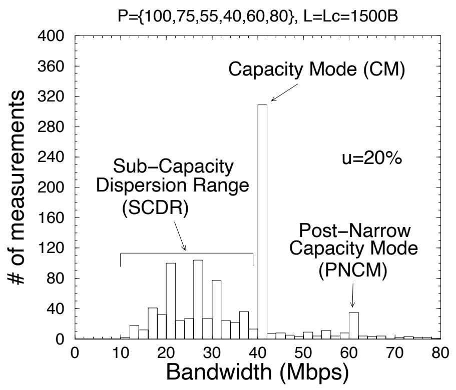
$$\Delta_{out} = \frac{L}{C_i} \tag{5.1}$$

Once the packet pair traverses all the links N along the path, we obtain that the dispersion measured in the receiver is:

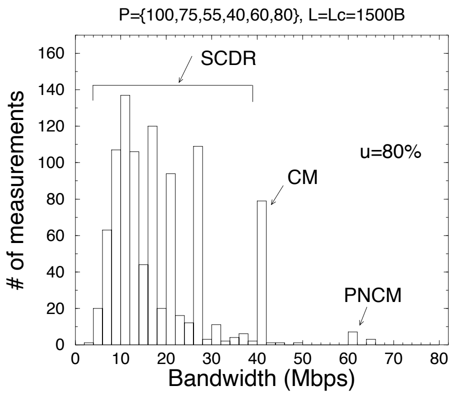
$$\Delta_R = \frac{L}{\min_{i=0\ldots N} C_i} = \frac{L}{C_n} \tag{5.2}$$

Equation 5.2 provides a simple model to estimate the narrow link capacity of an end-to-end path by measuring the inter-packet arrival time of packet pairs in a receiver node. However, it does not take into account the noise introduced by cross-traffic along the path. Congested links can affect the precision of the capacity estimation provided by the packet dispersion model.

On the other hand, we can capture a statistics of multiple packet dispersion measurements to estimate the transient network available bandwidth. Such a statistic can be represented in form of an histogram and relevant information can be extracted from it. A pioneer work in the analysis of the packet pair dispersion distributions is [16]. By using this model they study the packet pair dispersion histogram shapes looking for significant patterns that allow for detection of the end-to-end narrow link capacity or its available bandwidth. Figure 5.3 shows different examples of packet dispersion measurements histograms as analyzed by Dovrolis et al. Given a multi-hop simulated path with a narrow link capacity of 40Mbps, they analyze the distribution under different cross-traffic conditions after sending 1000 packet pair probes of fixed



(a) 20% cross-traffic.



(b) 80% cross-traffic.

Fig. 5.3 Packet dispersion distributions. Source: [16]

size 1500B. In the case of the 20% cross-traffic as in Fig. 5.3a a strong capacity mode is detected. This is because cross-traffic interference is limited and most packet pairs reach the destination with the rate of the narrow link. When cross-traffic increases, for example at 80% as in Fig. 5.3b, the distribution shape changes considerably and the capacity mode is not obviously detectable anymore. Many work in literature focused on extracting valuable information from such distributions, including our own contributions proposed in Chapters 6, 7, and 8.

5.2 Probe-gap curve

In the active probing area, a part from packet dispersion, two well established models for available bandwidth estimation are extensively covered in literature: the *probe-gap model* (PGM) and the *probe-rate model* (PRM). In both cases the analysis carried on is based on the observation of the inter-send gap of some crafted probes from a sender and the inter-arrival gap of those same probes in the receiver. The models are based on the *probe-gap curve* as shown in Figure 5.4. Assuming a fluid cross traffic model and representing the network between sender and receiver as a single FIFO queue, we define g_{in} and g_{out} as, respectively, the inter-packet gap when packets are entering the bottleneck (i.e. leaving the sender), and the inter-packet gap when packets are leaving the bottleneck (i.e. entering the receiver). By looking for the

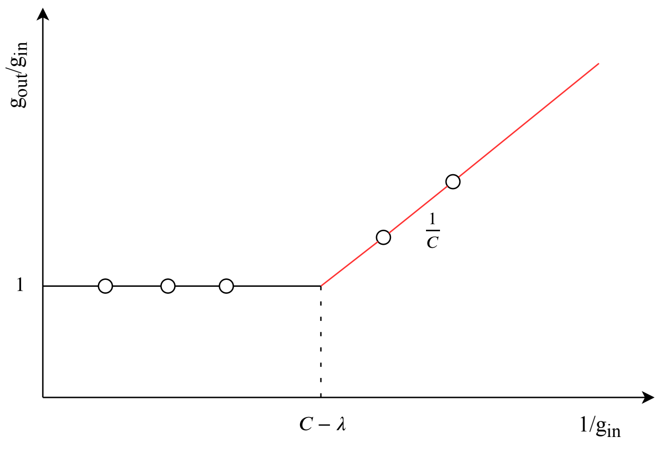


Fig. 5.4 Representation of the probe gap curve. The $C - \lambda$ point marks the available bandwidth.

bending point where $(g_{out}/g_{in}) > 1$, the probe-gap curve provides an estimation of the available bandwidth. PGM and PRM differ in the way they infer the bending point of the curve.

5.2.1 Probe-gap Model - PGM

The idea behind PGM is to send packet pairs at a rate equal to the one of the narrow link to then estimate the rate of the tight link. As such, PGM assumes that capacity is known in advance. A few tools implementing PGM foresee an exploratory phase in which capacity is inferred.

5.2.2 Probe-rate Model - PRM

PRM methods are also called *iterative* methods, as they gradually decrease the time interval between the probes or trains of probes to detect the probe-gap curve bending point, thus they do not take assumptions on the link capacity. While methods like these have been proven to be more accurate than PGM-based methods, they generally require more time and generate more probing traffic. During this period of time, cross traffic can considerably vary, causing incorrect estimations.

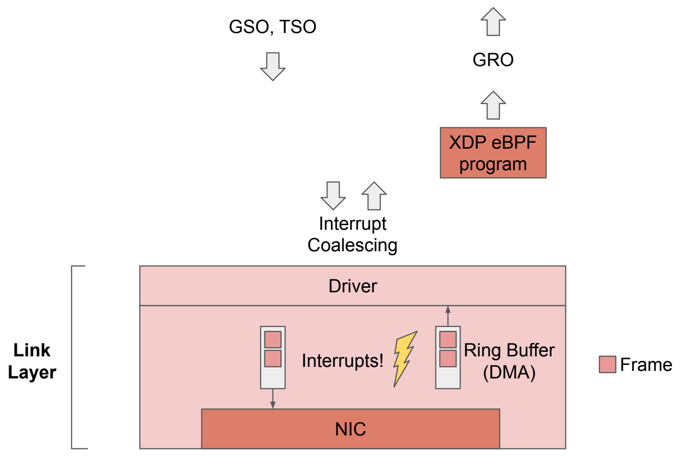


Fig. 5.5 GRO/GSO/TSO mechanisms maximize network throughput.

5.3 Inter-packet time measurement challenges in Linux

Packet dispersion and other available bandwidth estimation models such as PGM and PRM all rely on precise packets timestamping to determine the exact inter-packet time between probes. Obtaining such a measurement in modern operating systems can be quite challenging. Along the years, improving network throughput has been the focus of most end-user operating systems. Favouring throughput is usually achieved by batch-processing packets, minimizing the amount of interrupts handled by the operating system and the operations on protocol headers. The downside of this type of optimizations is the loss of timely information in respect to when packets are actually processed by the NIC and sent/received over/from the network.

In Figure 5.5 we can see that the Linux kernel implements various of these mechanisms. We will focus on GSO, GRO, and TSO. These are optimizations that allow the networking stack to handle SKBs worth of many bytes of data, delaying packetization/segmentation at a later moment (in case of egress traffic - GSO), or grouping adjacent packets in a single SKB (in case of ingress traffic - GRO). This reduces the amount of per-packet operations and improves throughput. For outgoing packets, another optimization is possible, the TCP Segmentation Offload (TSO): in this case the egress traffic segmentation is managed directly by the hardware NIC, when supported.

Another optimization is implemented in the communication between the NIC driver and the rest of the kernel, interrupt coalescing. As the name itself indicates, the idea is to reduce the amount of hardware interrupts generated from the NIC and handled by the OS upon packet reception. To do so, received packets are accumulated in a ring buffer and notified to the kernel only once a certain amount of packets has been received or a timeout is triggered.

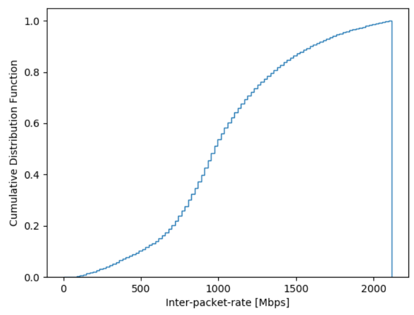
Given all these optimization mechanisms, the point in the stack at which packets are timestamped can affect greatly the accuracy of the inter-packet time measured, thus the inferred inter-packet rate used to build the packet dispersion distributions. For example, timestamping in a user-space application is usually inaccurate. We explored the possibilities provided by Linux to obtain a more accurate inter-packet time.

First we implemented an inter-packet measurement system based on the Linux eXpress Data-Path (XDP) [33]. XDP provides a high performance programmable network data path. It works only on the ingress path of the traffic received by a NIC. It allows to inject eBPF program in the stack hook just after the NIC driver before GRO is performed over incoming packets as shown in Figure 5.5. We set-up an experiment between two back-to-back servers running Linux Debian. The link between servers has a capacity of 1Gbps. The experiment consisted in an HTTP transfer of a 1GB file with no cross-traffic. We timestamped each incoming TCP acknowledgement ts_i in the sender with an XDP eBPF program, and registered each ACK sequence number ack_{num_i}. We then computed what we call *iter-packet rate* following 5.3:

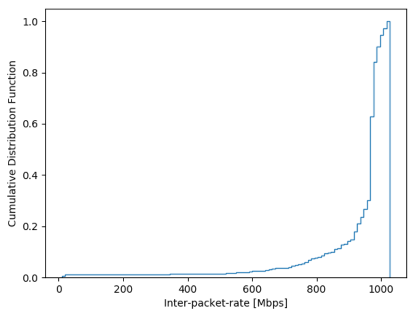
$$IP_{rate} = \frac{ack_{num_i} - ack_{num_{i-1}}}{ts_i - ts_{i-1}} = \frac{\delta_{acks}}{\delta_t} \tag{5.3}$$

Such a measurement allows to infer the rate of the packets originally sent by sender by looking exclusively at the connection acknowledgements. We will use such a metric extensively in Chapters 6 and 7.

In Figure 5.6a we can see the cumulative density function of the inter-packet rate measurements obtained when running the file transfer experiment and timestamping with XDP. As shown, more than 50% of the measurements is resulting in inter-packet rate of more than 1Gbps, which is not correct. This happens as a consequence of the interrupt coalescence performed by the NIC driver [56] [69]. ACKs arrivals are



(a) Cumulative density function of inter-packet rate measured by XDP.



(b) Cumulative density function of inter-packet rate measured by Libpcap with hardware timestamps.

Fig. 5.6 Inter-packet rate measurements in NFV for a 1Gbps link.

aggregated and the timestamp performed in the XDP program does not reflect the actual inter-packet arrival time as seen in the NIC.

We then repeated the experiment using tcpdump to capture packet traces in the receiver NIC. We enabled the hardware timestamp that was supported by the server hardware. In this way, libpcap has access to the timestamp taken by the hardware NIC instead of relying on a software timestamp. To the authors knowledge, XDP eBPF programs cannot get access to the hardware timestamp field which is filled in when instantiating the received packet socket buffer in higher levels of the Linux kernel network stack. As such, in presence of hardware timestamp support, libpcap is able to obtain very accurate inter-packet time measurements. Figure 5.6b shows the cumulative density function of the inter-packet rate obtained by libcap: now most of the measurements are in the 1Gbps range, showing that it is possible to achieve accurate inter-packet rate measurements.

5.4 Conclusions

In this chapter we exposed the main concepts behind the packet dispersion model. Packet dispersion is a method to analyze end-to-end network capacity and available bandwidth. Other models such as PGM and PRM have been introduced as their use is well established in the active probing available bandwidth estimation community. All these models rely on the inter-packet time measurement. We explored how to obtain this measurement in modern OSes such as Linux. We found out that it is possible to measure inter-packet time with good accuracy thanks to improved hardware/software support. This enables the research of new techniques based on the packet dispersion model for SD-WAN applications. In Chapter 6 a first proposal for available bandwidth estimation is presented.

Chapter 6

Available Bandwidth Estimation

Nowadays, improving the Quality of Experience of final users is of paramount importance. To this end, Software-Defined Networking (SDN) opens new horizons for network devices to be smart, and to take advantage of the network deep contextual knowledge. Network stacks in endpoints and edge routers play an important role in managing the available network resources, but they usually lack visibility. Bandwidth adaptation, on the other hand, has always been one of the fundamentals research topics in both routing architectures and transport protocols. As of today, endpoints can infer the available bandwidth from their achieved TCP throughput; however, TCP throughput does not necessarily reflect the network available bandwidth, depending mostly on the congestion control strategy adopted. Another way to estimate it is to actively generate probes with specific traffic patterns as extensively documented in literature [16, 63, 40, 48, 58, 19, 26]. We detect some limitations in both these approaches. TCP throughput can be misleading when estimating the available

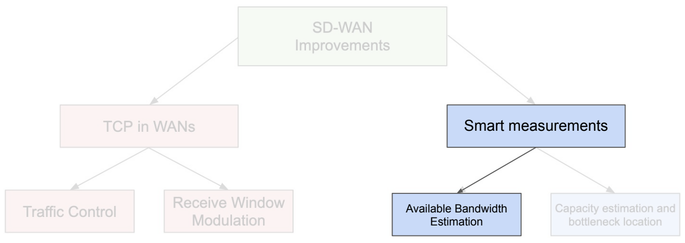

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 6.1 Chapter contribution to the dissertation.

bandwidth. Congestion control algorithms use specific metrics to interpret congestion and adapt their sending rate iteratively. Loss-based congestion controls react to packet loss. Such algorithms, combined with the big buffers of modern routers, can considerably degrade the performance of connections, causing bufferbloat [25]. Their behavior greatly affects the link utilization, finally impeding a correct estimation of the available network resources, as the network state is modified by their operation. More recent developments in TCP congestion control, such as Google's BBR [9], take another estimation strategy, much more promising and effective. However, its way of probing for available bandwidth still affects the network under analysis, consequently altering the available bandwidth estimation as detailed in Section 6.3. On the other hand, many of the developed active probing methods are not suitable for real-world estimation. Their biggest limitation is that they usually need to be deployed in both endpoints of the connection, making them unsuitable for single-sided analysis, especially on routers. Finally, most methodologies assume a fluid cross traffic model, defeating their accuracy in scenarios with highly variable cross traffic patterns.

In this study, we propose a Statistical Approach to Available Bandwidth EStimation, – SABES –, a passive probing method based on the packet dispersion model applied to TCP acknowledgments. Its main characteristics are:

- computationally inexpensive and therefore suited to real time analysis,
- does not need to be deployed at both endpoints,
- no assumptions about the cross-traffic model are made, is data-driven and based on an extensive set of simulations and real-world scenarios,

Possible applications of SABES include active congestion control strategies e.g., active queue management at an edge router or congestion control at the endpoint, and available bandwidth analysis. While it relies on accurate TCP segments time-stamping, we demonstrate that our filtering technique makes it robust even in virtualized environments.

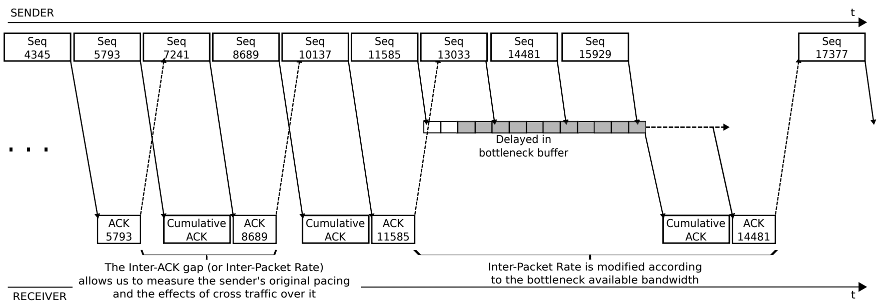

SENDER
t
Seq 4345
Seq 5793
Seq 7241
Seq 8689
Seq 10137
Seq 11585
Seq 13033
Seq 14481
Seq 15929
Seq 17377
. . .
Delayed in bottleneck buffer
ACK 5793
Cumulative ACK
ACK 8689
Cumulative ACK
ACK 11585
Cumulative ACK
ACK 14481
The Inter-ACK gap (or Inter-Packet Rate) allows us to measure the sender's original pacing and the effects of cross traffic over it
Inter-Packet Rate is modified according to the bottleneck available bandwidth
RECEIVER
t

Fig. 6.2 The Inter-Packet Rate is the rate of the originally sent data packets that is possible to infer from their TCP acknowledgments spacing.

6.1 SABES Heuristic

The idea behind SABES is to evaluate the inter-packet arrival time of the acknowledgments of a TCP flow. The ACK number carried in a packet allows the computation of the actual inter-packet rate of the packets that generated each specific ACK pair. For each ACK pair with ACK number ack_{num} we define the the inter-ack bytes as $\delta_{acks} = ack_{num_i} - ack_{num_{i-1}}$ and the inter-packet time δ_t is then: $\delta_t = ts_i - ts_{i-1}$. ts is the ACK timestamp taken either on the egress interface of the receiver or the ingress interface of the sender. We finally define the inter-packet rate - IP_{rate} as:

$$IP_{rate} = \frac{ack_{num_i} - ack_{num_{i-1}}}{ts_i - ts_{i-1}} = \frac{\delta_{acks}}{\delta_t} \tag{6.1}$$

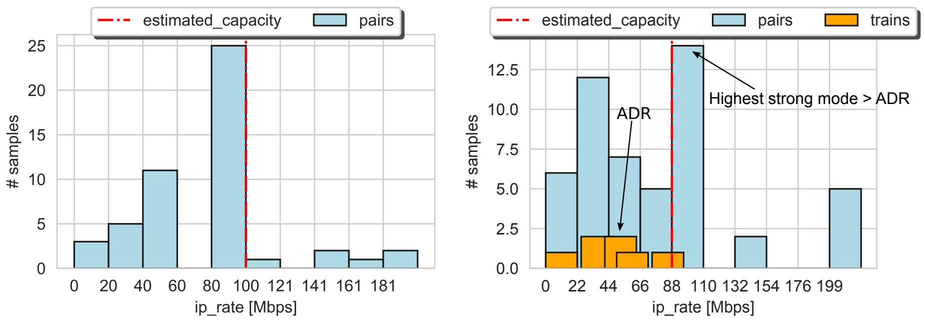
Duplicated and out of order ACKs-derived measurements are discarded. Figure 6.2 provides a graphical representation of the inter-packet gap/rate model using TCP acknowledgements. IP_{rate} can be measured both in sender and receiver. To be noted that in real systems, TCP stacks implement cumulative acknowledgements. Measuring the inter-ACK gap for every ACK-pair comprises the behavior of two or more original packet-pairs sent by the sender. Using the measurements obtained with Equation 6.1, we split the algorithm into two phases. The first phase tries to infer the connection narrow link capacity at the beginning of a TCP connection, while the second one tries to continuously estimate the available bandwidth.

6.1.1 Capacity Estimation

At the beginning of a TCP connection, while the sender congestion window is still growing, packets are sent in bursts with an inter-packet rate that approximates the sender NIC link speed; at that time, packets are said to be sent *back-to-back* and cross traffic is less likely to interfere with such packets; cross-traffic interference is more likely when TCP slow start has grown considerably or during TCP congestion avoidance. However, if there is a link with lower capacity than the sender NIC, packets will be queued before entering this link and the inter-packet rate measured on the acknowledgements will be paced accordingly. We base our capacity estimation technique on the one proposed by *pathrate* in [16]. We study the IP_{rate} distribution of the first α ACKs. Valid values for α vary according to the link capacity, the TCP slow start algorithm, the connection RTT, and the amount of cross traffic in the tight link. We always assume that the TCP flow in analysis is a bulk data transfer and all packets are the size of TCP MSS. We study a worst-case scenario, looking for the minimum amount of packets for α that makes a correct capacity estimation possible. We find that for a 100Mbps narrow link, with 90% cross traffic from multiple Pareto sources, 140ms RTT, and multiple competing flows, that at least 10 acknowledgments are needed to estimate the capacity. To this end, we bind the formula to the NIC capacity C_n, which we assume we can know in advance, being 1Gbps for this specific case. We derive then:

$$\alpha = \frac{C_n}{2 \cdot MSS \cdot RTT} \cdot \varepsilon \tag{6.2}$$

where ε is a constant derived from the data exploration of the previously cited worst case scenario. With such an α value we are able to complete the capacity estimation between 3 and 30 RTTs, with less cycles for larger RTTs. In this way, we can capture enough information to build the histogram statistic while not delaying the capacity estimation to a late moment in the connection lifetime. Being the first packets of a TCP connection sent back-to-back, we can follow the conclusions of [16] and analyze if the distribution highest mode represents more than $\beta\%$ of the total sample data. We choose $\beta = 30\%$ as shown in [16]; this applies to non-congested scenarios where a strong capacity mode is formed, as shown in Figure 6.3a. If the distribution is not found to have a single high mode, we proceed by studying the IP_{rate} generated by ACK-trains: we consider every N acknowledgements instead of each pair and



(a) Capacity estimation is obtained from single strong mode in packet pairs

(b) Capacity estimation is obtained from packet trains dispersion

Fig. 6.3 Capacity estimation following SABES heuristic.

perform the same computation described in Equation 6.1. This is equivalent to using packet trains as done in *pathrate*. We look at ACKs trains of size $N = 4$. In this scenario, IP_{rate} converges closer to the flow throughput; in [16] a similar concept is identified as Asymptotic Dispersion Rate (ADR). We pick the ADR value as the main mode of the distribution obtained from ACKs trains. If multiple mode have the same frequency, we pick the highest value one. We then look for the first mode in the original set of modes obtained from ACK pairs and select the first mode with a value higher than the ADR as capacity value as show in Figure 6.3b. The accuracy of this technique is affected by the bin size selected. In our case, we fix the number of bins to 10 instead of selecting a single bin size.

6.1.2 Available Bandwidth Estimation

SABES main objective is to first identify specific moments in which the IP_{rate} measurements are valid to be used to estimate available bandwidth. Since we are not controlling how the probes are generated, the packet pacing applied by TCP is not always adequate to proceed with the estimation. For example, this happens when the flow has a very low throughput, as per the application generating the traffic itself (e.g. interactive flows or constant bit-rate flows) or because of very high level of concurrency on the machine generating traffic (e.g., more than hundreds of bulky flows). A sliding-window mechanism based on the connection Round-Trip Time

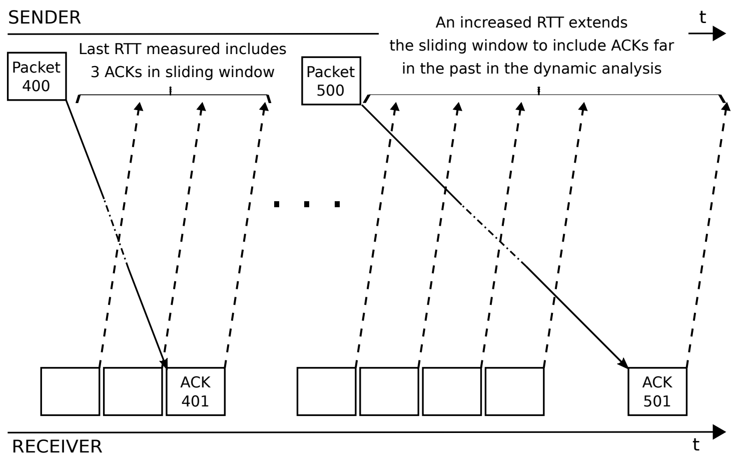

SENDER
Packet 400
Last RTT measured includes 3 ACKs in sliding window
Packet 500
An increased RTT extends the sliding window to include ACKs far in the past in the dynamic analysis
t
ACK 401
ACK 501
RECEIVER
t

Fig. 6.4 The number of samples selected to build the statistic depends on the last measured RTT.

(RTT) is used. When measuring in the sender the exponentially smoothed RTT $sRTT$ is used. In the receiver, the RTT used for the sliding window is measured during the three-way handshake. The dynamic sliding window $\mathscr{D}_i$ of the ACKs IP_{rate} is obtained as defined in Eq. 6.3:

$$\mathscr{D}_i = \{p_k.ip_{rate} | ts_{p_k} \in \mathscr{W}_i\} \tag{6.3}$$

$$\mathscr{W}_i = [ts_{p_i} - \min(1sec, sRTT_i), ts_{p_i}] \tag{6.4}$$

where ts is the packet timestamp, p_i is the last received ACK and $sRTT_i$ is the smoothed RTT computed up to the last RTT measurement. The maximum window horizon is set to 1 second behind ts_{p_i} as defined in Eq. 6.4. A graphical representation of the dynamic sliding window mechanism is provided in Figure 6.4. The result is filtered to remove statistical outliers that can derive from noisy measurements, using a simple inter-quartile range rule over the window in analysis. We discard duplicated ACKs. We finally filter any inter-packet time bigger than twice the RTT, as they are usually associated with retransmission timeouts or slowly growing congestion windows.

We use the samples selected through this dynamic sliding window mechanism - $\mathscr{D}_i$ - to build a statistical distribution of the inter-packet rate in form of an histogram.

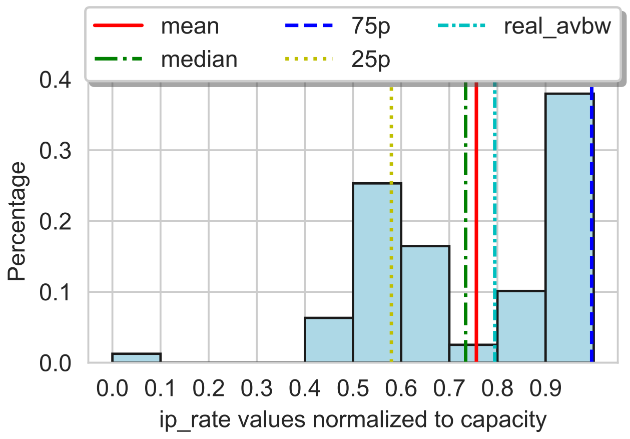


Fig. 6.5 Example of IP_{rate} distribution that matches SABES criteria obtained with the dynamic sliding window.

We studied this histogram for a big simulation data-set and derive an heuristic to infer when the IP_{rate} follows specific patterns. Given $\mathscr{D}_i$ we define its inter-quartile range as $\mathscr{I}(\mathscr{D}_i) = q_{75}(\mathscr{D}_i) - q_{25}(\mathscr{D}_i)$ and its mean-median distance as $\mathscr{M}(\mathscr{D}_i) = |q_{50}(\mathscr{D}_i) - \bar{x}(\mathscr{D}_i)|$, where $q_y()$ and $\bar{x}()$ are the y-quantile and mean operator, respectively. From extensive data-analysis of both simulations and real-world traces we detect that when $\mathscr{M}(\mathscr{D}_i)$ is small and $\mathscr{I}(\mathscr{D}_i)$ is big, $\bar{x}(\mathscr{D}_i)$ is representative of the network available bandwidth. We finally define two parameters ω and γ as the mean-median maximum distance and the minimum distribution spread, respectively. Both are derived from the capacity $\mathscr{C}$ estimated in 6.1.1; we experimentally find that $\omega = 0.1 \cdot \mathscr{C}$ and $\gamma = 3 \cdot \omega$ allow to identify distributions that meet our criteria. The set $\mathscr{B}$ of all available bandwidth estimations obtained from a single TCP flow is defined as:

$$\mathscr{B} = \{\bar{x}(\mathscr{D}_i)\,|\,\mathscr{M}(\mathscr{D}_i) \leq \omega \quad and \quad \mathscr{I}(\mathscr{D}_i) \geq \gamma\} \tag{6.5}$$

In Figure 6.5 is shown an example of valid IP_{rate} sliding histogram following the heuristic criteria. The samples are normalized according to the estimated link capacity. In this case, the mean value actually approximates the available bandwidth with an error of 3.5%.

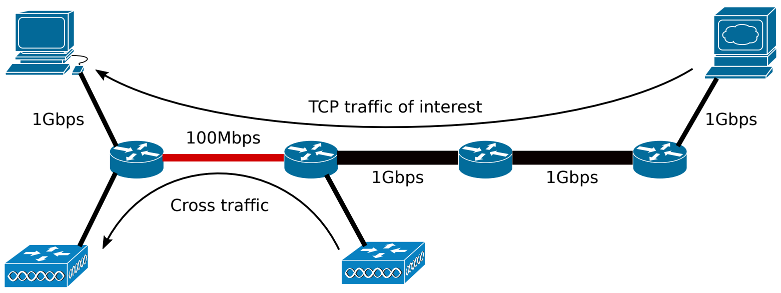


Fig. 6.6 Simulated topology with one single 100 Mbps bottleneck and hop-persistent cross-traffic flowing in the same direction of the main TCP data flow.

6.2 Heuristic Evaluation

6.2.1 Simulation environment

To validate our model, we simulated a network scenario in ns-3 [59]. The simulated dumbell topology consists of multiple intermediate routers and one single bottleneck link at 100Mbps, as shown in Figure 6.6. All the other links have 1Gbps capacity. We simulate scenarios with different values of end-to-end latency, varying between 20ms and 140ms. Cross-traffic is traverses only the bottleneck link (is said to be one-hop persistent). The TCP traffic of interest goes through all topology up to the receiver. Cross-traffic and traffic of interest are generated from different sources and received in different sinks. The cross-traffic generators simulated are both constant bit-rate and on-off Pareto sources. In both cases, cross-traffic is generated so to use 20%, 50%, and finally 90% of the bottleneck capacity. We generate tests with one single cross-traffic flow, up to 30 concurrent cross-traffic sources; cross-traffic flows duration is continuos until the end of the TCP transfers. The TCP traffic of interest flows are bulk data transfers of 1MB, 5MB and 50MB. We test generating only one TCP flow, with 20 and up to 50 concurrent TCP flows generated from the same host. TCP congestion control algorithm is the loss-based TCP-Reno (default in ns-3).

6.2.2 Heuristic simulation results

We show qualitative results for the heuristic in a single TCP flow with 20% Pareto cross-traffic scenario in Figure 6.7. Blue dots represent SABES estimations. The

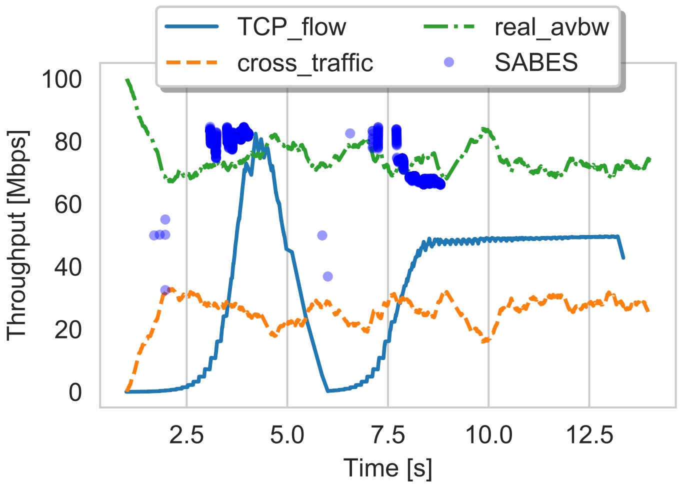


Fig. 6.7 Heuristic application to a single TCP flow, single bottleneck scenario. The mean absolute error of $\mathscr{B}$ is 7.4Mbps.

heuristic provides correct estimations based purely on acknowledgments-derived measurements in moments where the actual TCP throughput is far from matching the real available bandwidth. As a matter of fact in this scenario the loss-based TCP congestion control is never converging to an optimal link utilization. As a matter of fact, this flow TCP throughput diverges from the real available bandwidth with an error of 30Mbps in its congestion avoidance phase. On the other hand SABES heuristic is able to detect when the IP_{rate} measurement is representative of the available bandwidth and the mean absolute error of the estimation set $\mathscr{B}$ is 7 Mbps.

We show the distribution of the mean absolute error of the estimation in Figure 6.8. Estimations are computed over a validation data-set consisting of highly competitive simulations scenarios, with 50 TCP concurrent flows and 30 cross-traffic Pareto sources, using up to 50% and then 90% of the bottleneck capacity. We test this in a topology with an end-to-end RTT of 20ms and then 140ms. Bottleneck capacity is 100Mbps. Results improve in the 90% cross-traffic scenario in respect to the 50%. We deduce that the better estimation obtained in the 90% cross-traffic scenario is

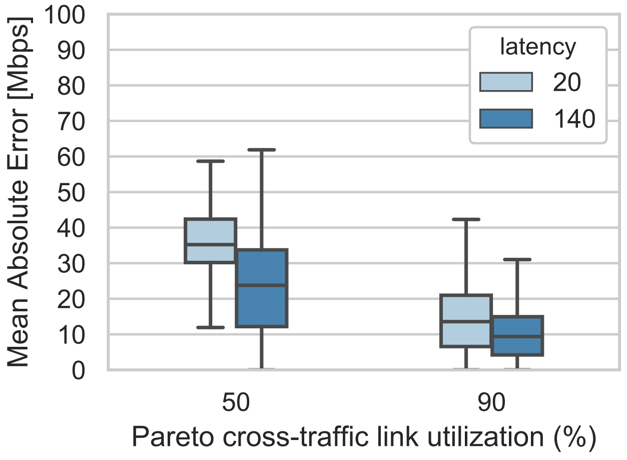


Fig. 6.8 Mean absolute error results of the estimations applying SABES in a simulated environment with 100Mbps bottleneck for different values of cross-traffic link utilization and latency.

due to the average TCP throughput being closer to the actual available bandwidth; in these cases the TCP congestion control is able to converge to a fairly precise value, improving the IP_{rate} statistic. On the other hand the estimations accuracy is independent on the RTT. Results obtained provide good confidence margins in the estimation with a computationally lightweight process implementable as part of a real-time system. SABES can be deployed both close to the sender or the receiver and its estimations are obtained exclusively from passive TCP analysis. However, the results distribution for the 50% cross-traffic case shows a median error value of approximately 30Mbps, which is rather high. In Chapter 7, we investigate the problem and propose an AI-enhanced solution, improving estimation results considerably.

6.3 Related Work in AvBw estimation

While SABES is based on the packet pair dispersion model as exposed in Chapter 5, available bandwidth estimation has been subject of extensive studies in the networking community, comprising other types of models. As seen in Chapter 2 the BBR TCP congestion control operates on the Kleinrock's optimal operating point [44] in which the available bandwidth and the round trip time, RTT, are estimated in order to determine the bandwidth-delay product (BDP). The endpoint probes periodically to estimate the tight link available bandwidth by pacing packets at higher rates than the previous estimation. However, BBR has been proven to build long-term standing queues that can cause misleading BDP estimation [30]. This causes the algorithm to often overestimate the BDP while not being fair to other competing flows [46]. Recent development in BBRv2 try to mitigate the problem, however early studies suggest that it is still sufferring from such problems [21]. We suggest that BBR BtlBw estimation could be improved by embedding packet pair dispersion analysis, whilst their current evaluation is based on the in-flight bytes estimation. SABES available bandwidth estimation strategy could be used as part of a congestion control scheme that preserves higher fairness characteristics such as those guaranteed by loss-based congestion controls. The SABES approach entails detecting moments when the TCP sending rate is higher than the available bandwidth, and analyzing the ACKs inter-packet distribution to detect its value at that time. Under normal TCP operating conditions, this type of behavior can provoke phases where the flow self-induced congestion is still not causing adverse events such as buffer-bloat or packet loss; to estimate the available bandwidth during those phases is beneficial. Such approach is also taken by TCP HyStart [27]: it is a heuristic to find a safe exit point for TCP slow start which infers the available bandwidth based on the clocking of TCP ACKs. HyStart mitigates the losses caused by slow start and it is used as the default slow start algorithm for the CUBIC congestion control; their heuristic includes packet trains being sent back-to-back during the slow start. In addition to what HyStart does, SABES filters out statistical anomalies in clocking of TCP ACKs. SABES can run in real-time during the whole lifetime of the TCP connection and find adequate moments to estimate available bandwidth.

In the area of active probing tools we identify related work using the packet dispersion, PGM, and PRM models.

Pathrate is a tool developed in [16], which is the first work in providing a rigorous definition of the packet disperion model and we extensively base our work on it. Other works include the more recent [35, 49, 54].

As explained in Chapter 5, idea behind PGM is to send packet pairs at a rate equal to the one of the narrow link to then estimate the rate of the tight link. As such, PGM assumes that capacity is known in advance. A few tools implementing PGM foresee an exploratory phase in which capacity is inferred. SABES does the same, exploiting some characteristics of TCP slow start to infer the narrow link capacity. Tools implementing PGM include [63], and [43]. Based on PGM is [40], a work that takes a similar approach as SABES although with some very relevant differences: even though they use TCP acknowledgments to estimate available bandwidth, they take into account both the sending rate of the data packets and the receiving rate of the ACKs, following the probe-gap model proposal. SABES looks exclusively to the ACKs clocking and its distribution to determine the available bandwidth, making it suitable for receiver-side-only TCP based estimation; to the authors knowledge, no other tools implements such an approach. Also, SABES does not need information provided by the connection socket or by the operating system, making it suitable for deployment in routers.

Examples of methods implementing a PRM approach are TOPP [48], pathChirp [58], and BART [19]. pathChirp was one of the first tool developed that used the concept of packet *chirps* which consist of packet trains sent with an exponentially decreasing inter-packet time. This approach aims at generating self-induced congestion in the tight link queue, causing increasing queueing delays. A similar effect is obtained with TCP flows, especially the ones using loss-based congestion controls such as CUBIC, and SABES takes advantage of that. BART reaches a fairly accurate real-time available bandwidth estimation, thanks to the application of Kalman filters for signal denoising, applied to the inter-packet rate measurements. However, the tool is not publicly available, and experiments have been conducted in the scale of tenths of megabits per second, which makes it unsuitable for today network needs. ASSOLO [26] exploits a new technique for signal denoising called Vertical Horizontal Filter (VHF) which proved to be accurate. The tool runs on Linux and needs in-kernel support for real-time task scheduling to guarantee accurate packet pacing.

6.4 Conclusions

In this study we presented SABES, a method to estimate network available bandwidth using passive measurements obtained from TCP traffic. SABES is computationally inexpensive, taking advantage of simple statistical analysis of TCP traffic. It can be deployed just on one side of the TCP connection, being it either close to the sender or the receiver, simplifying its deployment. SABES implements an heuristic that detects IP_{rate} distributions whose mean value approximates the network available bandwidth. SABES model was validated in simulations. We show that SABES heuristic provides a fair estimation of the available bandwidth of our validation data-set with a median mean absolute error of 30% of the bottleneck capacity. SABES is a good candidate for an estimator that is part of a TCP congestion control algorithm or other types of traffic control systems such as the one described in Chapter 4. However, the estimation results obtained have margin for improvement. We investigate an enhanced version of SABES in Chapter 7.

Chapter 7

Deep Neural Networks for AvBw estimation

With the advent of increased computational capabilities Artificial Intelligence has gained a lot of interest in the research community. AI comprises a vast amount of techniques and theories, which are united by the objective of designing algorithms that are able to "learn" from data. Under the umbrella of AI techniques, one that generated considerable interest is the Deep Neural Network (DNN). DNNs are a particular type of Artificial Neural Network (ANN) with multiple neuron layers between the network input and output. They are an effective tool in modeling complex non-linear problems starting from data samples of the specific domain analyzed.

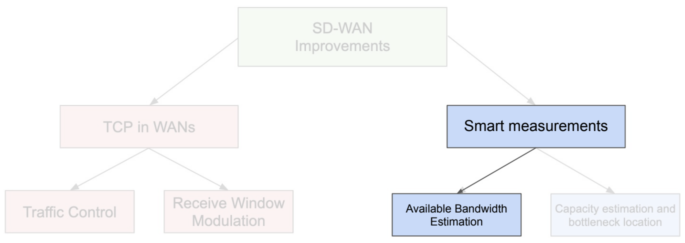
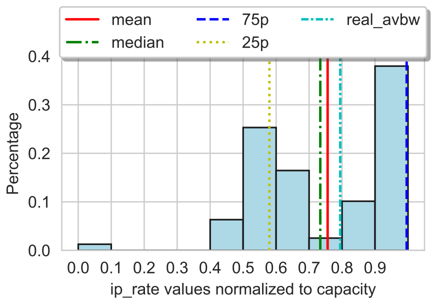
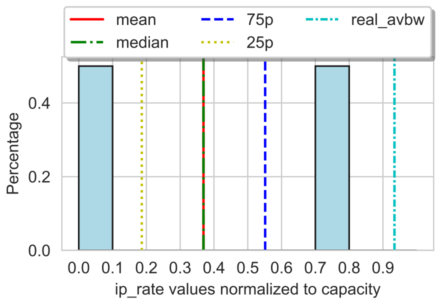

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 7.1 Chapter contribution to the dissertation.



(a) Mean is representative of the available bandwidth



(b) Mean is not representative of the available bandwidth

Fig. 7.2 The DNN classification problem.

With the development of SABES-NN we improve the proposal of Chapter 6 in the following ways:

- we use a neural network to filter out misleading measurements,
- we improve the results obtained by the heuristic approach proposed in Chapter 6 three-fold,
- we compare it to other state-of-the-art tools providing more precise results, up to 10x.

7.1 Neural Network design

The heuristic described in Chapter 6 tries to identify distributions shaped like the one in Figure 7.2a, looking for a mean value which is representative of the available bandwidth. However, there are distributions that match the heuristic criteria but that do not contain any significant mode close to the available bandwidth, such as the one shown in Figure 7.2b. The mean of such a distribution gives a 50% error in the estimation and no actual values are present in the bin closest to the distribution mean. The histogram shapes of Figure 7.2a and Figure 7.2b are notably different. To distinguish between them we train a DNN. DNNs have proven to be effective as classifiers when the input features are non-linear and with high dimensionality,

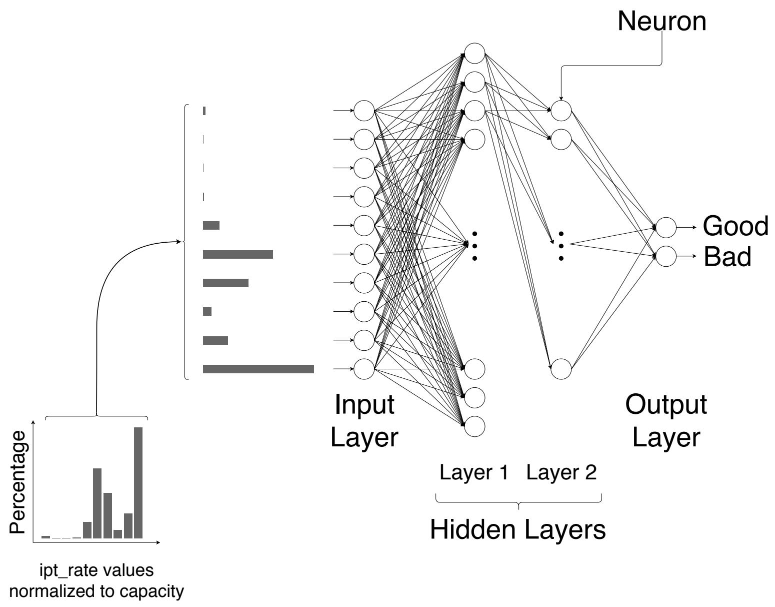
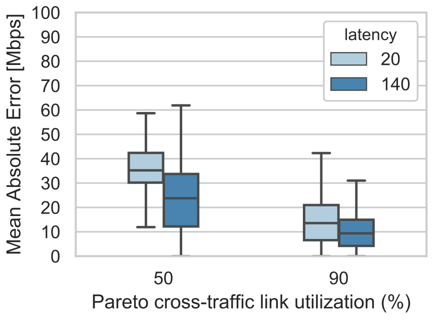


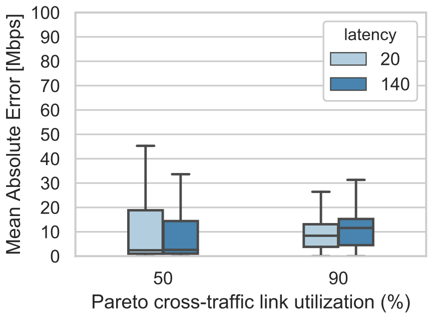
Fig. 7.3 Deep Neural Network acting as histogram classifier for available bandwidth estimation.

such as the sliding histograms produced by SABES heuristic. We train a neural network to classify good and bad histogram distributions; the generalization is obtained by normalizing the data to the estimated capacity and the total number of samples obtained in the sliding window. In this way, the neural network input data is independent of the link capacity and the sample size. We fix the histogram bins number to 10. This translates in a lower resolution when representing IP_{rate} distributions of high capacity links (i.e. more than 500Mbps), although we opt for this compromise to keep the neural network of reasonable size.

We run SABES heuristic and get the complete normalized sliding histograms of the simulations described in Section 6.2.1. We obtain over 140000 sliding histograms used as training data-set for the neural network. We run a supervised-learning training process labeling all the histograms *good* whose mean value provides an available bandwidth estimation with an error lower than $\pm 10\%$ of the bottleneck capacity, and as *bad* the other histograms. We identify that distributions that overestimate or underestimate have a similar shape, as far as the estimation error is small. The neural



(a) Heuristic only.



(b) Neural Network.

Fig. 7.4 Estimation results for SABES and SABES-NN applied to our validation data-set.

network used is a deep neural network, consisting of two hidden layers as shown in Figure 7.3. The network inputs are 10 neurons, one per normalized bin of the sliding histogram. The network output are two neurons, one per class - good and bad.

Following the generalization concepts exposed in [15] we find the minimum number of neurons per each hidden layer that avoids over-fitting. Given the neural network number of inputs N_i and outputs N_o, we use as upper boundary for the number N_h of the hidden layers neuron, the result of Equation 7.1:

$$N_h = \frac{N_s}{(\sigma * (N_i + N_o))}, \quad 2 \leq \sigma \leq 10 \tag{7.1}$$

The activation function for the input and hidden layers neurons is a rectified linear unit (ReLU) while the output layer activation is a normalized exponential function, also known as softmax.

7.2 Evaluation over simulation dataset

After training, we run again SABES but with the additional filter provided by the neural network. The validation data-set is the same used for the evaluation of the heuristic alone. The results obtained are shown in Figure 7.4b. The boxplot shows the improved accuracy provided by the usage of the neural network. The 50th

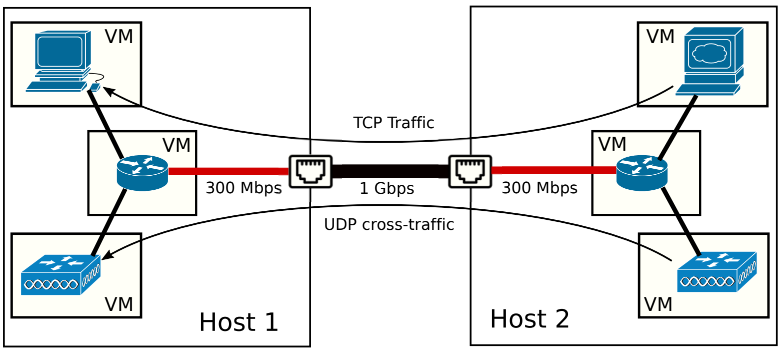


Fig. 7.5 Real testbed topology with one single 300Mbps bottleneck.

percentile of the estimations error is below 10Mbps in all the scenarios proposed, compared to the 30Mbps obtained when using the heuristic alone. While training a neural network can be computationally expensive, we quantify that adding the neural network evaluation increases the computation execution time only by 6% in respect to the heuristic alone.

7.3 Evaluation in real testbed

We finally tested SABES-NN in a real testbed. The testbed topology is a dumbell as shown in Figure 7.5. It is composed of six virtual machines, two for TCP traffic generation, two for UDP cross-traffic generation, and finally two acting as intermediate routers and shapers. The VMs are deployed in two different physical hosts. The two hosts are connected through a 1Gbps link, but we apply a traffic shaper of 300Mbps to the egress interfaces of both intermediate routers. We also add an artificial delay of 20ms between the two physical hosts using the Linux Traffic Control module NetEm. All VMs run Debian Linux; Host 1 is running KVM as hypervisor, while Host 2 has VMWare ESXi. TCP traffic is generated as large file transfers of 100 MB over HTTP. UDP cross-traffic is generated with the D-ITG tool [2]. We chose this tool as it allows to generate traffic following a Pareto distribution of choice as done in simulation. We capture the traffic in both sender and receiver VMs with tcpdump, which uses Libpcap as backend library. Hardware timestamps are enabled in the

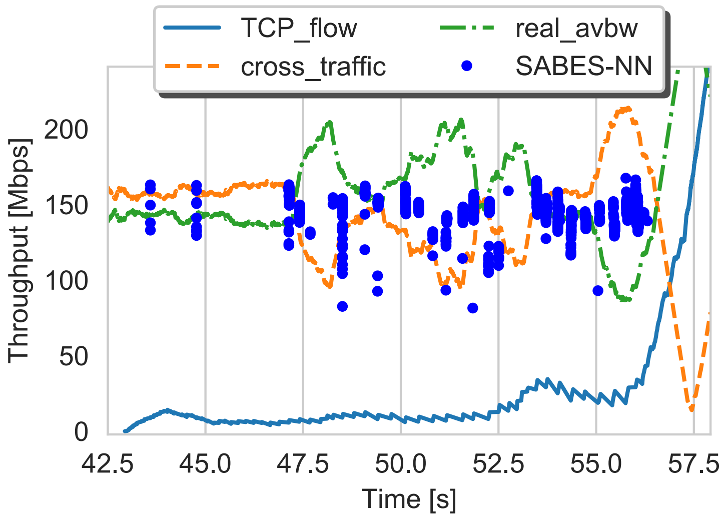


Fig. 7.6 SABES-NN running in a real testbed. Estimations mean absolute error is 8% of the bottleneck capacity.

hosts NICs and supported by the virtio driver used in the VMs, allowing for precise timestamping even in virtualized environment.

7.3.1 Real testbed results

In Figure 7.6, the results of SABES-NN estimations are shown for a single test. The test consists of three concurrent HTTP/TCP file transfers of 100 MB competing for a bottleneck link of 300Mbps with one UDP cross-traffic flow. The throughput of only one of the three TCP flows is shown. SABES-NN effectively detects moments where is possible to infer the real available bandwidth, even though the flow throughput is far lower. The mean absolute error of the $\mathscr{B}$ estimation set is less than 10% of the bottleneck capacity.

7.3.2 Comparison with ASSOLO

In section 7.3.1 SABES-NN has been tested in a real environment and it was possible to see how its estimation performs in comparison to a TCP flow throughput. In this

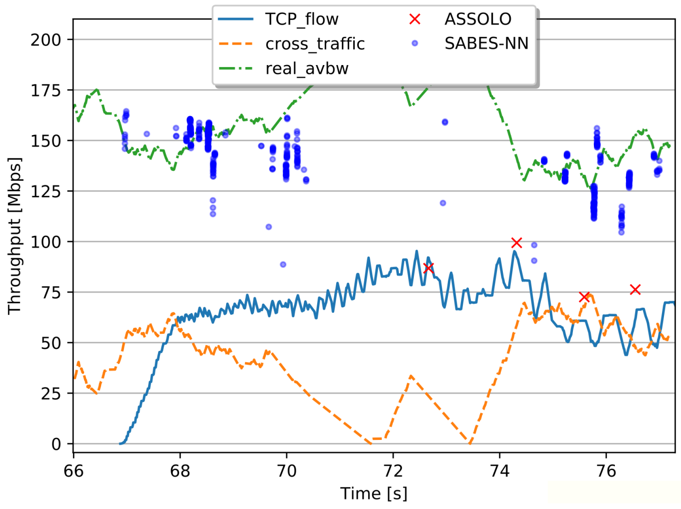


Fig. 7.7 SABES-NN compared to ASSOLO estimations.

section we compare SABES-NN with a modern active probing tool for available bandwidth estimation, ASSOLO [26]. ASSOLO follows a probe-rate model - PRM - to detect the bending point of the probe-gap model curve as shown in Figure 5.4. This means that it iteratively sends trains of packet probes at higher rates, looking for the point where $g_{out}/g_{in} > 1$. This means that ASSOLO, as most active probing tools, needs to be deployed in both endpoints of the connection to measure both g_{in} and g_{out}. ASSOLO also relies on very accurate timestamping, and its usage is advised only in systems that can run a real-time version of the Linux kernel. However, this is not the case for virtualized environments: even with a real-time kernel installed in the guest, the host hypervisor can still preempt the VM itself. ASSOLO assumes a fluid cross-traffic model with little variability and an average throughput value that is relatively stable during the duration of its estimation. We measure estimation period of ASSOLO of being around 20 seconds in our testing environment. During such a period of time cross-traffic can vary considerably, and this is the case for the experiment proposed.

In Figure 7.7 SABES-NN is compared to ASSOLO. The test scenario is the same described in Figure 7.6 but this time we run ASSOLO concurrently with the

TCP traffic. We detect that ASSOLO cannot provide accurate measurements when setting the intermediate shaper to values bigger than 200 Mbps so we reduce it to this value. In Figure 7.7 results are shown. During the 20 seconds of execution ASSOLO provides 5 estimations points. Its absolute mean error is of 30% of the bottleneck capacity (66 Mbps). SABES-NN continuos evaluation of passive TCP traffic provides around 1500 estimation points gathered from all the three concurrent TCP transfers. In Figure 7.7 only one TCP transfer is shown. The absolute mean error of SABES-NN estimations is of 3% of the bottleneck capacity (6 Mbps).

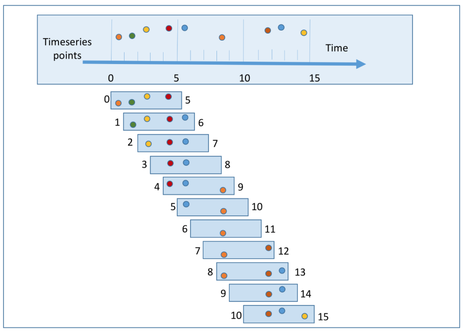
7.4 Improving SABES-NN: a direct estimation approach

We tried a different approach leveraging Neural Network to improve SABES-NN performance and scalability. First we tried reducing its computational footprint by switching from a per-packet sliding window analysis to a tumbling window one as explained in 7.4.1. A second step, exposed in 7.4.2, involved training a neural network to obtain a direct estimation of the available bandwidth instead of building a discriminator, bypassing the heuristic implemented in the original SABES proposal.

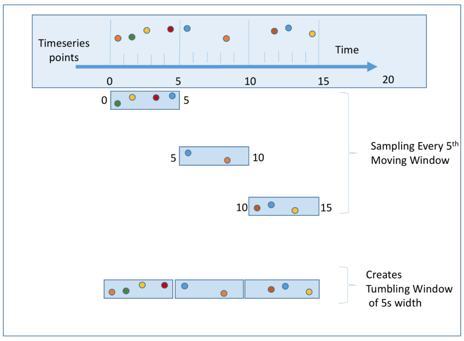
7.4.1 Time series analysis

In Figure 7.8 the two moving window models are shown on a simplified time series analysis. The sliding window mechanism exposed in section 6.1.2 follows the scheme shown in figure 7.8a. For each incoming packet, the window is moved forward of a single step, however, as exposed in the previous chapter, the window left margin is dependent from the last measured RTT. This causes consecutive windows to partially overlap as shown in Figure 7.8a. As noticeable, with a sliding window scheme (Fig. 7.8a), for the example time series, we obtain 11 sample windows. This accounts for a very precise picture of the time series evolution and allowed us to obtain more than 140000 histogram samples from our training set. However, in a prototypal implementation, running the complete heuristic and neural network discriminator in real-time with a per-packet sliding window proved having poor scalability.

To reduce the computational overhead we implemented a tumbling window scheme. As shown in Figure 7.8b, the tumbling window is triggered periodically and



(a) Sliding window time series analysis.


Timeseries points
Time
0
5
10
15
20
Sampling Every 5th Moving Window
Creates Tumbling Window of 5s width

(b) Tumbling window time series analysis.

Fig. 7.8 Different type of moving window for time series analysis; the tumbling window reduces the resolution but improves scalability. Source: [66].

consecutive windows never overlap. As for the sliding window case, we additionally limit the left window margin based on the RTT measured with the last packet of the current window. The tumbling window approach allowed our prototype to scale efficiently in real-time traffic analysis. A rigorous quantification of such improvement is still pending. However, as shown in Figure 7.8b we obtain only 3 windows with this method, compared to the 11 windows obtained with the sliding window approach. When applying the tumbling window analysis to the offline neural network training process, the number of samples obtained from the training set is reduced from $\sim$ 140000 to $\sim$ 15000, i.e. one order of magnitude less. In section 7.4.3 we will show how this affects the results obtained with SABES-heuristic and SABES-NN. In section 7.4.2 we present a new DNN approach trained from this reduced training set that shows the best performance so far in terms of precision and number of estimations obtained, that we call SABES-KDE.

7.4.2 SABES-KDE

Together with the tumbling window mechanism modification to SABES, we decided to research new ways to take advantage of the mathematical modeling capabilities of Neural Networks when studying network available bandwidth. We identify one of the limits of SABES-NN residing in the histogram representation. While it is easy to compute and store, the histogram representation is very dependent from the bin size selected. In SABES-NN we opted for a fixed number of bins, instead of a fixed bin size. This allowed us to obtain a representation independent from the end-to-end network capacity. To generalize the neural network trained with such histograms we also normalized the bin values range to the capacity value estimated in the first part of the algorithm (see 6.1). This type of representation poses two main challenges: a) the histogram resolution gets coarser at higher capacity values and b) we need correct capacity estimations to obtain reliable available bandwidth estimations. We address the former by using a Kernel Density Estimation representation of the probability density function obtained from the IP_{rate} measurements. KDEs allow for a finer representation of the distribution, with a lower loss of details than histograms. To be noted that KDEs are not agnostic to the resolution problem as the shape of the function is strongly affected by the kernel type selected and its bandwidth, being the last one the equivalent of the histogram bin size - see [62] for additional information on KDE. For SABES-KDE we use a gaussian kernel with a bandwidth $\beta = 0.2$. To address the capacity estimation problem we decided to normalize the KDE to a fixed capacity value, which is big enough to include values in a range significant for our application domain. Specifically, we opted to normalize to $C_{norm} = 1Gbps$, independently from the actual narrow link capacity. An example of the representation obtained with this methodology is shown in Figure 7.9. The narrow link capacity of the testbed from which the sample is taken is 100Mbps, actually capping the KDE to a 0.1 maximum value in respect to the selected C_{norm}.

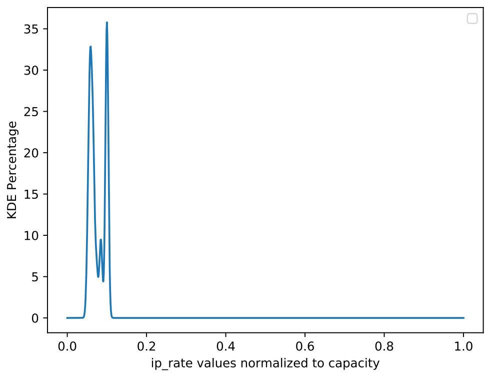
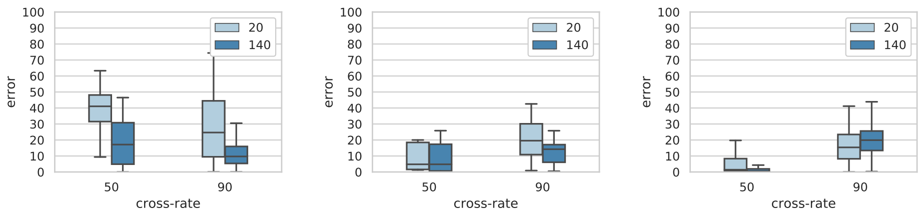


Fig. 7.9 SABES-KDE function obtained during a single tumbling window.



(a) SABES- 1103 estimations.

(b) SABES-NN - 91 estimations.

(c) SABES-KDE - 6352 estimations.

Fig. 7.10 Validation set estimation error using the tumbling window and different estimation strategies.

Using this representation during the tumbling window analysis, we obtained a series made of 10 random samples selected from each normalized KDE function. We trained a DNN with 10 inputs and a single output ReLu neuron which provides the direct available bandwidth estimation in the range [0,1]; the actual available bandwidth is then obtained by multiplying by C_{norm}. SABES-KDE neural network has a lower dimensionality than the SABES-NN one because we use Equation 7.1 to select the hidden layers size. The hidden layers dimension derived from Equation 7.1 is dependent from the input training set size, which is one order of magnitude less for SABES-KDE compared to SABES-NN. We tried to train SABES-NN neural network with the reduced tumbling window training set but it could not converge to a satisfactory training accuracy, so we kept the neural network originally trained with the 140000 sliding window samples to compare the estimation results.

7.4.3 SABES-KDE results evalution

To evaluate the modifications introduced with the tumbling window and SABES-KDE, we run SABES, SABES-NN, and SABES-KDE against the simulated validation set already used in 6.2.2 and 7.2. As shown in Figure 7.10a, the pure SABES heuristic approach in combination with the tumbling window provides poor performance with a mean absolute error of 26.52%. Distribution whiskers go up to 70%. With the heuristic filter we obtain only $\sim$ 1000 estimations from the original data set of $\sim$ 6000 tumbling windows. Adding the SABES-NN discriminator filter

Method %	Mean Absolute Error	% samples
SABES	26.52%	17.4%
SABES-NN	16.37%	1.4%
SABES-KDE	10.07%	100%

Table 7.1 All SABES flavours estimation error compared.

improves considerably the heuristic estimation accuracy as shown in Figure 7.10b. The mean absolute error in this case goes down to 16.37%, however we retain just 91 samples of the original 6000. Figure 7.10c shows the error distribution obtained from SABES-KDE. Performance is improved compared to SABES-NN, with a mean absolute error of 10.07%. As SABES-KDE does not filter in any way the input representation we retain all the analysis samples. All these results are summarized in Table 7.1.

7.4.4 SABES-KDE limitations and future work

The preliminary evaluation results of SABES-KDE shown in Section 7.4.3 are promising. The tumbling window approach is scalable enough for real-time analysis, but its loss in resolution over the time series is challenging for SABES and SABES-NN. SABES-KDE is the only flavour that has shown excellent estimation performance in conjunction with the tumbling window scheme while providing a continuos available bandwidth evaluation, even when analyzing passive TCP traffic. However, this first iteration of SABES-KDE is based on a simplification of the KDE representation, which is the normalization by a fixed C_{norm} value, and not the actual narrow link capacity. This limits the neural network applicability to the normalization range. Also, the training set must have observations in the full normalized range to achieve a satisfactory generalization, even in the selected range. For example, we trained SABES-KDE with a training set made exclusively of experiments with a narrow link of 100Mbps while keeping $C_{norm} = 1Gbps$. We then applied all SABES flavours to a simulated validation data set with a narrow link capacity of 500Mbps. In this case, the mean absolute error of SABES-KDE was of 54%, against the 16% obtained with SABES-NN because SABES-KDE neural network was not trained with samples in the 500Mbps range. To be noted that SABES-NN is not trained with any sample in the 500Mbps range either, but normalizing the histogram against the estimated narrow link capacity allows its neural network to achieve good generalization performance. A version of SABES-KDE using an estimation of the narrow link capacity will be the focus of future research.

7.5 Related Work in DNN and ML usage for AvBw estimation

In recent years a few approaches in available bandwidth estimation involving neural networks and machine learning have been proposed. Notably, a first approach using NNs is presented in a paper from 2005 [20]. The authors used a traces dataset to train an artificial neural network to predict the actual network bandwidth and compare it to a system called Network Weather Service (NWS), which provides seasonal network resources availability based on statistical metrics. They improve the NWS predictions with the NN approach, however they need to train a different NN for each network under study, making this approach not scalable nor general.

In [10] a different Machine Learning approach is used: the authors opted for a Support Vector Machine (SVM) model used to solve a regression problem given the input of a set of experiments to obtain the available bandwidth estimation. We also considered SVMs to implement SABES-NN discriminator, looking to exploit the binary classification capabilities of such model, but results were not satisfactory. This work also takes an active probing approach, obtaining interesting results when using packet-chirps as probes (as done in the notorious pathChirp [58]). Chirps are probing packet trains where the inter-packet rate varies exponentially in a given probing range. SABES-NN heuristic and neural network identify inter-packet rate distributions of TCP connections that resemble the ones of active probing chirps.

Khangura et al. in [41] provide a first approach to available bandwidth estimation using a shallow neural network that directly provides the estimation. As the case for SABES-NN, the neural network input is normalized by a δ factor, providing a scale invariant neural network. However, their approach is still based on an iterative active probing method that generates probes at a given rate. SABES-NN provides estimation based on TCP passive measurements and its neural network detects histogram shapes that are valid to extrapolate estimations, it does not provide a direct value for the available bandwidth as it is in [41]. They provide additional validation of their proposal in [42]. Finally, in [39] the same authors take a reinforcement learning approach. The active probing generator uses a reward function that tries to maximize the relationship between the sending rate r_{out} and the actual network available bandwidth. This approach is promising and could be applied as part of the control loop of transport protocol congestion control algorithm and we will consider similar approaches in our future investigations.

7.6 Conclusions

In this chapter we presented an available bandwidth estimation technique that leverages Deep Neural Networks, improving the heuristic proposed in Chapter 6. It works by detecting patterns in the distributions obtained through the dynamic sliding windows of SABES. The DNN acts as discriminator of histograms that are better suited for the estimation. SABES-NN improves the estimation results over the validation set reducing the median of the mean absolute error distribution down to 10% of the estimated bottleneck capacity, compared to the 30% obtained through the heuristic alone. SABES-NN, as SABES, relies on a good initial estimation of the bottleneck capacity. The approach proposed to estimate the end-to-end narrow link capacity through the first ACKs of the TCP connection is based on a heuristic proposed in [16]. We then studied an alternative approach training a neural network to provide a direct estimation of the available bandwidth, SABES-KDE. The input representations of this DNN are KDEs normalized by a fixed C_{norm} value, making it independent from the actual narrow link capacity estimation. While the results obtained are promising, its generalization capabilities are poor. Better results could be achieved normalizing by the narrow link capacity instead of a fixed value. This motivated us to study alternative ways to obtain precise narrow link estimations in Chapter 8 through the definition of a new packet dispersion model.

Chapter 8

Narrow link estimation and location

In todays Internet massive amounts of data are being moved due to data hungry applications, for example multimedia content streaming which, combined with a reduction in cost and augmented reliability for high speed broadband access, result in new challenges for the future Internet. Even though the Internet core is largely over-provisioned, parts of the network can still represent a bottleneck in presence of cross-traffic coming from multiple sources, especially in access links. End-to-end capacity determination, together with available bandwidth estimation, has always been a key research topic in the networking community [18, 16, 37, 38, 35, 63, 26]. Such measurements are useful to improve bandwidth allocation, packet scheduling, and congestion control, both in endpoints and routers. Capacity estimation is of

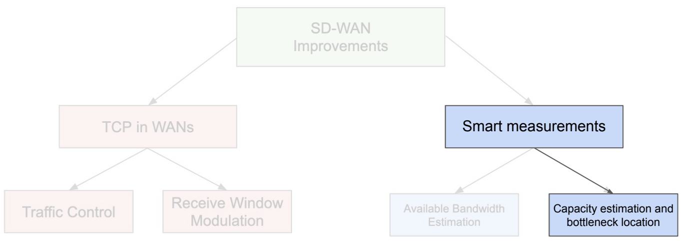

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location

Fig. 8.1 Chapter contribution to the dissertation.

the uttermost importance in modern systems that rely on dynamic algorithms for resources allocation, as, for example, 5G mmWave networks. Such environments, and wireless environment in general, present considerable challenges as the resources allocated to the different network devices can frequently vary. Capacity estimation techniques can also be effectively exploited to verify network sizing and planning in enterprises WAN or even detecting Service Provider throttling in residential connections, making such a measurement a very effective tool to be deployed in as SD-WAN device.

In this chapter we propose a novel approach to the capacity estimation problem. Our technique improves state of the art analysis of packet pair dispersion, introducing the concept of *packet pair dispersion delay*. The novel concept of *hidden* packets is also introduced. It consists in sending packets with smartly crafted TTL, to later improve the accuracy of the estimation with a series of machine learning approaches. As a consequence, this technique provides a twofold benefit; on the one hand it allows to compute the link with the smallest available capacity on the path, namely, the narrow link. On the other hand, it provides the specific hop in the network where such narrow link is located, giving a very accurate picture of the network status and topology. An important feature of the proposed contribution is that it can be exploited end-to-end, while requiring a modest amount of network and computational resources, making it suitable for real world usage. We leave as an important part of our future work the implementation of the proposed solution in a real environment. We validate both the accuracy and the feasibility of our algorithm through simulation of environments with realistic traffic loads and cross-traffic events, where we obtain an estimation error for the narrow link capacity of less than 1% in most scenarios.

8.1 Theoretical bases

8.1.1 Smallest Link Capacity Set

Given an end-to-end path we represent it as a oriented weighted acyclic graph from a source to a destination node. Links weights are the links capacities. We define the Smallest Link Capacity Set (**SLCS**) as the set of all the links of an end-to-end path which capacity value constitute the narrow link of a complete-path-subset; a

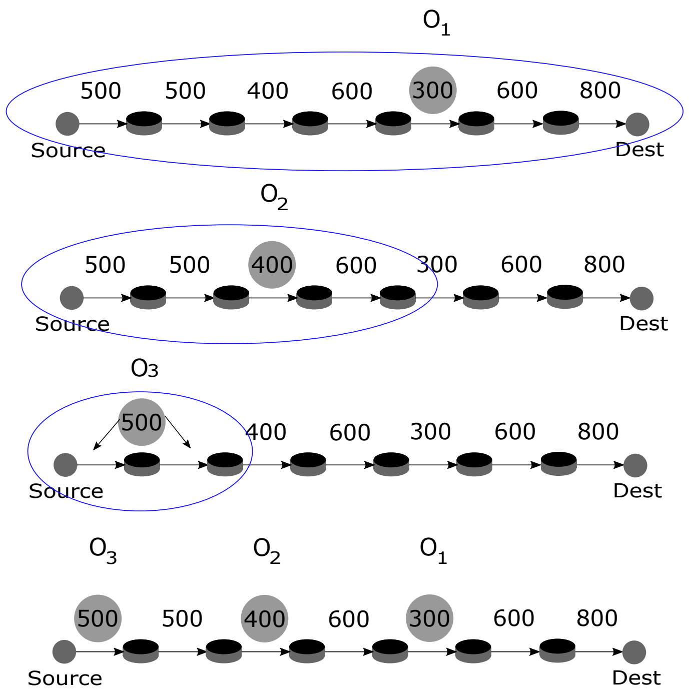


Fig. 8.2 Smallest Link Capacity Set - **SLCS**. Capacities are expressed in Mbps.

complete-path-subset is any sub-segment of the end-to-end path that includes the source node. If in a complete-path-subset multiple consecutive links have the same capacity value, they are considered as a single link. Figure 8.2 shows a 7-hop path with different link capacities (500, 500, 400, 600, 300, 800, 600 *Mbps*) from source to destination. Thus, the **SLCS** is $\{O_1, O_2, O_3\}$ where $O_1 = (300Mbps, link_5)$, $O_2 = (400Mbps, link_3)$, $O_3 = (500Mbps, link_1)$. End-to-end RTT is 280ms. Our new representation exposed in 8.1.3, is able to estimate the capacity value of all the link in the SLCS, starting with the end-to-end narrow link. It achieves this by actively generating packet pair probes and using a new type of packet pair dispersion analysis.

8.1.2 Packet pair dispersion

Here we summarize the packet pair dispersion model introduce in Chapter 5. Packet pair dispersion is the basis of many network inference framework. To obtain such measurement, pairs of probe packets of a given size Z are sent over an end-to-end path between two end-points at a given rate R. To be able to measure the narrow link capacity value C_n, R must be higher than C_n. The dispersion of a packet pair τ is the time interval between the instant the last bit of the first packet and the instant the last bit of the second packet of the probing pair is received at the destination. We derive the *inter-packet rate* as:

$$IP_{rate} = \frac{Z}{\tau} \tag{8.1}$$

Typically, bandwidth estimation tools collect a certain amount of measurements derived from Equation 8.1 either from actively probing the network or by means of passive measurements. They then build a histogram representing its distribution during the measurement period. Various techniques have been tested throughout the years to remove noise from these distributions and extrapolate the end-to-end available bandwidth or its narrow link capacity. We introduce a new representation that takes into account not only the packet pair dispersion but also the propagation delay that each of the pair packets incur, and finally estimate the narrow link capacity based on three distributions: the packet pair dispersion distribution and each packet propagation delay distribution.

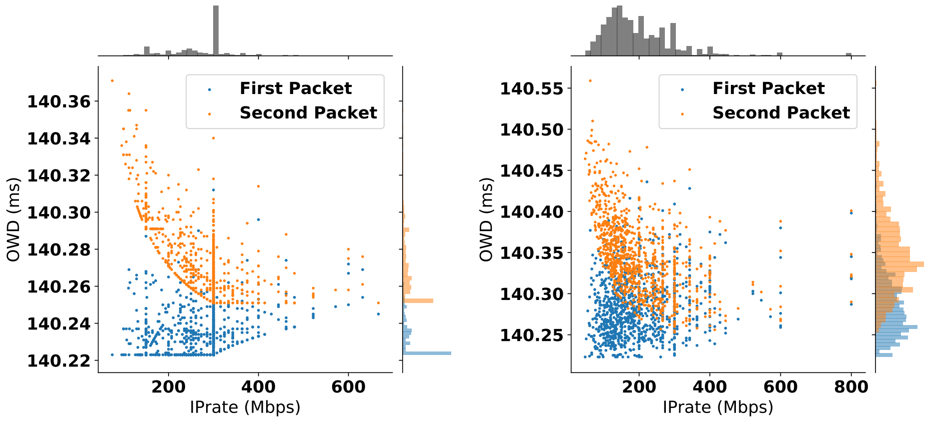
8.1.3 Packet pair dispersion-delay

Consider a N-hop path defined by two sequences: capacities sequence $\mathbf{C} = \{C_0, C_1, \ldots, C_N\}$, and delays $\mathbf{D} = \{D_0, D_1, \ldots, D_N\}$. In a end-to-end context, we can only get measures on the sender and the receiver end points. When a packet pair reaches the receiver, we can compute the one-way delay (**OWD**) of each packet of the probing pair: OWD_1 for the first packet received, and OWD_2 for the second packet. When packets traverse a network their propagation delay is affected by different factors: the physical propagation delay in the transmission medium and the processing delay at each hop (serialization, routing, etc...) constitute the deterministic delay, D_{det}. Packets can also incur in buffering delay caused by capacity impairment between links that we define as D_{buffer}. Finally, packets can be delayed due to possible congestion along the path which we define as the stochastic delay D_{sto}. As a result, each probing pair packet OWD is obtained as $OWD = \sum_{i=0}^{N} D_i$. where the delay at each hop i, D_i is given by:

$$D_i = D_{det,i} + D_{buffer,i} + D_{sto,i} \tag{8.2}$$

Every time a pair of packets is sent across the end-to-end path, it can be affected by cross traffic (stochastic delay D_{sto}) and/or by limitations of a link capacity (buffering delay D_{buffer}). These two factors can increase the propagation delay of either one or both packets of the pair, also affecting the relative distance τ at which the pair was sent from the source causing variations in the IP_{rate} measured in the receiver. The three values of OWD_1, OWD_2, and IP_{rate} configure the *packet pair dispersion-delay* (**ppdd**) triplet that we use to find the narrow link capacity of an end-to-end path so that each $ppdd_i = (OWD_{1,i}, OWD_{2,i}, IP_{rate,i})$. Figure 8.3a shows a representation for the metric triad. Blue dots represent the first packet and orange dots the second packet. On the y-axis we plot the OWD of each probe, while on the x-axis is the IP_{rate} of the each probing pair. On the top and right side of the plot are shown the histograms of IP_{rate}, OWD_1, and OWD_2 respectively. The measurements are obtained from a simulation generated in the ns-2 simulator. The simulated topology is the one shown in Figure 8.2. 1000 ICMP packet pair probes of size $Z = 1500B$ and an $IP_{rate} = 1Gbps$ are generated. We generate cross-traffic along the path as UDP packets of 1500B. The generation pattern follows a Pareto distribution with a shape value $\alpha = 1.9$. Cross traffic is generated by 16 sources-destination pairs between *each* link (following [16] nomenclature cross-traffic is *one-hop persistent*). In Figure 8.3a packet pair triplets (IP_{rate}, OWD_1, and OWD_2) are plotted in a scenario with an average 20% of each link capacity occupied by cross-traffic.

In the absence of cross traffic the only factor adding delay to the packet pair is the capacity limit of the narrow link. When this happens, only the second packet of the packet pair is affected, resulting in an increased value for OWD_2. This is, $OWD_2 = D_{det} + D_{buffer}$ and $OWD_1 = D_{det}$, where D_{det} is made up with deterministic delays and D_{buffer} is the buffering delay caused by the capacity limitations of the bottleneck. In Figure 8.4, the blue dashed line represents the minimum value acquired by the first packet. These packets are not affected by buffering, so that $OWD_1 = D_{det}$. The red dashed line represents the minimum value acquired by the second packet of the probing pair, which is affected by buffering at the bottleneck due to the capacity limit, giving $OWD_2 = D_{det} + D_{buffer}$. Taking into account that the narrow link capacity is $300Mbps$, which corresponds to an inter-packet of $40\mu s$ with $1500B$ of packet size, and the inter-packet sent is $12\mu s$ (1Gbps), it is possible to observe that the distance between the blue dashed line and the red dashed line corresponds to



(a) 20% of cross traffic.

(b) 70% of cross traffic.

Fig. 8.3 Packet pair delay dispersion representation under different network conditions. Cross-traffic is one-hop persistent.

$D_{buffer} = 28\mu s$. The sum of the distance between the blue and red dashed lines and the inter-packet sent is equal to the inter-packet of the narrow link. It is worth noting that the maximum value of IP_{rate} achieved by the packets laying over the blue dashed line is the narrow link capacity. Similarly, the minimum value in IP_{rate} achieved by the second packets laying over the red line is the narrow link capacity. Both values coincide over the vertical black dashed line , which is the capacity of the narrow link on the path, O_1. We distinguish three zones in the relative behavior of the packet pair dispersion and delay: when the IP_{rate} is below the narrow link capacity C_n, D_{sto} due to cross-traffic has a major impact on the second packet of the probing pair, causing a dilation of the packet pair dispersion, and so a lower IP_{rate}. For IP_{rate} values exactly at C_n, D_{sto} affects both packets in the same way, causing either none or a constant incremental offset in the delay. IP_{rate} values bigger than C_n are due to cross-traffic in the links posterior to the narrow link. In this case the first packet of the pair tends to reduce its relative distance in respect to the first packet, causing a reduction in the measured packet pair dispersion. It is worth noting the shape of the packet pair dispersion-delay for the two packets: when $IP_{rate} < C_n$ the second packet **ppdd** distribution follows a concave form; for $IP_{rate} > C_n$ the second packet follows a linear form. The first packet follows a linear form for $IP_{rate} < C_n$

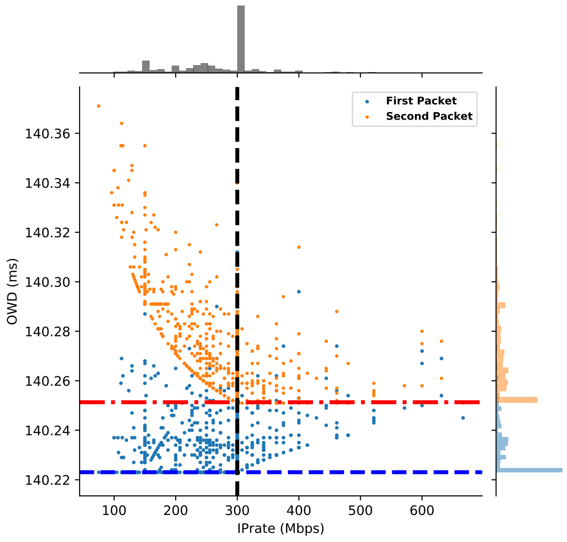


Fig. 8.4 Red dashed line represents $OWD_2 = OWD_{det} + D_{buffer}$. Blue dashed line is $OWD_1 = OWD_{det}$, not affected by additional delays. Vertical black dashed line is the capacity of the bottleneck on the path, O_1.

and a convex form for $IP_{rate} > C_n$. This behavior will be the focus of a future work on the technique to improve its results.

While in the scenario shown in Figure 8.3a the capacity mode could be detected visually and numerically from the regular packet dispersion, the situation changes in presence of higher level of cross-traffic. In Figure 8.3b we show the same scenario as Figure 8.3a but with 70% of cross traffic. To be noted that in nowadays networks such amount of cross-traffic in a one-hop persistent fashion is hardly found. We simulate such a scenario to recreate noisy conditions that can be found in real-world environments which are related to the difficulties involved in performing precise measurements, especially in time-stamping. In such a scenario the capacity mode is not identifiable with a simple visual inspection anymore. Also the OWD distributions vary substantially. The overall **ppdd** distributions shapes do not follow the pattern described previously in a clear way. In Section 8.2 we define a heuristic approach that, combined with state of the art machine learning techniques, allows to identify the capacity mode in the IP_{rate} distribution with a high level of confidence even in noisy environments.

8.2 Narrow link capacity determination

To determine the capacity of the narrow link we take into account the *ppdd* triplets distributions, reducing the area of interest around the capacity of the narrow link. To achieve so, we find a set of ppdd, the $\mathscr{S}_{ppdd}$, so that its triads OWD_1 and OWD_2 values are concentrated in the low part of their distributions. We define the three distributions as $\mathscr{B}_{IPr}, \mathscr{B}_{owd1}, \mathscr{B}_{owd2}$ for the IP_{rate}, OWD_1, and OWD_2 measurements respectively. To isolate the measurements from relevant packet pair probes we select a quantile value q_x from $\mathscr{B}_{owd1}$ that can be statistically close to the linear form that B_{owd1} describes when the respective pair IP_{rate} values is lower than C_n. We find experimentally that when $x = 30$, the quantile q_x provides a good approximation over a wide range of scenarios, up to 90% of cross-traffic. We then look for the a low quantile q_k in $\mathscr{B}_{owd2}$. We define the $\mathscr{S}_{ppdd}$ set as:

$$\mathscr{S}_{ppdd} = \{ppdd_i \mid OWD_{1,i} < q_{30}(\mathscr{B}_{owd1}) \; and \; OWD_{2,i} < q_k(\mathscr{B}_{owd2})\} \tag{8.3}$$

where $OWD_{1,i}, OWD_{2,i} \in ppdd_i$. The percentile q_k is found iteratively with unitary increments of k until it satisfies the condition of Equation 8.4:

$$\left\{k \mid len(\mathscr{S}_{ppdd}) > \alpha\right\} \tag{8.4}$$

where α is the minimum number of packets that we find to be representative to build the statistics. For this work we find $\alpha = 5$ experimentally. The selection $\mathscr{B}_{sel}$ of the $\mathscr{B}_{IPr}$ distribution is finally extracted from $\mathscr{S}_{ppdd}$:

$$\mathscr{B}_{sel} = \left\{IP_{rate,i} \mid IP_{rate,i} \in \mathscr{S}_{ppdd}\right\} \tag{8.5}$$

Once $\mathscr{B}_{sel}$ is obtained as in Equation 8.5 we apply a kernel density estimation (KDE) transformation over it. KDE is a method to estimate the probability density function of a random variable obtained from samples of such distribution. The

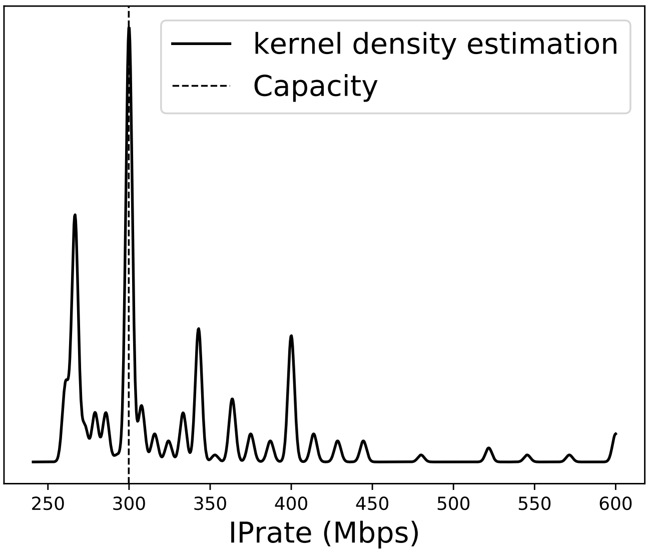


Fig. 8.5 Capacity determination with Kernel Density Estimation based on the $\mathscr{B}_{sel}$ selection obtained with the proposed heuristic.

main control parameters for KDE methods are the kernel type and bandwidth. For this study we use a gaussian kernel with bandwidth set to 2. Further optimizations could be obtained with a fine tuning of the KDE bandwidth. Once obtained the KDE continuos distribution, we can search for its global maximum and use it as narrow link capacity estimation. In Figure 8.5 this technique is applied to the case shown in Figure 8.3b. With 70% of cross-traffic finding the capacity mode by simply inspecting the $\mathscr{B}_{IPr}$ distribution would have been very difficult. Thanks to the ppdd representation we provide the exact value of the narrow link capacity even in such a noisy environment.

8.2.1 Capacity estimation evaluation

We provide an evaluation of the ppdd model narrow link capacity estimation performance in a variety of simulation scenarios. In Table 8.1 we show the narrow link capacity estimation C_n obtained from measurements generated in the topology described in Figure 8.2. As explained in Section 8.1.3, cross-traffic is generated in

CT %	$C_{n,f}$	$C_{n,r}$
10%	300.00	300.00
20%	300.00	300.00
30%	300.01	300.00
40%	300.01	300.01
50%	300.05	299.99
60%	300.08	300.00
70%	300.02	300.03
80%	300.09	295.02
90%	299.97	292.66

(a) C_n estimation with increasing amount of cross-traffic.

#probes	$C_{n,f}$	$C_{n,r}$
1000	300.02	300.06
500	299.92	286.13
100	299.90	284.98
50	299.99	282.39

(b) C_n estimation when reducing the amount of probing pairs.

Table 8.1 ppdd narrow link capacity estimation summary. Nominal C_n is 300Mbps. $C_{n,f}$ is the estimation obtained with fixed cross-traffic packet size of 1500B, while $C_{n,r}$ with variable size cross-traffic packets.

form of UDP traffic, following a Pareto random distribution. 16 sources of cross-traffic inject traffic over each link of the path, in a one-hop persistent fashion. The probing pairs are ICMP Echo Reply packets of fixed size $Z = 1500B$. To obtain our estimation we measure OWD_1, OWD_2, and IP_{rate} in the destination node.

In Table 8.1a we show the estimation results obtained with increasing values of cross-traffic when generating 1000 ICMP probes of 1500B. The first column C_n is with cross-traffic packets of fixed size of 1500B. The column $C_{n,r}$ shows the results when the cross-traffic packets size varies following a uniform random distribution with values between 40B and 1500B. As shown, the estimations are almost always exact, independently from the network conditions. We see a slight underestimation in presence of 80% and 90% random cross-traffic, where ppdd estimation has an error of less than 3%.

In Table 8.1b we show the technique results when reducing the number of probes generated from 1000 down to only 50. The cross-traffic amount is 70%, a rather high value. The worst estimation is when using only 50 probes; the result produced has a marginal error of approximately 6% against a reduction of 95% of the amount of probing traffic generated.

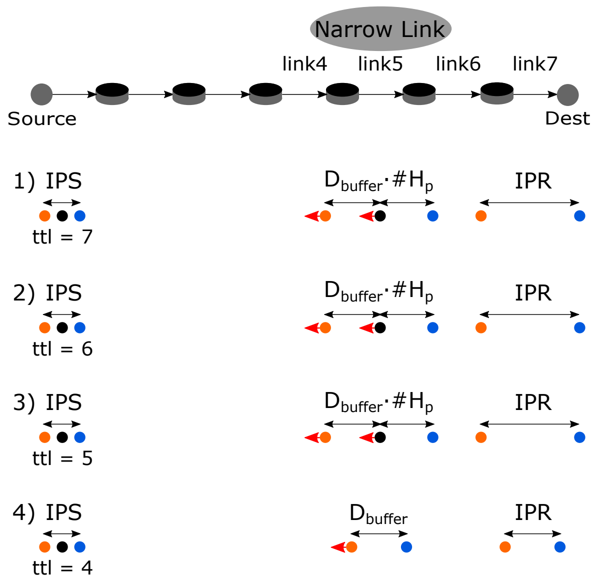


Fig. 8.6 Schematic example of the Hidden Inter-packet Red-shift Effect (HIRE) over a non-congested path.

8.3 Hidden red-shift effect

In this section we define a technique that will be applied both for locating the narrow link position on the path and to estimate the capacity of the rest of links belonging to the SLCS. The technique consists in injecting *hidden* packets H_p between the probing pairs generating packet probing trains. All train packets have size Z. The additional H_p packets are said to be hidden as they are generated with a Time To Live (TTL) value in their corresponding IP header field that will cause their expiration during transit, before reaching the destination. The first and last packet of the train are generated with an **IPS** $\equiv IP_{rate,sent} = Z/\tau$. In absence of cross-traffic along the path the dilation effect caused by the hidden packets over the probing pair dispersion measured in the receiver ($IP_{rate,recv}$ that we call **IPR**) is proportional to the amount of injected hidden packets and the links queueing delay D_{buffer} added up to the node where the hidden packets expired. Taking advantage of this effect, we can evaluate the differences in the IPR behavior varying iteratively the TTL value of the hidden packets. This enables both narrow link location and SLCS capacity estimation.

8.3.1 Narrow link location

Figure 8.6 shows an example that summarizes the HIRE technique. In this case, four iterations are needed to determine the narrow link. Three packets are sent: blue (first packet pair probe), black (hidden packet), and orange (second packet pair probe). Blue and orange have a TTL value big enough to reach the destination. Black packets are the hidden packets and have different TTL at each iteration. Three packets are sent with an IPS rate value between the blue and orange packets equivalent to the narrow link C_n capacity. Additionally, a hidden packet with $TTL = 7$ is sent between the blue and orange packets. When the train of packets reaches the narrow link at hop-5, both orange and black packets suffer a red-shift effect due to D_{buffer}. As a result the inter-packet time between all packets of the train increases. At the same time OWD_1 and OWD_{Hp} increase. When the probing train reaches hop-7 the hidden packet is dropped leaving only the blue and orange packets. The packets are finally captured at the destination with a rate IPR. The process is repeated iteratively reducing the TTL value. Up to $TTL = 5$ the IPR value measured in the destination will be similar. When the hidden packet is generated with a $TTL = 4$, it will expire when reaching hop-4 just before the narrow link. The IPR measured in the destination will increase. Only the orange packet suffers the *red-shift effect* resulting in an increase in the packet dispersion equal to D_{buffer}.

To detect this change of behavior we repeat the described process generating a number of probing trains N_t per each TTL value with a fixed amount of hidden packets $\#H_p$. In Figure 8.7 we apply a heuristic to the 20% cross-traffic scenario to locate the narrow link position. Per each group of $N_{t,i}$ of TTL_i we compute the mean of the IPR values measured by each train. We then use a simple heuristic to detect the behavior change between the groups post and pre narrow link. We start adding the computed mean values to a distribution and compute its standard deviation. If, when adding a new mean value of the TTL_i group, the standard deviation increases more than a given threshold γ, we detect a behavior change and select TTL_{i-1} as the location of the narrow link. For this study we use $\gamma = 0.2$.

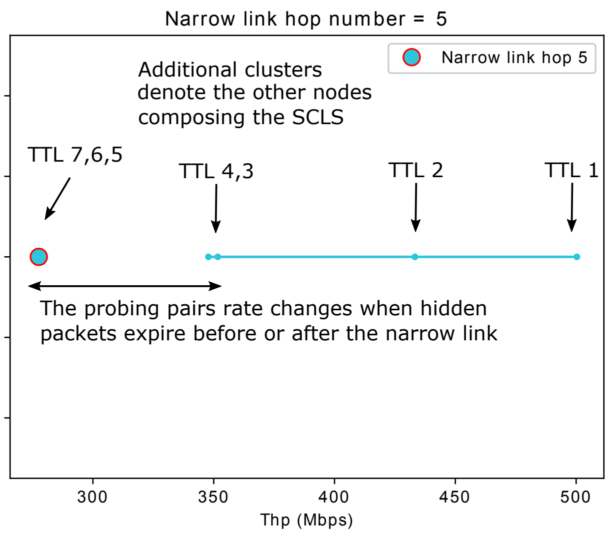


Fig. 8.7 HIRE applied with $N_t = 20$ and $\#H_p = 5$. The narrow link position is detected when the probing train dispersion behavior changes due to hidden packets expiration.

8.3.2 SLCS capacity determination

Thanks to the hidden red-shift effect, HIRE is able to determine the capacity of the rest of links of the Smallest Link Capacity Set, SLCS. To achieve it, we need to evaluate first the narrow link capacity and its location. Once we obtain them, we generate another probing session with $IPS = C_n$. The hidden packets should also be generated so that they expire before reaching the narrow link. In this way we guarantee that the probing trains will be forwarded through the narrow link without being buffered there. However they will see a dilation in their dispersion causing the measured IPR to follow Equation 8.6:

$$IPR = \frac{Z}{(\tau + D_{buffer}) \cdot (\#H_p + 1)} \tag{8.6}$$

It is then possible to estimate the capacity $C_{n,i}$ of the node i of the path by generating the hidden packets H_p with TTL i and measuring the IPR_i following

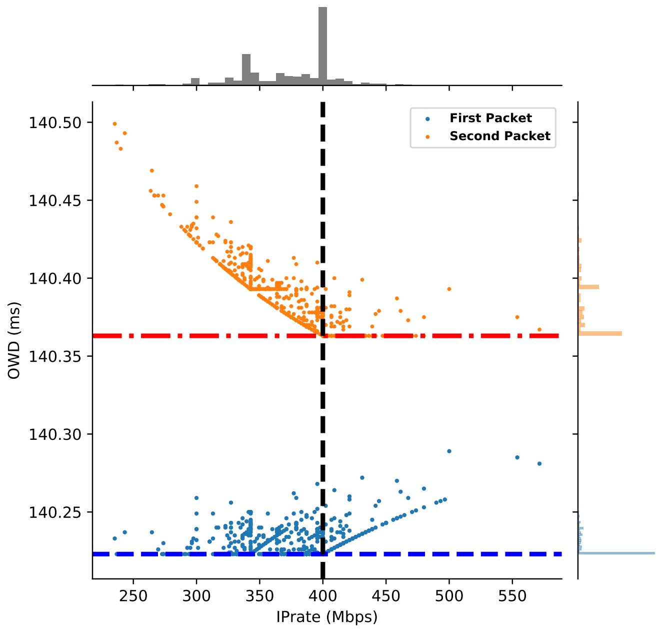


Fig. 8.8 Capacity determination of the O_2 link of the SLCS.

Equation 8.7:

$$C_{n,i} = IPR_i \cdot (\#H_{p,i} + 1) \tag{8.7}$$

Figure 8.8 shows this technique applied to estimate the capacity of the second SLCS link of the topology of Figure 8.2, indicated as O_2. The simulation environment has 20% of cross-traffic. Probing pair packets are generated with $IPS = 300Mbps$. Five intermediate hidden packets of 1500B are injected in between each pair. Finally, $TTL_{Hp} = 3$ so that the hidden packets will expire right after O_2 but before reaching O_1. The value provided is exactly the capacity of the O_2 link.

8.4 Related Work

Literature regarding packet pair dispersion and capacity estimation is extensive. As of today many schemes for the narrow link capacity estimation have been proposed [18, 16, 37, 38, 35]. Most models, either based on active probing or passive measurements, measure the dispersion between packet pairs or packet trains. Packet pairs dispersion has been proven to be a noisy measurement [16, 17, 35]. The effort of most works in this area have been focusing on signal processing and de-noising of the intern-packet rate distribution derived from packet pairs dispersion. Few approaches derive available bandwidth measurements by looking at signatures in the queuing delay generated by probe trains emitted with specific patterns [58, 55]. In this work we propose a novel approach to capacity estimation that takes into account both packet dispersion and propagation delay on a tri-dimensional plane. Such approach, combined with state of the art machine learning techniques, provides very precise estimations for the narrow link capacity.

Our study also addresses the narrow link position location on the end-to-end path. Few techniques such as the one presented in [31] provides a solution to determine the position of the currently congested bottleneck links but none before had provided the position of the narrow link as well as its capacity. By taking advantage of TTL expiring packets injected in between probing pairs and the packet dispersion-delay analysis, a new method for narrow link location is presented. When adding *hidden* expiring intermediate packets, the probing pairs change their rate when crossing the narrow link, causing a *red-shift* effect that allows for narrow link location. Such a measurement is dependent from precise packet time-stamping which is often a problem, especially in current SDN virtualized environments. Opposed to that, our approach simplifies deployment, easing the tool adoption. The approach for narrow link capacity estimation and location is validated in simulations, showing the tool robustness in very competitive environments, with irregular cross-traffic patterns occupying up to 90% of the end-to-end path.

8.5 Conclusions

In this chapter we presented ppdd and HIRE, which are new approaches for estimating the narrow link capacity of an end-to-end path based on probing pairs. Their main contributions include a new way of interpreting the packet pair dispersion in conjunction with each pair packet propagation delay that we call the *packet pair dispersion delay* - **ppdd**. We also present the *hidden packets red-shift effect* which consists in injecting TTL expiring packets in between of probing pairs to cause a rate reduction when crossing specific links along the path. Based on this two new concepts we develop different techniques to estimate the narrow link capacity, as well as its location along the path. HIRE can even provide the capacity estimation of other links previous to the narrow link. We validate HIRE in a simulation environment with realistic characteristics. In scenarios with up to 70% random cross-traffic, HIRE provides exact narrow link estimations, with an error lower than 1%. We prove its estimation precision even in spite of very high level of cross-traffic (up to 90%) while using very few probes, with a negligible estimation error of less than 6%. HIRE method provides very precise measurements for the narrow link capacity, which is a prerequisite to obtain a good available bandwidth estimation from SABES, the tool presented throughout chapters 6 and 7. In the future, the new packet pair dispersion delay model proposed in this chapter could be adapted to passive TCP analysis, improving the results obtained also for available bandwidth estimation.

Chapter 9

Towards a Stateful SD-WAN traffic controller

In this thesis we presented different contributions that address both control and network status assessment from an edge router perspective such as it could be an SD-WAN node. Following the initial proposal depicted, we investigated two main topics: TCP optimization in WANs and improved WAN visibility thanks to smart measurements. In the following sections we present a proposal for a stateful edge router architecture that takes advantage of all the exposed building blocks to effectively enhance the end user quality of experience thanks to an SD-WAN device as shown in Figure 9.1.

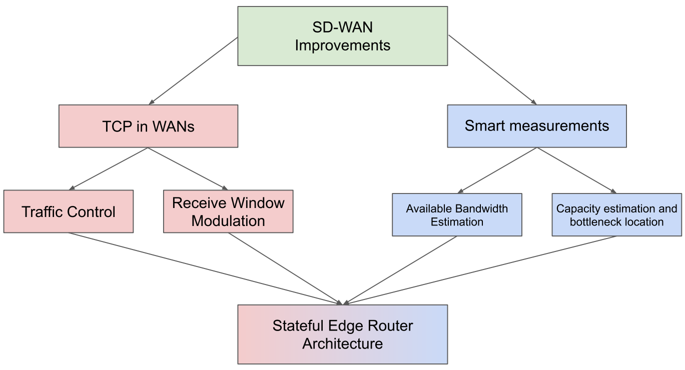

SD-WAN Improvements
TCP in WANs
Smart measurements
Traffic Control
Receive Window Modulation
Available Bandwidth Estimation
Capacity estimation and bottleneck location
Stateful Edge Router Architecture

Fig. 9.1 The research project final proposal for a Stateful Edge Router Architecture.

9.1 Stateful Edge Router Architecture

The Stateful Edge Router Architecture (SERA) consists of a modular architecture to build a scalable and extensible stateful software edge router that optimizes network flows based on the network state. In Figure 9.2 we introduce its main building blocks. SERA main component is the Control Strategy Evaluator (CSE) which decides the best combination of control strategies to optimize the traffic. CSE inputs come from a variety of sources; where the most relevant ones are:

- Live network monitor: measurements of network metrics of the flows currently handled by the software router, such as inter-packet arrival time, burstiness, throughput, etc...
- Traffic classifier: a module able to classify the incoming traffic flows type based on the statistics fed to it by the network monitor.
- Active probing engine: a module that scans the network resources by proactively sending probes, e.g. to estimate network congestion and bottlenecks toward a specific destination.
- Traffic policies: a set of user defined policies to assign priorities to specific types of traffic; these policies could be optimized by means of a learning process.

The CSE is extensible by design, thus it shall support new input generators. Its objective is to find the optimal control technique or set of techniques that maximize network resource exploitation while not harming latency sensitive flows. Overall, our proposal seeks to improve the users Quality of Experience.

SERA scheme enables multiple actionable control strategies to be implemented and chained, e.g. Rate Control, proxy or ECN, as shown on the right side of Figure 9.2. New incoming flows trigger the re-evaluation of the control status. A specific control strategy can be assigned to a new incoming flow, which also may affect the strategy applied to the other ongoing flows.

SERA could optimize specific connections by proxying them; this could improve considerably TCP throughput when the endpoints cannot reach link utilization with a

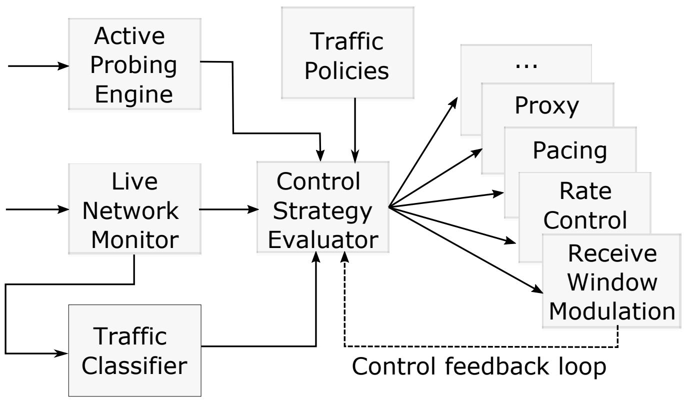


Fig. 9.2 Software Stateful Edge Router Architecture.

single TCP transfer due to a limited maximum receive buffer set in the TCP stack. It is a very common scenario given that the Linux default maximum TCP receive buffer is 4MB. This buffer becomes a bottleneck for a connection over a Long Fat Networks. If SERA detects that specific endpoints have reduced TCP buffer sizes, it can systematically start proxying their connections and locally tune TCP buffers to reach link utilization, acting as a Performance Enhancing Proxy (PEP) (see Chapter 2). On the other hand, if it detects connections being too aggressive in their behavior it could apply a scheme such as the Receive Window Modulation (RWM - Chapter 4) to effectively limit their throughput according to the end-to-end BDP or the user defined policies which were reconciliated in the Traffic Policies component. The end-to-end BDP can be estimated by means of passive traffic analysis performed by the Live Network Monitor applying algorithms such as SABES or SABES-NN (Chapter 6 and 7). Other periodic measurements performed by the Active Probing Engine provide additional information for the CSE, such as the end-to-end narrow link capacity, that represents the maximum threshold for the RWM control. Such measurement is obtained by means of active probing and applying the packet-pair dispersion-delay model (ppdd) described in Chapter 8.

9.2 Takeaways and final remarks

In the previous section we depicted a comprehensive architecture for a smart SD-WAN device. Its components can be implemented following the proposals presented in this thesis. Specifically, in this research project we presented a series of original contributions, including:

- The **Receive Window Modulation** scheme: RWM is a control mechanism that exploits TCP flow control. It allows to throttle TCP connections without dropping packets, independently from the sender congestion control algorithm. This addresses a criticality of AQM schemes towards modern scalable TCP implementations such as BBR. We proved that it can improve loss based congestion control throughput in a variety of scenarios up to 70%. It reduces end-to-end latency, mitigating bufferbloat up to 2.5x compared to uncontrolled TCP connections. It does not need cooperation from the end-points interpreting specific congestion signals as it relies on standard TCP semantics, simplifying its adoption.
- A **Statistical Available Bandwidth Estimation** algorithm: SABES proposes a new approach to available bandwidth estimation, following the packet pair dispersion model applied to passive TCP traffic analysis. It is based on a heuristic aimed at detecting inter-packet dispersion distributions of TCP traffic that allow to identify the network available bandwidth. An improved approach based on Deep Neural Networks allows SABES to obtain less than a 10% estimation error in more than 50% of the cases under analysis.
- The **Packet-pair dispersion-delay** model: the *ppdd* model is a bi-dimensional representation of a set of packet pairs sent as active probes, that shows both the packet pair dispersion and the absolute delay of each pair packet. It allows to identify with precision the end-to-end narrow link capacity thanks to a heuristic approach based on state of the art machine learning methodologies. The technique was then extended with the addition of intermediate TTL expiring probes, enabling the **Hidden Packets Red-shift Effect**. HIRE is able to locate the narrow link logical position in the end-to-end path, increasing the topology visibility.

Future work based on the results obtained in this project includes:

- Comparison in terms of fairness and throughput of RWM and common AQM schemes, especially in presence of scalable TCP congestion controls such as BBR.
- Deployment and study of a distributed network of nodes implementing RWM. We describe such a scenario in [60]. The RWM semantics could induce emergent behaviors in distributed networks that could make them converge to a full and fair network resources distribution.
- Extension of the SABES-KDE approach presented in 7.4.2 to reach broader generalization of the trained neural network.
- Application of the ppdd model to passive TCP traffic in search of a better representation to measure both narrow link capacity and available bandwidth.
- Implementation of the SERA controller, leveraging all the contributions exposed in this dissertation. Such a job implies additional research to implement components such as the Traffic Classifier and the Traffic Policies reconciliator.

References

[1] Use linux traffic control as impairment node in a test environment (part 2), Aug 2018. URL https://www.excentis.com/blog/use-linux-traffic-control-impairment-node-test-environment-part-2.

[2] Stefano Avallone, S Guadagno, Donato Emma, Antonio Pescapè, and Giorgio Ventre. D-itg distributed internet traffic generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.* IEEE, 2004.

[3] Fred Baker and Gorry Fairhurst. Ietf recommendations regarding active queue management. *draft-ietf-aqmrecommendation-11 (work in progress)*, 2015.

[4] S Bensley, D Thaler, P Balasubramanian, and L Eggert. G. judd," data center tcp (dctcp): Tcp congestion control for data centers. Technical report, RFC 8257, DOI 10.17487/RFC8257, 2017.

[5] John Border, Markku Kojo, Jim Griner, Gabriel Montenegro, and Zach Shelby. Rfc3135: Performance enhancing proxies intended to mitigate link-related degradations, 2001.

[6] Jesper Dangaard Brouer. Network stack challenges at increasing speeds.

[7] Martin A Brown. Traffic control howto. 2006. URL https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html.

[8] N Cardwell, Yuchung Cheng, S Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. Bbrv2: A model-based congestion control. In *Presentation in ICCRG at IETF 104th meeting*, 2019.

[9] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control. *Queue*, 14(5): 50:20–50:53, October 2016. ISSN 1542-7730. doi: 10.1145/3012426.3022184.

[10] Ling-Jyh Chen, Cheng-Fu Chou, and Bo-Chun Wang. A machine learning-based approach for estimating available bandwidth. In *TENCON 2007-2007 IEEE Region 10 Conference*, pages 1–4. IEEE.

[11] Francesco Ciaccia, Oriol Arcas-Abella, Diego Montero, Ivan Romero, Rodolfo Milito, Rene Serral-Gracia, and Mario Nemirovsky. Improving tcp performance and reducing self-induced congestion with receive window modulation. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019.

[12] Francesco Ciaccia, Ivan Romero, Oriol Arcas-Abella, Diego Montero, René Serral-Gracià, and Mario Nemirovsky. Sabes: Statistical available bandwidth estimation from passive tcp measurements. In *2020 IFIP Networking Conference (Networking)*, pages 743–748. IEEE, 2020.

[13] Francesco Ciaccia, Ivan Romero, René Serral-Gracià, and Mario Nemirovsky. Hire: Hidden inter-packet red-shift effect. In *2020 IEEE Global Communications Conference*. IEEE, 2020.

[14] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. Pi2: A linearized aqm for both classic and scalable tcp. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 105–119, 2016.

[15] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[16] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *Proceedings IEEE INFOCOM 2001*. IEEE, 2001.

[17] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions On Networking*, 12(6):963–977, 2004.

[18] Allen B Downey. Using pathchar to estimate internet link characteristics. *ACM SIGCOMM Computer Communication Review*, 29(4):241–250, 1999.

[19] Svante Ekelin, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, J-E Mangs, Bob Melander, and Mats Bjorkman. Real-time measurement of end-to-end available bandwidth using kalman filtering. In *2006 ieee/ifip network operations and management symposium noms 2006*. IEEE, 2006.

[20] Alaknantha Eswaradass, X-H Sun, and Ming Wu. A neural network based predictive mechanism for available bandwidth. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 10–pp. IEEE, 2005.

[21] Ferenc Fejes, Gergő Gombos, Sándor Laki, and Szilveszter Nádasy. On the incompatibility of scalable congestion controls over the internet. In *2020 IFIP Networking Conference (Networking)*, pages 749–754. IEEE, 2020.

[22] Mike Fisk and Wu-chun Feng. Dynamic right-sizing in tcp. Technical report, Los Alamos National Lab., Los Alamos, NM (US), 2001.

[23] Matt Fleming. A thorough introduction to ebpf, 2017. URL https://lwn.net/Articles/740157/.

[24] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413, 1993.

[25] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 2011.

[26] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a new method for available bandwidth estimation. In *2009 Fourth International Conference on Internet Monitoring and Protection*. IEEE, 2009.

[27] Sangtae Ha and Injong Rhee. Taming the elephants: New tcp slow start. *Computer Networks*, 2011.

[28] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008. ISSN 0163-5980. doi: 10.1145/1400097.1400105.

[29] Stephen Hemminger et al. Network emulation with NetEm. In *Linux conf au*, pages 18–23, 2005.

[30] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2017.

[31] Ningning Hu, Li Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. Locating internet bottlenecks: Algorithms, measurements, and implications. *ACM SIGCOMM Computer Communication Review*, 34(4):41–54, 2004.

[32] IO Visor Project. BCC: BPF Compiler Collection. . URL https://github.com/iovisor/bcc.

[33] IO Visor Project. Xdp: express data path. . URL https://www.iovisor.org/technology/xdp.

[34] Van Jacobson, Robert Braden, and David Borman. Tcp extensions for high performance. Technical report, 1992.

[35] Nicolas Kagami, Roberto Irajá Tavares da Costa Filho, and Luciano Paschoal Gaspary. Capest: Offloading network capacity and available bandwidth estimation to programmable data planes. *IEEE Transactions on Network and Service Management*, 2019.

[36] Lampros Kalampoukas, Anujan Varma, and KK Ramakrishnan. Explicit Window Adaptation: a Method to Enhance TCP Performance. 1:242–251, 1998.

[37] Rohit Kapoor, Ling-Jyh Chen, Li Lao, Mario Gerla, and M Young Sanadidi. Capprobe: A simple and accurate capacity estimation technique. *ACM SIGCOMM Computer Communication Review*, 34(4):67–78, 2004.

[38] Sachin Katti, Dina Katabi, Charles Blake, Eddie Kohler, and Jacob Strauss. Multiq: Automated detection of multiple bottleneck capacities along a path. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 245–250, 2004.

[39] Sukhpreet Kaur Khangura and Sami Akın. Measurement-based online available bandwidth estimation employing reinforcement learning. In *2019 31st International Teletraffic Congress (ITC 31)*, pages 95–103. IEEE, 2019.

[40] Sukhpreet Kaur Khangura and Markus Fidler. Available bandwidth estimation from passive tcp measurements using the probe gap model. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2017.

[41] Sukhpreet Kaur Khangura, Markus Fidler, and Bodo Rosenhahn. Neural networks for measurement-based bandwidth estimation. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2018.

[42] Sukhpreet Kaur Khangura, Markus Fidler, and Bodo Rosenhahn. Machine learning for measurement-based bandwidth estimation. *Computer Communications*, 144:18–30, 2019.

[43] Jin Cheol Kim and Younghee Lee. An end-to-end measurement and monitoring technique for the bottleneck link capacity and its available bandwidth. *Computer Networks*, 58:158–179, 2014.

[44] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications.

[45] Mingzhe Li, Mark Claypool, and Robert Kinicki. Packet dispersion in ieee 802.11 wireless networks. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 721–729. IEEE, 2006.

[46] Shiyao Ma, Jingjie Jiang, Wei Wang, and Bo Li. Fairness of congestion-based congestion control: Experimental evaluation and analysis. *arXiv preprint arXiv:1706.09115*, 2017.

[47] Matthew Mathis and Jamshid Mahdavi. Forward acknowledgement: Refining tcp congestion control. *ACM SIGCOMM Computer Communication Review*, 26(4):281–291, 1996.

[48] Bob Melander, Mats Bjorkman, and Per Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Globecom'00-IEEE. Global Telecommunications Conference. Conference Record (Cat. No. 00CH37137)*. IEEE.

[49] Foivos Michelinakis, Nicola Bui, Guido Fioravantti, Joerg Widmer, Fabian Kaup, and David Hausheer. Lightweight capacity measurements for mobile networks. *Computer Communications*, 84:73–83, 2016.

[50] David Murray, Terry Koziniec, Sebastian Zander, Michael Dixon, and Polychronis Koutsakis. An analysis of changing enterprise network traffic characteristics. In *2017 23rd Asia-Pacific Conference on Communications (APCC)*, pages 1–6. IEEE, 2017.

[51] Szilveszter Nádas, Gergő Gombos, Péter Hudoba, and Sándor Laki. Towards a congestion control-independent core-stateless aqm. In *Proceedings of the Applied Networking Research Workshop*, pages 84–90, 2018.

[52] Mario Nemirovsky, René Serral-Gracià, Francesco Ciaccia, and Ivan Romero Ruiz. Intelligent adaptive transport layer to enhance performance using multiple channels, December 21 2017. US Patent App. 15/626,130.

[53] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[54] Farzaneh Pakzad, Marius Portmann, and Jared Hayward. Link capacity estimation in wireless software defined networks. In *2015 International Telecommunication Networks and Applications Conference (ITNAC)*, pages 208–213. IEEE, 2015.

[55] Anup Kumar Paul, Atsuo Tachibana, and Teruyuki Hasegawa. An enhanced available bandwidth estimation technique for an end-to-end network path. *IEEE Transactions on Network and Service Management*, 13(4):768–781, 2016.

[56] Ravi Prasad, Manish Jain, and Constantinos Dovrolis. Effects of interrupt coalescence on network measurements. In *International Workshop on Passive and Active Network Measurement*, pages 247–256. Springer, 2004.

[57] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, RFC Editor, September 2001. URL https://tools.ietf.org/html/rfc3168.

[58] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop*, 2003.

[59] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, 2010.

[60] Ivan Romero, Francesco Ciaccia, René Serral-Gracià, and Mario Nemirovsky. Automatic communication network control, May 11 2020. US Patent App. 20/32,386.

[61] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 404–417, 2017.

[62] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[63] Jacob Strauss, Dina Katabi, Frans Kaashoek, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*.

[64] Tim Szigeti, Christina Hattingh, Robert Barton, and Kenneth Briley Jr. *End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks*. Cisco press, 2013.

[65] Tarik Taleb, Nei Kato, and Yoshiaki Nemoto. An explicit and fair window adjustment method to enhance TCP efficiency and fairness over multihops satellite networks. *IEEE Journal on Selected Areas in Communications*, 22(2): 371–387, 2004.

[66] VMWare Wavefront. Using moving and tumbling windows to highlight trends. URL https://docs.wavefront.com/query_language_windows_trends.html.

[67] Felix Ming Fai Wong, Carlee Joe-Wong, Sangtae Ha, Zhenming Liu, and Mung Chiang. Improving user QoE for residential broadband: Adaptive traffic management at the network edge. In *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pages 105–114. IEEE, 2015.

[68] N. Cardwell Y. Cheng. Rack: a time-based fast loss detection algorithm for tcp. RFC draft-cheng-tcpm-rack-00, RFC Editor, October 2015. URL https://tools.ietf.org/html/draft-cheng-tcpm-rack-00.

[69] Qianwen Yin and Jasleen Kaur. Can machine learning benefit bandwidth estimation at ultra-high speeds? In *International Conference on Passive and Active Network Measurement*, pages 397–411. Springer, 2016.

Appendix A

TCP Internals

TCP is a stateful connection that automatically retransmits lost or damaged packets, while adapting to the conditions of the endpoints (Flow Control) and the network (Congestion Control). A TCP connection is initiated exchanging three packets (three-way handshake), and finishes similarly (although it can remain half-closed if only one of the endpoints closes its connection). The state machine is shown in Figure A.1.

TCP flow control and window size adjustment is mainly based on two key mechanism: Slow Start and Additive Increase/Multiplicative Decrease (AIMD), also known as Congestion Avoidance (RFC 793 and RFC 5681). Below we describe these mechanisms and how they are related.

A.1 Flow Control

Flow Control basically means that TCP will ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume. It's pretty similar to what's normally called Back pressure in the Distributed Systems literature. The idea is that a node receiving data will send some kind of feedback to the node sending the data to let it know about its current condition. To control the amount of data that TCP can send, the receiver will advertise its Receive Window (rwnd), that is, the spare room in the receive buffer. Every time TCP receives a packet, it needs to send an ack message to the sender, acknowledging it received that packet correctly, and with this

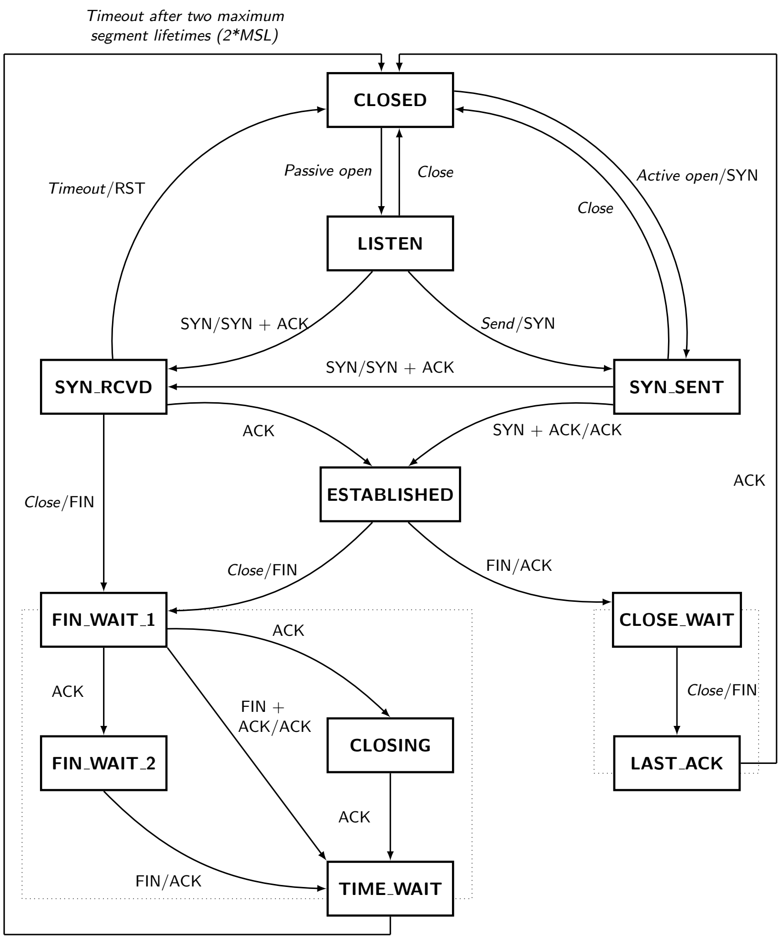


Fig. A.1 TCP State Machine in Linux.

ack message it sends the value of the current receive window, so the sender knows if it can keep sending data. TCP uses a sliding window protocol to control the number of bytes in flight it can have. In other words, the number of bytes that were sent but not yet acked. Let's say we want to send a 150000 bytes file from node A to node B. TCP could break this file down into 100 packets, 1500 bytes each. Now let's say that when the connection between node A and B is established, node B advertises a receive window of 45000 bytes. Seeing that, TCP knows it can send the first 30 packets ($1500 \times 30 = 45000$) before it receives an acknowledgment. If it gets an ack message for the first 10 packets (meaning we now have only 20 packets in flight), and the receive window present in these ack messages is still 45000, it can send the next 10 packets, bringing the number of packets in flight back to 30, that is the limit defined by the receive window. In other words, at any given point in time it can have 30 packets in flight, that were sent but not yet acked. Now, if for some reason the application reading these packets in node B slows down, TCP will still ack the packets that were correctly received, but as these packets need to be stored in the receive buffer until the application decides to read them, the receive window will be smaller, so even if TCP receives the acknowledgment for the next 10 packets (meaning there are currently 20 packets, or 30000 bytes, in flight), but the receive window value received in this ack is now 30000 (instead of 45000), it will not send more packets, as the number of bytes in flight is already equal to the latest receive window advertised. The sender will always keep this invariant:

$$LastByteSent - LastByteAcked \leq ReceiveWindowAdvertised \tag{A.1}$$

A.2 Zero Windows and the Persist Timer

In the case a receiver advertises a zero window, TCP will stop transmitting data as the receiver's buffer is full. However, After the receiver advertises a zero window, if it doesn't send any other ack message to the sender (or if the ack is lost), it will never know when it can start sending data again. This would lead to have a deadlock situation, where the receiver is waiting for more data, and the sender is waiting for a message saying it can start sending data again. To solve this problem, when TCP receives a zero-window message it starts the persist timer, that will periodically send a small packet to the receiver (usually called WindowProbe), so it has a chance to advertise a nonzero window size.

A.3 Slow Start

To avoid that a starting TCP connection floods the network, a Slow Start mechanism was introduced in TCP. This mechanism effectively probes to find the available bandwidth. In addition to the window advertised by the receiver, a Congestion Window (cwnd) value is used and the effective window size is the lesser of the two. The starting value of the cwnd window is set initially to a value that has been evolving over the years, the TCP Initial Window. After each acknowledgment, the cwnd window is increased by one MSS. By this algorithm, the data rate of the sender doubles each round-trip time (RTT) interval (actually, taking into account Delayed ACKs, rate increases by 50% every RTT). For a properly implemented version of TCP this increase continues until:

- the advertised window size is reached,
- congestion (packet loss) is detected on the connection,
- there is no traffic waiting to take advantage of an increased window (i.e. cwnd should only grow if it needs to).

When congestion is detected, the TCP flow-control mode is changed from Slow Start to Congestion Avoidance. Note that some TCP implementations maintain cwnd in units of bytes, while others use units of full-sized segments.

A.4 Congestion Avoidance

Once congestion is detected (through timeout and/or duplicate ACKs), the data rate is reduced in order to let the network recover. Slow Start uses an exponential increase in window size and thus also in data rate. Congestion Avoidance uses a linear growth function (additive increase). This is achieved by introducing - in addition to the cwnd window - a slow start threshold (ssthresh). As long as cwnd is less than ssthresh, Slow Start applies. Once ssthresh is reached, cwnd is increased by at most one segment per RTT. The cwnd window continues to open with this linear rate until a congestion event is detected. When congestion is detected, ssthresh is set to half the cwnd (or to be strictly accurate, half the "Flight Size". This distinction is important if the implementation lets cwnd grow beyond rwnd (the receiver's declared window)). cwnd is either set to 1 if congestion was signalled by a timeout, forcing the sender to enter Slow Start, or to ssthresh if congestion was signalled by duplicate ACKs and the Fast Recovery algorithm has terminated. In either case, once the sender enters Congestion Avoidance, its rate has been reduced to half the value at the time of congestion. This multiplicative decrease causes the cwnd to close exponentially with each detected loss event.

A.5 Fast Retransmit

In Fast Retransmit, the arrival of three duplicate ACKs is interpreted as packet loss, and retransmission starts before the retransmission timer (RTO) expires. The missing segment will be retransmitted immediately without going through the normal retransmission queue processing. This improves performance by eliminating delays that would suspend effective data flow on the link.

Fast Retransmit (1990) was superseded by Forward Acknowledgement [47], which is being replaced by Recent Acknowledgement (RACK) [68] for algorithms like BBR: “RACK uses the notion of time, instead of packet or sequence counts, to detect losses, for modern TCP implementations that can support per-packet timestamps and the selective acknowledgment (SACK) option. It is intended to replace the conventional DUPACK threshold approach and its variants, as well as other nonstandard approaches.”.

A.6 Fast Recovery

Fast Recovery is used to react quickly to a single packet loss. In Fast recovery, the receipt of 3 duplicate ACKs, while being taken to mean a loss of a segment, does not result in a full Slow Start. This is because obviously later segments got through, and hence congestion is not stopping everything. In fast recovery, ssthresh is set to half of the current send window size, the missing segment is retransmitted (Fast Retransmit) and cwnd is set to ssthresh plus three segments. Each additional duplicate ACK indicates that one segment has left the network at the receiver and cwnd is increased by one segment to allow the transmission of another segment if allowed by the new cwnd. When an ACK is received for new data, cwnd is reset to the ssthresh, and TCP enters congestion avoidance mode.