Understanding and Mitigating Congestion in Modern Networks

Yin Xu
B.Sc. Fudan University

A THESIS SUBMITTED

FOR THE DEGREE OF PH.D. IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2014

Acknowledgement

I want to express my first and foremost gratitude to my supervisor, Prof. Ben Leong. In these years, I have learned a lot from his patient and kind guidance in research and also in life. I am grateful for his infinite patience and being extremely supportive. He is like a beacon guiding my way to success and a bright future.

I am also grateful to my friends and collaborators: Wei Wang, Ali Razeen, Wai Kay Leong, Daryl Seah, Jian Gong, Guoqing Yu, Jiajun Tan, Andrew Eng, and Zixiao Wang. Without their assistance, I would not have finished all the work on time. Every hour spent with them is a lifetime of wealth.

I would also like to acknowledge my parents for their selfless and noblest love. It is at my mother and father's knee that I acquire the noblest, truest and highest dream. Thanks to them in helping me walk towards my dream.

Special mention goes to my wife, Yanping Chen, the most beautiful and wonderful woman in the world. Her smile and encouragement provide me incessant power to overcome all the tough problems during my research. She is the *special one* that always stand behind my success. She is the *only woman* that hold up half my sky. She provides me the most peaceful harbor, *the home*.

Finally, special thanks to my baby, Xinchen Xu. He is such a cute boy and I love him so much. He is the new shining star of my life. Thank you for coming into my life when I was experiencing a bottleneck in my research. It is his smile and steps that motivate me to carry on.

Publications

- Yin Xu, Zixiao Wang, Wai Kay Leong, and Ben Leong. "An End-to-End Measurement Study of Modern Cellular Data Networks." In *Proceedings of the 15th Passive and Active Measurement Conference (PAM 2014)*. Mar. 2014.

- Wai Kay Leong, Aditya Kulkarni, Yin Xu and Ben Leong. "Unveiling the Hidden Dangers of Public IP Addresses in 4G/LTE Cellular Data Networks." In *Proceedings of the 15th International Workshop on Mobile Computing Systems and Applications (HotMobile 2014)*. Feb. 2014.

- Wai Kay Leong, Yin Xu, Ben Leong and Zixiao Wang. "Mitigating Egregious ACK Delays in Cellular Data Networks by Eliminating TCP ACK Clocking." In *Proceedings of the 21st IEEE International Conference on Network Protocols*, Oct. 2013.

- Yin Xu, Ben Leong, Daryl Seah and Ali Razeen. "mPath: High-Bandwidth Data Transfers with Massively-Multipath Source Routing." IEEE Transactions on Parallel and Distributed Systems, Volume 24, Issue 10, pp 2046-2059, Oct. 2013.

- Yin Xu, Wai Kay Leong, Ben Leong, and Ali Razeen. "Dynamic Regulation of Mobile 3G/HSPA Uplink Buffer with Receiver-side Flow Control." In *Proceedings of the 20th IEEE International Conference on Network Protocols*, Oct. 2012.

Abstract

To design an efficient transmission protocol and achieve good performance, it is essential to understand and address the issue of network congestion. With modern networks, we now not only have new opportunities, but also have more challenges. In this thesis, we investigate network congestion issues in the context of modern wired Internet and cellular data networks.

In the wired Internet, the capacity of access links has increased dramatically in recent times [47]. As a result, the bottlenecks are moving deeper into the Internet core. When a bottleneck occurs in a core (or AS-AS peering) link, it is often possible to use additional detour paths to improve the end-to-end throughput between a pair of source and destination nodes. We propose and evaluate a new *massively-multipath (mPath) source routing* algorithm to improve end-to-end throughput for high-volume data transfers. We demonstrate that our algorithm is practical by implementing a system that employs a set of proxies to establish one-hop detour paths between the source and destination nodes. Our algorithm can fully utilize the available access link bandwidth when good proxied paths are available, without sacrificing TCP-friendliness. It can also achieve throughput comparable to TCP when such paths cannot be found. For 40% of our test cases on PlanetLab, mPath achieves significant improvements in throughput. Among these, 50% achieves a throughput of more than twice that of TCP.

While the congestion in wired Internet is relative well studied, there are still gaps in our understanding of congestion in cellular data networks. We believe that it is critical to better understand the characteristics and behavior of cellular data networks, as there has been a significant increase in cellular data usage [1]. With both laboratory experiments and crowd-sourcing measurements, we investigate the characteristics of the cellular data networks for the three mobile ISPs in Singapore. We found that i) the transmitted packets tend to arrive in bursts; ii) there can be large variations in the instantaneous throughput over a short period of time; iii) large separate downlink buffers are typically deployed, which can cause high latency at low speeds; and iv) the networks typically implement some form of fair queuing policy for all the connected devices. Our findings confirm that cellular data networks behave differently from conventional wired and WiFi networks, and

our results suggest that more can be done to optimize protocol performance in existing cellular data networks. We then measure and investigate the "self-inflicted" congestion problem caused by a saturated uplink in cellular data networks. We found that the performance of downloads in cellular data networks can be significantly degraded by a concurrent upload that saturates the uplink buffer on the mobile device. In particular, it is common for the download speeds to be reduced by over an order of magnitude from 2,000 Kbps to 100 Kbps.

To mitigate the uplink saturation problem, we propose a new algorithm called *Receiver-side Flow Control* (RSFC) that regulates the uplink buffer on the data senders of the cellular data networks. RSFC uses a feedback loop to monitor the available uplink capacity and dynamically adjusts the TCP receiver window (`rwnd`) accordingly. We evaluate RSFC on the cellular data networks of three different mobile ISPs and show that RSFC can improve the download throughput from less than 400 Kbps to up to 1,400 Kbps. RSFC can also reduce website load times from more than 2 minutes to less than 1 minute some 90% of the time in the presence of a concurrent upload. Our technique is compatible with existing TCP implementations and can be easily deployed at the mobile proxies without requiring any modification to existing mobile devices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the Internet has evolved rapidly in recent years, conventional wisdom and assumptions may not hold any more. In particular, we identified two major trends for modern networks: i) the access link capacity of the wired Internet is increasing dramatically [47]; ii) an increasing amount of Internet traffic is carried by the cellular data networks [1]. In this thesis, we investigate the congestion problems for these two scenarios and propose methods to mitigate the problems we identified.

For the wired Internet, bottlenecks have been observed to be shifting away from the network edges and happen at the core link due to the growing capacity of access links [8]. As the last mile bandwidth is set to increase dramatically over the next few years [47], we expect that this trend will accelerate. In this sense, we design and implement a new *massively-multipath (mPath) source routing* mechanism to improve the utilization of the available last mile bandwidth when there is core link congestion [98].

For cellular data networks, the characteristics and behavior are not well studied and the performance of the current transmission protocols is also far from

satisfactory. Hence, we first conduct a measurement study to understand the characteristics and behavior of the cellular link [100]. In particular, we identify a "self-inflicted" congestion problem caused by the saturated uplink. Then, we propose a *Receiver-side Flow Control (RSFC)* algorithm to solve the problem [99].

1.1 Addressing Congestion in the Wired Internet

Research has shown that there are often less-congested paths than the direct one between two end-hosts over the Internet [81, 44]. These alternative paths through the Internet core were initially not exploitable as bandwidth bottlenecks used to be in the "last mile". As last mile bandwidth is set to increase dramatically over the next few years [47], we expect that increasingly the available bandwidth will be constrained by core link bottlenecks. We now have the opportunity to exploit path diversity and use multiple paths concurrently to fully saturate the available access link bandwidth for high-volume data transfers, e.g. scientific applications [60] or inter-datacenter bulk transfers [64].

While the idea of multipath routing is not new, previously proposed systems either require multi-homing support [97] or the maintenance of an overlay network with only a small number of paths [104]. Our approach is to use a large set of geographically-distributed proxies to construct and utilize up to hundreds of detour paths [81] between two arbitrary end-hosts. By adopting one-hop source routing [38] and designing the proxies to be stateless, we also require significantly less coordination and control than previous systems [104, 11] and ensure that our system would be resilient to proxy failures. Our system, which we call *mPath* (or massively-multipath source routing), is illustrated in Figure 1.1.

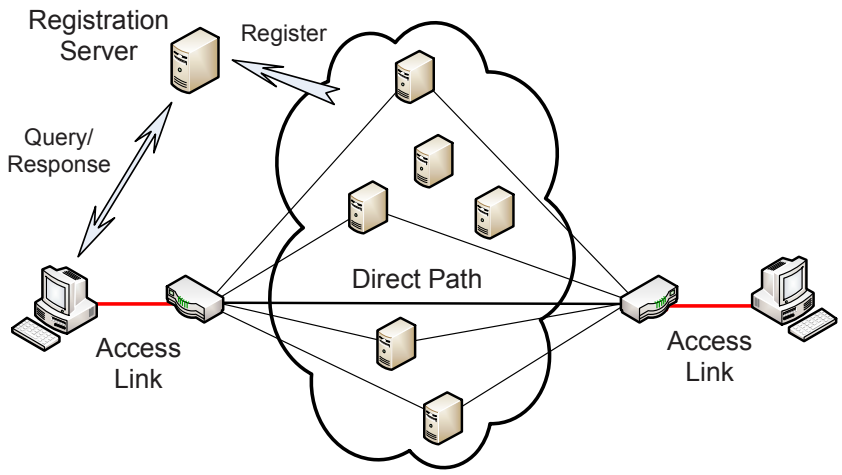


Figure 1.1: Massively-multipath source routing.

There are a number of challenges in designing such a system: (i) good alternative paths may not always exist, and in such cases the performance should be no worse than a direct TCP connection; (ii) when good alternative paths do exist, we need to be able to efficiently identify them and to determine the proportion of traffic to send on each path; and (iii) Internet traffic patterns are dynamic and unpredictable, so we need to adapt to changing path conditions rapidly.

Our key contribution, which addresses these design challenges, is a combined congestion control and path selection algorithm that can identify bottlenecks, apportion traffic appropriately, and inter-operate with existing TCP flows in a TCP-friendly manner. The algorithm is a variant of the classic additive increase/multiplicative decrease (AIMD) algorithm [26] that infers shared bottlenecks from correlated packet losses and uses an operation called *load aggregation* to maximize the utilization of the direct path. The design goal of our algorithm is to supplement the direct path by exploiting the proxied paths when the congestion happens in the core link, without sacrificing the utilization of the direct path.

We model and analyze the performance of mPath to show that our algorithm (i) is TCP-friendly, (ii) will maximize the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) will rapidly eliminate any redundant proxied paths.

We validated our model with experiments on Emulab and evaluated our system on PlanetLab with a set of 450 proxies to show that our algorithm is practical and achieves significant improvements in throughput over TCP for some 40% of the end-hosts. Among these, half of them achieves more than twice the throughput of TCP. In addition, when good proxied paths cannot be found or the bottleneck is at a common access link, mPath achieves throughput that is comparable to TCP and stabilizes in approximately the same time.

1.2 Characteristics of Cellular Data Networks

Cellular data networks are carrying an increasing amount of traffic with their ubiquitous deployments and have significantly improved in speed in recent years [1]. However, networks such as HSPA and LTE have very different link-layer protocols from wired and WiFi networks. It is thus important to have a better understanding of the characteristics and behavior of cellular data networks.

In this thesis, we investigate and measure the characteristics of the cellular data networks for the three ISPs in Singapore with experiments in the laboratory as well as with crowd-sourced data from real mobile subscribers. The latter was obtained using our custom Android application that was used by real users over a 5-month period from April to August 2013. From our results, we make the following common observations on the existing cellular data networks: i) transmitted

packets tend to arrive in bursts; ii) there can be large variations in the instantaneous throughput over a short period of time, even when the mobile device is stationary; iii) large separate downlink buffers are typically deployed in mobile ISPs, which can cause high latency at low speeds; and iv) mobile ISPs typically implement some form of fair queuing policy for all the connected devices.

Our findings confirm that cellular data networks behave differently from conventional wired and WiFi networks, and our results suggest that more can be done to optimize protocol performance in existing cellular data networks. For example, the fair scheduling in such networks might effectively eliminate the need for congestion control if the cellular link is the bottleneck link. We have also found that different ISPs and even different devices use different buffer configurations and queuing policies. Whether these configurations are optimal and what makes a configuration optimal are candidates for further study.

We further investigate the performance issues when there are concurrent uploads and downloads in cellular data networks. This problem has attracted much attentions recently because the increasing popularity of mobile devices and online social networks has caused simultaneous uploads and downloads to become commonplace in cellular data networks. For example, the fans at a recent sports event uploaded 40% more data (such as photos and videos) than they downloaded [15]. It would therefore be not surprising to find users attempting to access websites while photos and video are being uploaded in the background. Our measurement study shows that in the presence of a simultaneous background upload, 3G download speeds can be drastically reduced from more than 1,000 Kbps to less than 100 Kbps. With 3G poised to become even more ubiquitous [1], there is an urgent need to understand and address this "self-inflicted" congestion problem.

Since upload speeds in cellular data networks are typically lower compared to the download speeds, the downstream ACKs will be queued behind the data packets in the uplink buffer when there are concurrent flows in both directions. The ACKs can sometimes be severely delayed and cause the download speeds to slow to a crawl. We confirm with experiments that the ACK delay is the main cause of downlink under-utilization under such circumstances.

1.3 Addressing Self-inflicted Congestion in Cellular Data Networks

While one might be tempted to think that the uplink saturation problem is a manifestation of the well-known ACK compression problem [103], Heusse et al. recently demonstrated that ACK compression rarely occurs in practice and even if it does, it has little effect on performance [42]. Instead, they showed that the degradation in performance is a result of the uplink buffer not being appropriately sized for the available link capacity.

TCP buffer sizing is also a well-studied problem. There is an old rule of thumb that the size of a general buffer should be set to the bandwidth-delay product [93] (BDP). More recently, it was found that it should be set to $BDP/\sqrt{n}$, where n is the number of long lived flows [13]. Unfortunately, these rules cannot be applied to cellular data networks directly because they exhibit significant spatial and temporal variation. For example, we have observed that the available uplink bandwidth can vary by as much as two orders of magnitude within a 10-minute interval. Therefore, to fully utilize the available uplink capacity, we cannot use a

fixed buffer size at the cellular interface of the mobile devices. Instead, the size of the uplink buffer needs to be dynamically adjusted according to the available bandwidth.

In this thesis, we describe *Receiver-side Flow Control* (or RSFC), a method to dynamically control the uplink buffer of the sender from the receiver. The technique of using `rwnd` to control a TCP flow has been employed in other contexts [35, 87, 57, 54, 12, 24, 51]. However, to the best of our knowledge, we are the first to apply this technique to improve the utilization of a 3G mobile downlink in the presence of concurrent uploads.

The key challenge is for the TCP receiver to accurately estimate the current uplink capacity and to determine the appropriate `rwnd` to be advertised so that the number of packets in the uplink buffer is kept small without causing the uplink to become under-utilized. To solve this challenge, our approach uses the TCP timestamp to continuously estimate the one-way delay, queuing delay and RTT. Then, our approach uses a feedback loop to continuously estimate the available uplink bandwidth and advertises an appropriate TCP receiver window (`rwnd`) according to the current congestion state. This approach can dynamically adapt to the variations of the cellular link.

We evaluated RSFC extensively on three mobile ISPs using Android phones and show that RSFC can significantly improve downlink utilization, especially in an ISP with consistently low upload speeds. In our experiments, we found that RSFC can improve download speeds from lower than 400 Kbps to up to 1400 Kbps and reduce the time taken to load websites in the presence of concurrent uploads from more than 2 minutes to less than 1 minute some 90% of the time. We also showed that RSFC is compatible with existing TCP implementations.

1.4 Contributions

The key contributions of this thesis are two practical network protocols that can be deployed immediately, mPath and RSFC. mPath is designed to mitigate core link congestion problem in the wired Internet. RSFC is designed to mitigate "self-inflicted" congestion problem in cellular data networks.

mPath is a new multipath source routing algorithm that uses multiple detour paths concurrently to route around the core link congestion and better utilize the access link. Our studies corroborate the fact that the congestion of the wired Internet can happen in the Internet core quite often and show that detour paths are useful to route around the core link congestion. The key mechanism is a combined congestion control and path selection algorithm that can identify bottlenecks, apportion traffic appropriately, and inter-operate with existing TCP flows in a TCP-friendly manner. The major contributions and insights include: i) using the actual data to probe the path conditions; ii) decoupling the congestion detection and the sequence reordering by using two sequence numbers, a *stream sequence number* and a *path sequence number*; 3) inferring shared bottlenecks from correlated packet losses and using an operation called *load aggregation* to maximize the utilization of the direct path.

RSFC is a new flow control algorithm that only requires modifications at the receiver side of the upstream and solves the "self-inflicted" congestion problem caused by a saturated uplink buffer. Our studies show that the saturated uplink buffer can degrade the downlink performance significantly. Our key approach and contribution to solve this problem is a feedback loop that can dynamically adapt to the variations of the cellular link, by i) using the TCP timestamp to continuously

estimate the one-way delay, queuing delay and RTT; ii) inferring whether the link is congested using the queuing delay instead of the packet loss; iii) continuously estimating the available uplink bandwidth and advertising an appropriate TCP receiver window (`rwnd`) to regulate the send rate of the upstream flow.

1.5 Organization of this Thesis

The rest of this thesis is organized as follows: in Chapter 2, we provide an overview of the related work. In Chapter 3, we discuss the massively-multipath source routing system designed for the wired Internet. In Chapter 4, we show our measurement studies about the characteristics and behavior of cellular data networks and in particular, the "self-inflicted" congestion problem caused by the saturated uplink. In Chapter 5, we describe a Receiver-side Flow Control (RSFC) to solve the uplink saturation problem. In Chapter 6, we summarize this work and discuss some of the open issues and future research directions.

Chapter 2

Related Work

In this chapter, we provide an overview of the existing literature related to this thesis. We first discuss the work related to mPath in Section 2.1. Then, we discuss some of the interesting findings of previous measurement studies in Section 2.2. Finally, we discuss the previous work related to RSFC in Section 2.3.

2.1 Massively-Multipath Source Routing

In this section, we describe the prior work in the literature related to mPath. We first discuss the Internet bottlenecks and the path diversity. Then, we proceed to describe the previous solutions, including detour routing [81, 44, 11, 104, 38], multipath TCP [97], parallel TCP [85, 40] and split TCP [67, 49, 16]. Finally, we conclude with a discussion of three major components associated with multi-path algorithms, namely *path selection*, *congestion control* and *shared bottleneck detection*.

2.1.1 Internet Bottlenecks

Internet bottlenecks were commonly thought to occur at the access links. While this was true years ago when the access link capacity was small, this thought no longer holds true today. Akella et al. were the first to dispute this assumption, by highlighting that nearly half of the Internet paths they investigated had a non-access link bottleneck with an available capacity of less than 50 Mbps [8]. Hu et al. suggested that bottlenecks could exist everywhere, at access links, peering links or even inside Autonomous System (AS) with measurement studies using *Pathneck* [44]. For example, they found that up to 40% of the bottlenecks are located within an AS. Our current experience with mPath seems to corroborate these findings. In addition, the last mile bandwidth is set to increase dramatically over the next few years as the deployment of the fiber to the home [47], we expect that this trend will accelerate and the end-to-end data transfers will be further constrained by the core link bottlenecks.

The reason why the Internet bottlenecks happen at the core links is that the BGP routing algorithm only selects one routing path, and the path is susceptible to "hot potato" routing as the ISPs may attempt to maximize their own profit. "Hot potato" routing has been shown to degrade the end-to-end performance significantly [73] and cause delay Internet routing convergence problem [62]. The sub-optimality of the direct routing path has also been proved by many researchers who showed that there often exists better detour paths that are less congested than the direct path between two end-hosts over the Internet [8, 44]. With all these less congested detour paths, it is possible for us to use multiple paths concurrently in order to fully utilize the available access link bandwidth for high-volume data

transfers, e.g. scientific applications [60] or inter-datacenter bulk transfers [64]. In this thesis, we investigate the possibility of using multipath and propose a new massively-multipath sourcing routing method to exploit the path diversity to better utilize the access link.

2.1.2 Detour Routing

The benefits of detour routing have been demonstrated by many researchers [81, 44, 105]. Savage et al. had earlier shown that some 30% to 80% of paths could be improved by detour routing [81]. Hu et al. also found that 52.72% of overlay attempts were useful out of 63,440 attempts [44]. Zheng et al. further investigated the *triangle inequality violations* (TIVs) phenomenon of the Internet routing which suggested that it could be beneficial to relay the packets with some intermediate nodes [105]. Many prior systems exploited this fact and tried to improve the end-to-end performance by using one or few paths, including RON [11], mTCP [104], Skype [59], ASAP [78] and one-hop source routing [38].

Anderson et al. built a resilient overlay network (RON) [11] based on detour routing and showed that the system could recover from most of the outages and path failures. RON enabled a group of nodes to communicate with each other so that a better detour node could be selected when the original path failed. A better detour node could help to route around most of the failures so that the recovery would be faster. RON also tried to integrate the path selection with distributed applications more tightly, so that the applications could select a path with best quality using the most crucial metric, e.g. delay or throughput. They also showed RON can decrease the delay and loss rate, and improve the throughput for some

data transfers. An interesting rule they discovered was that using one intermediate node is enough to find good detour paths, which was also verified by another work [38] and hence followed by mPath.

mTCP was built upon RON and was the first system that attempted to improve the throughput by using multiple paths simultaneously [104]. The authors claimed that it is inefficient to use conventional TCP congestion control mechanism when the system operates over multiple paths. Instead, mTCP performed congestion control for each subflow to minimize the negative influence of the poorer paths. However, we found that such method would be too aggressive when the system uses tens or hundreds of paths concurrently. The authors were also aware of this problem and proposed a shared congestion detection mechanism to identify and suppress subflows that traversed the same set of congested links. We found that their method was not sufficiently adaptive and we proposed a mechanism that is able to react to the shared congestion better. mTCP also used a naive path selection method in that it selected the disjoint paths using *traceroute*. They claimed it was impractical to use all the paths simultaneously. However, we found that it is possible to dynamically add and remove the paths until all hundreds of paths are used during transmission with little overhead and within an acceptable interval. In addition, the detour paths go through the same bottleneck could actually be used simultaneously for the purpose of load balance.

The most serious drawback of these two works is the scalability. RON incurs a lot of communication overhead, and is hence not scalable to large networks. mTCP is built upon RON and uses *traceroute* to find disjoint paths. Hence, it is impossible to employ tens or hundreds of paths simultaneously in mTCP. mPath differs from these systems in that it aims to maximize throughput by using hun-

dreds of light-weight proxies in a source routing manner instead of depending on an overlay network.

Skype [59] and ASAP [78] also used overlay networks to reduce the latency for VoIP applications. Skype was the most well known commercial application that employed the relay nodes to improve the VoIP quality. The relay nodes in Skype were used for two purposes: searching clients and relaying voice packets. Ren et al. found three major issues of the Skype system: i) many relay peer selections are sub-optimal; ii) the waiting time to select a relay node could be quite long; and iii) there are many unnecessary probes in Skype, which reduce the scalability. They then proposed a way to use information of the AS topology to select the relay nodes. Our work differs with these two that we focus on the throughput instead of the delays.

Gummadi et al. were the first to propose one-hop source routing to address RON's scalability issues [38]. They developed a random-k path selection algorithm that the sender selected one or more intermediaries and attempted to reroute the packets through them after it detected a path failure. If one of the random relay node could bypass the failure point, the communication would be restored immediately. The major challenge was how to select a good detour path. By comparing the history-k (select the best k paths from previous transfers) and BGP-paths-k (select the most disjoint k paths with the BGP routing information), they found a random-k algorithm was already sufficient with least overhead. In their environment, $k = 4$ resulted in the best trade-off between accuracy and overhead. Our work differs with theirs that we attempts to improve throughput by using multiple paths simultaneously.

2.1.3 Multi-homing and Multipath TCP

Another common mechanism that can provide path diversity is multi-homing [7], but it needs to be supported by the ISPs at the network-layer. Multipath TCP (MPTCP) [97] was developed to support multipath TCP over multi-homing and had also been proposed for use in intra-datacenter bulk transfers [77].

The design of MPTCP is similar with mPath in many factors. Like mPath, MPTCP also uses two levels of sequence number, *connection-level sequence number* and *subflow-level sequence number*. The connection-level sequence number can be used to order and reassemble the packets, similar as the function of stream sequence number in mPath. The subflow-level sequence number can be used to control the congestion and detect the losses, similar as the function of path sequence number in mPath. In this way, MPTCP is also able to retransmit the same part of connection-level sequence space on different subflow-level sequence number. MPTCP is also similar with mPath in the congestion control algorithm that both use a coupled algorithm and preserve the TCP-friendliness with the awareness of the shared bottleneck problem.

The major difference between MPTCP and mPath is how they manage the paths. MPTCP requires support of multi-homing and the number of paths is small. mPath can exploit, but does not require, multi-homing. The potential number of paths used by mPath can be much larger. Hence, in MPTCP, it seeks only to allocate traffic optimally over a fixed (and small) set of available paths, while in mPath, it needs to solve two separate problems simultaneously: (i) identify good proxied paths out of several hundred paths; and (ii) allocate the optimal amount of traffic to the good proxied paths. Also, MPTCP and mPath take different ap-

proaches to distribute the data. MPTCP tries to balance the congestion among the paths by considering the overhead of each path. However, mPath should maximize the usage of direct path because it involves the least resource requirement. The proxied paths will only be used when they can help to route around the bottleneck of the direct path. Another implementation difference is that MPTCP is a direct extension of current TCP and it utilizes TCP option to include the additional information, while mPath is an application-layer protocol works above UDP.

2.1.4 Parallel TCP and Split TCP

mPath also differs from Parallel TCP [85, 40] and Split TCP [67, 49, 16]. Parallel TCP was proposed to increase throughput by exploiting multiple TCP flows at the expense of TCP-friendliness. In mPath, we strictly adhere to the AIMD mechanism to maintain TCP-friendliness. Split TCP increases throughput by exploiting the pipeline parallelism of multiple low-latency segments, which requires buffering of data at the proxies and breaks end-to-end guarantees. mPath does not use this mechanism because we intend to maintain the end-to-end guarantees and keep the proxies light-weight and stateless without buffering. mPath differs with these two works that mPath improves the throughput by simply routing around core link bottlenecks.

2.1.5 Path Selection

Path selection, that is finding an optimal detour path set, is one of the most crucial component of a multipath system. One category of the mechanism is to discover and select the detour paths by active probing [104, 11, 59], which would

often decrease the scalability of the system. RON assessed the path quality of the communication and evaluation between nodes [11]. mTCP was built upon RON and discovered the disjoint paths with *traceroute* [104]. Fei et al. introduced a heuristic, the AS-level earliest-divergence rule, that achieved a reasonable trade-off between the accuracy and overhead [31]. In this rule, they claimed that if the AS path to an intermediate node diverges early from the direct path, the AS path from that nodes tends to merge back into the direct path relative late, hence the detour path through that node is more disjoint with default path. With this rule, they reduced the probing overhead from $O(N^2)$ to $O(N)$ in the p2p environment. Even in commercial software like Skype [59], the scalability is reduced because of the unnecessary probes [78].

To improve the scalability, Gummadi et al. suggested that a random-k path selection method was sufficient in their one-hop source routing system, after comparing with the history-k and BGP-paths-k mechanism [38]. While random-k is much more scalable and incurs little overhead, it is not accurate all the time. In mPath, we also select the proxied paths randomly as a first step. Then, a monitor module will dynamically assess the path quality and adaptively add and drop paths depending on their performance. We believe such a passive approach will introduce least overhead and likely to be more scalable in practice. The only drawback of this approach is that it requires a slightly larger buffer to handle the reordering, which is acceptable in today's computers.

2.1.6 Multipath Congestion Control

The conventional AIMD [26, 37] algorithm employed in TCP is easily implemented and works well in achieving fair bandwidth distribution between competing flows. Our congestion control algorithm is a variant of AIMD that uses information from multiple paths in a correlated manner. This is similar to the idea of Congestion Manager [19], where congestion control is performed for multiple applications for a single host. While the more recent TCP variants like TCP CUBIC [39] and Compound TCP [89] modified how the congestion window is increased/decreased to improve the TCP efficiency, we do not implement them because our purpose is to investigate the way to route around core link bottlenecks by using multiple paths, not improve the TCP efficiency itself. We believe our approach is compatible with the recent TCP variants.

In mTCP [104], congestion control is performed for each individual path without coordination among paths. In our experiments, we found that this strategy would be overly aggressive when there are a large number of paths and it has been shown that coordinated congestion control is better [97, 58], so we also adopt a coordinated approach. The congestion control algorithm of mPath is similar in many ways to that of MPTCP proposed and analyzed by Raiciu et al. and we have verified that our algorithm satisfies all the requirements that they proposed [97]. mPath differs from MPTCP in the design goal where mPath uses proxied paths to supplement the direct path only when the direct path is constrained by the core link bottleneck, while MPTCP tries to distribute the data fairly among the existing paths according to the path conditions. Our key innovation is a *load aggregation* mechanism that attempts to maximize the utilization on the direct path and causes

the congestion windows for redundant proxied paths to converge to zero.

There have also been a number of theoretical works on multipath congestion control algorithms based on fluid models [41] and control theory [94]. Raiciu et al. simulated these algorithms and found that they do not work well in practice [97].

2.1.7 Shared Bottleneck Detection

Several algorithms [80, 102, 104, 55] have been proposed to detect the shared bottleneck. These algorithms are based on two fundamental observations: i) losses or delays experienced by any two packets passing through the same point of congestion exhibit some degree of positive correlation; ii) losses or delays experienced by any two packets that do not share the same point of congestion will exhibit little or no correlation [80]. Rubenstein et al. proposed correlation testing techniques using either loss events or delays observed across paths [80]. FlowMate inferred shared bottlenecks by computing the correlation in packet delay [102]. mTCP detected shared bottlenecks with a list of timestamps that record the time of fast retransmit events [104]. Katabi et al. detected the shared congestion passively based on the observation that an aggregated arrival trace from flows that share a bottleneck has very different statistics from those that do not share a bottleneck [55]. In particular, the entropy of the inter-arrival times is much lower for aggregated traffic sharing a bottleneck.

All these algorithms were designed to detect static shared bottlenecks with the assumption that they do not change during the transmission. However, we found that the bottleneck could potentially change over time when we use detour paths. Hence, existing algorithms are not suitable for use in mPath. Instead, we

apply a simpler and more dynamic mechanism called *loss intervals* to quickly infer whether the packet losses happen at the bottleneck in the direct path and proxied paths.

2.2 Measurement Study of Cellular Data Networks

A number of measurement studies have been conducted under various kinds of cellular data networks, from the older networks like GPRS [69], 3G/UMTS [27] and 3G/CDMA [66] to the more recent networks like HSPA(+) [52, 90, 10, 75] and LTE [46, 96]. In this section, we summarize some of the interesting findings in the previous works.

2.2.1 Measurement of General Performance

Many existing works have measured the overall performance of the commercial cellular data networks in terms of throughput and delay [52, 90, 10, 75, 86]. Generally, these results painted a positive picture of the cellular data networks that the overall performance has improved significantly as the technology advanced. For example, Tan et al. mentioned that the HSDPA performs much better than the previous 3G/UMTS networks [90]. Sommers and Barford observed with a large set of data from SpeedTest [3] that while most of the 3G networks perform worse than WiFi network, the LTE has already outperformed the WiFi network [86]. We also observe similar trend in our measurement studies.

One common finding of the previous works is that the throughput and latency in cellular data networks may vary significantly [66, 90, 86]. Tan et al. observed that the capacity varies not only across different ISPs, but also across different

cells from the same ISP and it is practically impossible to predict the actual cell capacity with current known model [90]. Liu et al. found that the wireless channel data rate shows significant variability over long time scales on the order of hours, but retains high predictability over small time scales on the order of milliseconds [66]. Sommers and Barford observed that the variation of the cellular data networks is much higher than that of the WiFi network [86]. Our measurement result shows that the actual speed the users could achieve varies significantly in minutes as the result of the resource sharing between users.

The huge queuing delay of the cellular data networks is also investigated by many researchers [51, 90, 14, 29, 63]. Tan et al. observed that the queuing delay for data services is significant and the average latency can be several seconds [90]. Jiang et al. measured the buffers of 3G/4G networks for the four largest U.S. carriers as well as the largest ISP in Korea using TCP and examined the bufferbloat problem [51]. They observed that the delay of the cellular data networks can be quite large because the existing of large buffers. Other works focused on measuring and characterizing the delay of cellular data networks [14, 29, 63]. Our work extends these works by investigating the buffer sizing and queuing policies of different mobile ISPs, and we have found some surprising differences among the three local ISPs, which resulted significant differences in the queuing delay.

Winstein et al. mentioned in passing that packet arrivals on LTE links do not follow an observable isochronicity [96]. They examined the inter-packet arrival time and proposed to use the pattern to estimate the number of packets that should be sent in the near future. In our measurement studies, we provide the detailed measurements that corroborate their claims made in [96], and discuss the implications of the observed burstiness on instantaneous throughput estimation.

2.2.2 Measurement of Interactions between Layers

Since there exists significant differences in the physical and MAC layer between the cellular data networks and the conventional wired or WiFi networks [91], many measurement studies were conducted to understand the effect of the wireless channel to the applications and network protocols.

Cicco and Mascolo evaluated the congestion control algorithms of Reno, BIC and Westwood TCP over the earlier UMTS network [27]. They found that i) a single TCP connection will under-utilize the available downlink capacity, but will fully utilize the uplink; ii) the three TCP variants they investigated perform similar; iii) the queuing delay can be very large for the TCP. Our measurement studies under the more advanced HSPA(+) network supplement their findings.

Liu et al. investigated the effects of cellular channel to the transport protocols under the CDMA 1xEV-DO network [66]. They found that the loss-based TCP variants perform similar in throughput and are unaffected by channel variations due to the presence of large buffers, while the delay-based TCP like Vegas [22] performs relatively worse. However, we find that the delay-based algorithm is useful in controlling the queuing delay and with an effective buffer control mechanism, the performance degradation can be negligible.

Aggarwal et al. discussed the fairness of 3G/HSPA network and found that the fairness of TCP is adversely affected by a mismatch between the congestion control algorithm and the scheduling mechanism of Radio Access Network (RAN) [6]. Our measurement results supplement their arguments that the fairness is actually kept well when the link is not under-utilized. It seems that the fairness control of the TCP is redundant and not necessary.

A recent study conducted by Huang et al. showed various interesting effects of network protocols and application behaviors on the performance under the LTE network [46]. They observed that: i) the LTE network has significantly shorter state promotion delays and lower RTTs than those of 3G network; ii) many TCP connections significantly under-utilize the available bandwidth. iii) the application behaviors and parameter settings are not LTE-friendly. Based on these observations, they highlighted the needs to develop more LTE-friendly protocols. We agree with their statements and we further examine the possibility of eliminating the congestion control towards cross traffic in the cellular data networks.

2.2.3 Mobility Performance Measurements

Comparing to the conventional wired or WiFi networks, the cellular data networks are more advantage in supporting the mobility. Many research works were conducted to understand the influence of the mobility on the performance.

Tso et al. suggested that the mobility is a double-edged sword because it could reduce the performance significantly and improve the fairness at the same time [92]. They also observed that the triggering and the final results of handoffs are often unpredictable. Liu et al. observed that the variation in mobile scenario is much higher than stationary scenario and they found that the current mechanism like opportunistic channel-aware scheduler is effective and typically yields more gains for mobile scenario [66].

Deshpande et al. compared the 3G network and WiFi network performance under vehicular mobility environment [28]. They found that WiFi network has frequent disconnections but a faster speed when connected. The 3G network, on

the contrast, offers lower throughput but better coverage and connections. These results suggested that a multipath solution using both 3G network and WiFi network could be an effective design, which is also proposed in [97, 20, 25] and adopted by the IOS 7 [21].

While these works provided us good insights about the performance under the mobile scenario, we only focus on the stationary performance and do not investigate the mobility issues in the current measurement studies. We are interested in the performance under mobile scenarios, e.g. the performance under Mass Rapid Transit (MRT) in Singapore, and leave it as a future direction of the research.

2.2.4 Measurement of Power Characteristics

Qian et al. undertook a detailed exploration of the power characteristics and the radio resource control (RRC) state machine in the 3G/UMTS networks by analyzing real cellular traces and measuring from real users [76]. By accurate interfering of the RRC state machine, they characterized the behaviors of the RRC state machine. They also found that the RRC state machine may influence the performance and power consumption a lot. Huang et al. followed this work and investigated the RRC state machine and power characteristics in 4G/LTE networks [45].

While in our measurement studies, we do not measure the RRC state machine and power characteristics directly, these observations assist us in the analysis of the performance. For example, we observe that the delays of some very first packets are huge comparing to others, which can be explained by the state promotion model of these two works. As our measurement studies mainly focus on the continuous performance of the cellular data networks, we ignore those initial packets

that we only measure the performance after the power state is promoted.

2.3 Problem of Saturated Uplink

The impact of saturated uplink on download performance is a well-studied problem. This problem was first characterized as the *ACK compression* problem that the ACKs of the downstream TCP get compressed in the uplink buffer when upload speeds are low. The ACKs are then sent out in bursts, causing the self-clocking mechanism of TCP to break [103, 53]. However, more recently, Heusse et al. showed that in practice, the *Data Pendulum* effect is more prevalent than ACK compression [42]. According to their observation, when there are concurrent upload and download connections, it is possible for the buffers on both sides of the connections take turns to fill up and fully utilize their links while the other idles, provided the buffer sizes are configured correctly. However, when the buffer sizes are misconfigured relative to the link capacities, the link with lower capacity and larger buffer will become the sole bottleneck. We have verified that in cellular data networks, the uplink can become a bottleneck and cause the downlink to be under-utilized quite often.

2.3.1 Previous Solutions

Many different solutions have been proposed to solve the uplink saturation problem, including: i) prioritizing the ACKs and optimizing how the ACKs are sent [53, 18, 70]; ii) using separate queues for ACKs and data packets [74]; iii) using sender-side congestion control algorithms that are designed to achieve low delay, like Vegas [22] and LEDBAT [84]; iv) using parallel TCP connections [85, 40];

v) eliminating TCP ACK clocking [65].

Balakrishnan et al. proposed many techniques to improve the performance for the two-way traffic under the asymmetric links [18]. Their methods mainly focus on optimizing the ACKs, including: i) decreasing the rate of acknowledgments on the constrained reverse channel by *ACK congestion control* and *ACK filtering*; ii) a TCP sender adaptation mechanism that reduces the source burstiness when ACKs are infrequent; iii) scheduling the ACKs firstly at the reverse bottleneck router. These techniques were further specified in RFC 3449 [17]. Similar, Ming-Chit et al. proposed to vary the number of data packets acknowledged by an ACK based on the estimated congestion window [70]. Kalampoukas et al. also examined the methods like providing priority to ACKs and limiting the data packet queue [53]. They then suggested to use a connection-level bandwidth allocation mechanism in order to guarantee a minimum throughput for the slow connection and make the throughput of fast connection sensitive to only its own parameters. All these techniques were proposed to solve the *ACK compression* problem, and hence they were not sufficient to solve the uplink saturation problem caused by the *Data Pendulum* effect.

A recent solution with the awareness of *Data Pendulum* effect was proposed in order to achieve full resource utilization in both directions [74]. The key idea of this work was an *Asymmetric Queuing* mechanism that serves the TCP data and ACK traffic though two different queues. However, this work only provided simulation results in residential broadband networks, and it is not clear whether it could be deployed in cellular data networks practically.

There are also some sender-side congestion control algorithms that are designed for the purpose of achieving low delay, like Vegas [22] and LEDBAT [84].

These works could potentially be used in cellular data networks, and actually in our experiments, we found that the Vegas performed quite well in reducing the uplink delay. However, these methods require client-side modifications, which makes them harder to be deployed for all the mobile devices [4].

Parallel connections can also be used to improve the efficiency of the TCP [85, 40]. However, we verified with experiments that parallel TCP flows are not sufficient to improve the downlink utilization because a slow and saturated uplink will ultimately delay the ACKs and become the bottleneck.

RSFC is much easier and practical to deploy because it only needs minor modifications to the TCP stack at the receiver side of the upstream (which is the server) but strictly *no modifications* at the mobile device or the router. The current architecture of the cellular data networks makes it even easier to deploy RSFC, that it can be deployed at the ISP's transparent proxies. More recently, Leong et al. proposed a new TCP variant called TCP-RRE to solve a more general problem caused by the asymmetric link in a different way. Their key idea is to mitigate the egregious ACK delays by eliminating TCP ACK clocking at the sender side of the downstream and use rate control instead of window-based congestion control. However, this solution will still be constrained by the receive window, once the ACKs are delayed too much, since it does not reduce the number of data packets in the uplink buffer. We believe the combination of TCP-RRE and RSFC will provide a more complete solution to this problem.

2.3.2 Receiver-side Flow Control

The technique of controlling the advertised receive window, `rwnd`, to regulate a TCP flow is not new. Many previous works have used this technique in different contexts and for various purposes [35, 87, 57, 54, 12, 24, 51].

Freeze-TCP [35] advertised a zero window from the mobile device when it detected poor network conditions, i.e. a temporal high bit error rates of the wireless links or a temporary disconnection due to signal fading or handover. With this method, the receiver could prevent the sender from sending any more packets and reduce the sender's effective congestion window. When the network condition has recovered, the receiver could indicate the sender to resume the sending by advertising a normal window.

Spring et al. used receiver based congestion control policies to improve the performance for different types of concurrent TCP flows [87]. By prioritizing the various flow types with different value of `rwnd`, it could improve the response time for interactive network applications while maintaining high throughput for bulk-transfer connections. Key et al. exploited similar ideas to create a low priority background transfer service [57].

The explicit window adaptation scheme proposed by Kalampoukas et al. also used `rwnd` to control the downstream queue size to achieve fairness between window-based and rate-based congestion control algorithm [54]. Andrew et al. used a similar method to fairly share the available bandwidth between users [12].

Chan and Ramjee evaluated the impact of link layer retransmission and opportunistic schedulers on TCP performance [24, 23]. They then proposed *Window Regulator* algorithms that used the `rwnd` to convey the instantaneous wireless

channel conditions to the sender and an ACK buffer to absorb the channel variants. Their mechanisms work at the network layer and require the access to the radio network controller (RNC).

Jiang et al. also proposed a dynamic receive window adjustment (DRWA) mechanism to tackle the bufferbloat problem [51]. The bufferbloat problem, is a phenomenon where an extremely long delay is caused by the oversized buffers [34]. In DRWA, it increases the `rwnd` when the current RTT is close to the observed minimum RTT and decreases the `rwnd` when the RTT becomes larger due to queuing delay.

In RSFC, we solved a different problem from these previous proposals that our intention is to improve the downlink utilization in cellular data networks by controlling the number of data packets queued in the uplink buffer when there is concurrent download and upload. Our key contribution here is to apply this technology in the scenario of cellular data networks.

2.3.3 TCP Buffer Management

Another category of solutions is to set the buffer size properly. The classic rule of thumb was to set the buffer size to the bandwidth-delay product $(\overline{RTT} \times C)$ [93] and more recently, it was suggested that it should be sufficient to set the buffer size to $(\overline{RTT} \times C)/\sqrt{n}$ [13], where C is the data rate of the connection and n is the number of long-lived flows. Enachescu et al. suggested to reduce the buffer size further to $O(\log W)$, where W is the window size of each flow [30]. They claimed that the buffer size could be reduced to a few tens of packets with only trivial sacrificing in utilization, under the assumption that the data packets arrived

in a Poisson arrival pattern. A TCP pacing [5] method could be used to achieve the Poisson arrival pattern.

However, unlike the Internet routers which typically have fixed throughput, there can be significant variation in the cellular link speeds. Hence, a fixed-sized approach to buffer sizing is not feasible in the cellular data network environment. In order to fully utilize the available capacity of the uplink, a dynamic approach to size the buffers properly is necessary. However, this method would incur significant modifications at the mobile device side. Again, it is not easy for all the mobile devices to be deployed with such mechanism [4]. We believe our techniques of regulating the uplink queue length at the receiver side is a more feasible and general way than controlling the buffer size directly at the sender side.

Chapter 3

Massively-Multipath Source Routing

In the conventional wired Internet, the assumption was that the bottlenecks were always at the access links. However, with the tremendous increase of the access link speed, it was found that the bottlenecks were moving deeper into the core Internet [8]. As last mile bandwidth is set to keep increasing dramatically over the next few years [47], we expect that this trend will accelerate and end-to-end data transfers will be increasingly constrained by core link bottlenecks. On the other hand, research has shown that there are many detour paths that can potentially be used to improve the overall performance [81, 44].

In this chapter, we investigate this problem and propose a new massively-multipath (mPath) source routing system to exploit the path diversity and better utilize the access link. Our key mechanism is a combined congestion control and path selection algorithm that can identify bottlenecks, apportion traffic appropriately, and inter-operate with existing TCP flows in a TCP-friendly man-

ner. The algorithm is a variant of the classic additive increase/multiplicative decrease (AIMD) algorithm [26] that infers shared bottlenecks from correlated packet losses and uses an operation called *load aggregation* to maximize the utilization of the direct path.

We organize this chapter as follows: in Section 3.1, we describe the mPath algorithm and the design of our system. In Section 3.2, we present a theoretical model for our multipath congestion control algorithm. In Section 3.3, we present our evaluation results.

3.1 System Design & Implementation

We first describe the design and implementation of the mPath source routing system. As illustrated in Figure 1.1, the network is composed of a set of proxies that are tracked by a central registration server (RS). Proxies in mPath are light-weight because they do not maintain connection state. The destination address is embedded in every data packet, so proxies can simply forward the packets received to the destination node. The RS tracks the active proxies in the system and returns a subset of the proxies to a source node when it needs to initiate a new mPath connection. We currently implement the RS as a simple server application, but it can be easily replaced with a distributed system for greater reliability and/or scalability. The application at the end-hosts is provided with a connection-based stream-like interface similar to TCP to perform the data transfer, even though the underlying protocols supporting this interface are UDP-based and therefore connectionless. Depending on the nature of the application supported, the proxies can either be dedicated servers or mPath clients.

We use UDP instead of TCP for various practical reasons. For one, mPath needs direct control over the packet transmissions (and retransmissions) to implement the congestion control algorithm that coordinates between the different mPath flows. Moreover, the use of TCP would limit the scalability of the system since a source node might need to communicate with hundreds of proxies and the overhead of opening and maintaining hundreds of TCP connections is excessive. Given that the majority of hosts on the Internet are behind Network Address Translators (NATs), it is also advantageous to use UDP because the NAT hole punching process for UDP is typically simpler, faster and more likely to succeed than that for TCP [83].

A data transfer begins when the source node establishes a direct connection to the destination. Simultaneously, the source node also queries the RS to obtain a list of available proxies. The data stream from the application is packetized and the packets are initially sent only on the direct path. When congestion is detected on the direct path, packets are forwarded via the proxies in an attempt to increase the throughput. Acknowledgments for the received data packets are sent from the destination back to the source along the direct path. A congestion manager and scheduler module monitors the acknowledgments to determine the quality of the various paths and controls the transmission and retransmission of packets. Finally, packets are reordered at the destination to produce the original data stream. This process is illustrated in Figure 3.1.

In general, the congestion control on the direct path is similar to TCP. Modifications to the standard TCP AIMD algorithm were made to coordinate between the multiple paths and ensure that the combined paths do not behave more aggressively than TCP in increasing the overall congestion window. Also, we imple-

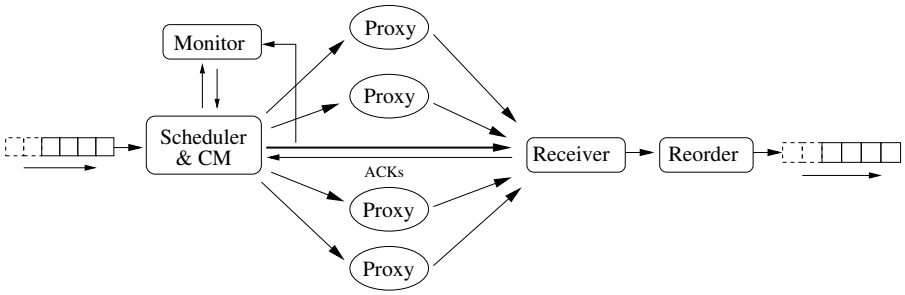


Figure 3.1: Overview of mPath.

mented a simple algorithm to infer correlated losses between the direct path and proxied paths, and a *load aggregation* mechanism to aggregate traffic onto the direct path when a shared bottleneck is detected. Our algorithm causes the traffic on redundant proxied paths to converge to zero over time. While the overall idea is relatively simple, there are a number of implementation details required to get the system to work in a practical setting. These details are described in the following sub-sections.

3.1.1 Proxy Probing

Given the large number of proxies, each mPath connection starts with a probing phase that uses data packets to identify proxies that are unreachable, non-operational or exhibit non-transitive connectivity [32]. As mPath is tolerant of packet reordering and losses, it is acceptable to use data packets in the probing process instead of active probe packets.

Probing starts immediately after the source establishes a direct connection to the destination and receives a proxy list from the RS. To prevent path probing from interfering with the data transfer process, we limit the probing rate to one probe every 250 ms, which is approximately the average inter-continental

roundtrip time [88]. Sending one probe packet every RTT will not likely interfere with the data transfer because the sender is expected to forward tens or hundreds of packets in one RTT. When the sender decides to probe a proxy, it will randomly select a proxy from the proxy list and attempt to forward a data packet through it to the destination. If the sender receives an ACK for the data packet within τ seconds, the proxy is considered usable and is added to the *available list*, which is the set of proxies that can be used to forward packets. On the other hand, if the sender fails to receive an ACK within τ seconds after two consecutive probing attempts, the proxy will be marked as *unusable*. Once all the proxies in the list have been probed, mPath will request for more proxies from the RS. Clearly, the threshold τ limits the maximum RTT of the proxied paths in the system and controls the trade-off between the quality of the paths selected and the size of the buffer required at the destination to handle reordering. We show in Section 3.3.7 that a value of τ that is two times of the direct path's RTT yields a sufficiently large number of good proxied paths and an acceptable amount of reordering.

3.1.2 Sequence Numbers & Acknowledgments

TCP was designed for a single direct path and only needs one sequence number to handle both ordering and the detection of packet loss. In mPath, packet transmission across multiple paths can result in significant reordering at the receiver. A single sequence number would suffice to preserve ordering. However, having only one sequence number would make it harder to detect packet losses for individual paths, which is needed for proper congestion control. We considered using SACK and the scoreboard data structure proposed in mTCP [104] to record in-

formation for all the paths. However, we found this method to be inefficient in handling hundreds of paths. Like MPTCP [97], we use two sequence numbers: a *stream sequence number* and a *path sequence number*. The stream sequence number is used to identify and retransmit lost packets, while the path sequence number is used to detect packet loss and control the congestion window for each path.

Acknowledgments. When the destination successfully receives a number of packets, an ACK packet is sent back to the sender, which contains both the global stream sequence number as well as a set of path entries for the paths on which the receiver had received data packets. Like TCP, the receiver cumulatively acknowledges the receipt of packets by sending back the stream sequence number of the earliest missing packet. In addition, each ACK packet also contains a set of path entries, each recording the largest sequence number seen and the accumulated count of the packet losses observed on the associated path. The path-level acknowledgment is based on the latest packet received rather than the earliest missing packet because we have decoupled stream ordering from path packet losses and we can use new sequence numbers for retransmissions. This also allows mPath to retransmit lost packets on a different path. The accumulated packet loss count is included to ensure that the sender has a more accurate view of the packet losses on each path. Given that there are occasional losses of ACK packets, this allows the congestion control algorithm to recover in the event that it wrongly infers that there are data packet losses.

Negative Acknowledgments. The sender is notified of the holes in the stream sequence with a stream-level NACK (SNACK) packet and of holes in the path sequence with a path-level NACK (PNACK) packet. As holes in the stream se-

quence can be the result of reordering across multiple paths, SNACKs are not sent immediately when the holes are detected. Since only paths with RTTs less than τ seconds are used as proxies, we wait up to τ seconds for holes in the stream sequence to be filled before sending a SNACK to avoid false positives. To prevent an overflow of the receiver's buffer arising from the delayed retransmission requests, SNACKs will be sent immediately if the receiver's buffer is more than half full. When the sender receives a SNACK, it retransmits the required packets immediately but does not modify the congestion window. Unlike SNACKs, PNACKs are sent as soon as packet losses are detected. The sender reacts to a PNACK according to the congestion control algorithm and performs a quick retransmission with a newly selected path.

ACK Aggregation. mPath sends ACKs, SNACKs and PNACKs via the direct return path. We found that it is quite common for ACKs to be lost when there is congestion on the return path. If ACKs were sent for every packet, mPath might experience greater ACK losses than TCP because the number of ACK packets for mPath can exceed the number of data packets sent on the direct path if a large number of data packets are sent along proxied paths. To reduce congestion, we reduce the rate at which ACK packets are generated by aggregating up to 10 acknowledgments into a single ACK packet. To ensure that acknowledgments are not delayed excessively, we also limit the delay to no more than 10 ms. We also considered the possibility of using the proxied paths to send the ACKs, but we found that it will only make the system more complicated and increase delay, without achieving any benefits over our chosen method of using the direct return path.

3.1.3 Path Scheduling & Congestion Control

As the quality of proxied paths can vary significantly and change over time, it is not possible to statically determine the optimal set of paths. Previous work on path selection is mostly based on active probing, i.e. using *ping* [11] or *traceroute* [104], which incurs a large overhead and does not yield accurate results for the entire transmission period. Our approach to path selection is to passively detect changes in path quality and to dynamically react to these changes. This is achieved with a multipath additive increase/multiplicative decrease (AIMD) algorithm in a coordinated manner [97, 58]. As the direct path incurs less overhead and typically has lower delay than the proxied paths, the design goal of our congestion control is to use the direct path as much as possible and supplement the direct path with the proxied paths only when the direct path is constrained by the core link bottleneck.

Proxied Path Creation. mPath first sends packets on the direct path to the destination. In this state, the system controls congestion, much like standard TCP, by employing a 'slow-start' phase and halving the congestion window if loss is detected. However, in addition to halving the congestion window, packet loss may also trigger the creation of proxied paths. The number of new proxied paths to be created when a loss is detected is a proportion β of the direct path's congestion window (and limited by the number of paths in the available list). We found that $\beta = 0.25$ achieves a good trade-off between the time taken to find good proxies and the utilization of the direct path. Proxies are chosen at random from the unused proxies in the available list. If all the available proxies have been used before, mPath chooses the best proxy that it has observed thus far. For each newly created

proxied path i, mPath also maintains a congestion window w_i that is initially set to one. To prevent rapid and uncontrolled creation of proxied paths, the system will only create new paths after all existing paths have encountered loss.

Multipath AIMD. mPath eliminates bad paths and exploits good paths by scaling congestion windows with an additive increase/multiplicative decrease (AIMD) algorithm [26]. Like TCP, a packet loss causes the congestion window of the affected path to be halved. The main difference between mPath and TCP is in how the congestion windows are increased when ACKs are received. During slow-start, the congestion window of the direct path will increase by one for every ACK received. In congestion control mode, cumulative ACKs received will increase the congestion window of either the direct path or the proxied paths. The congestion window of the proxied paths is increased with probability P and the congestion window of the direct path is increased with probability $1-P$. The probability P is obtained with the following formula:

$$P = \frac{w_0}{W}\rho + \frac{W - w_0}{W} \tag{3.1}$$

where ρ is the proportion of proxied paths that have not encountered loss, w_0 is the congestion window of the direct path, and $W = \sum w_i$ is the total congestion window (over all paths inclusive of the direct path). The intuition is to use P to apportion the load between the direct path and the proxied paths according to their states. If some proxied paths have never encountered loss, we would like to increase their congestion windows rapidly; if all the proxied paths have encountered loss, then any increase in the overall congestion window should be divided between the direct and proxied paths according to their estimated relative

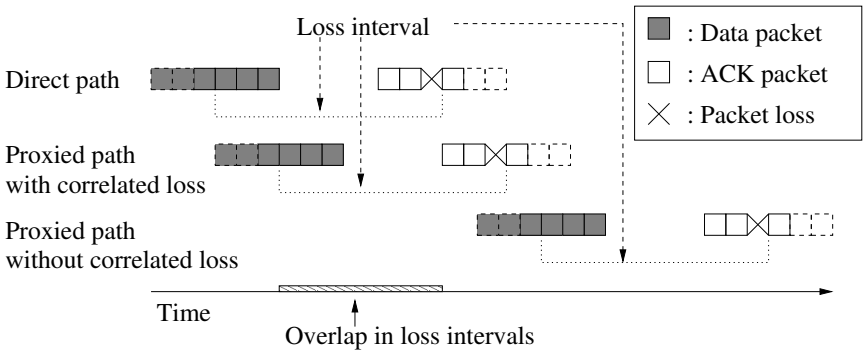


Figure 3.2: Inference of correlated packet losses.

available bandwidths.

When we decide to increase the congestion window of an existing proxied path, mPath selects an active path that has recently received an ACK and increases its congestion window by one. Paths that have never encountered loss are given higher priority. If all the paths have encountered loss, the addition goes to path i with probability $\frac{w_i}{\sum w_j}$, where w_j is the congestion window of proxied paths that have recently received ACKs.

Shared Bottleneck Detection. We need to identify the shared bottlenecks between proxied paths and the direct path so that we can shift traffic to the direct path. We use a simple but effective scheme to infer the existence of such bottlenecks: mPath records the transmission time for every packet sent and the loss detection time if an ACK arrives indicating that packets were not received. Each path then maintains a time range ("loss interval") indicating when the last lost packet was detected and the time that the packet was sent on the path. If the loss interval on the direct path overlaps with the loss interval on a proxied path, we deduce that the packet losses are correlated and the paths share a common bottleneck. This is illustrated in Figure 3.2.

One key drawback of this method is that it is likely to produce false positives. However, we argue that if two paths do not share a bottleneck, a false positive would require packet losses to occur on both paths within a window that is typically less than several hundred milliseconds. As losses happening on two independent paths that do not share the same point of congestion will exhibit little or no correlation [80], we expect the probability of such an event to be extremely low. In addition, a single false positive will not influence our algorithm significantly and the probability that several false positive shared losses will occur between the direct path and the same proxied path is even lower (the probability would be p^n, where p is the probability of one false positive shared loss and n is the number of false positive events). There are no false positives in our emulation experiments and the good performance achieved in our PlanetLab experiments suggests that false positives are not a significant concern.

Load Aggregation. Upon detecting the shared bottleneck, mPath will then move a proportion $\min(\alpha, \frac{w_0}{\sum w_j})$ of the proxied path's remaining congestion window to the direct path, where w_j is the congestion window for the proxied path that experienced correlated packet loss. The upper bound $\frac{w_0}{\sum w_j}$ guarantees that the congestion window of the direct path will not be more than w_0. In other words, mPath will decrease proxied path i's congestion window w_i to half for a normal loss on that path, but decrease it to $\frac{w_i}{2}(1 - \min(\alpha, \frac{w_0}{\sum w_j}))$ for a correlated packet loss and add $\frac{w_i}{2}\min(\alpha, \frac{w_0}{\sum w_j})$ back to the congestion window of the direct path. We call this operation *load aggregation*. We found that $\alpha = 0.5$ achieves a good trade-off between the utilization of good proxied paths and the direct path, and reduces the utilization of bad paths relatively quickly.

We choose to gradually decrease the congestion window of a proxied path in-

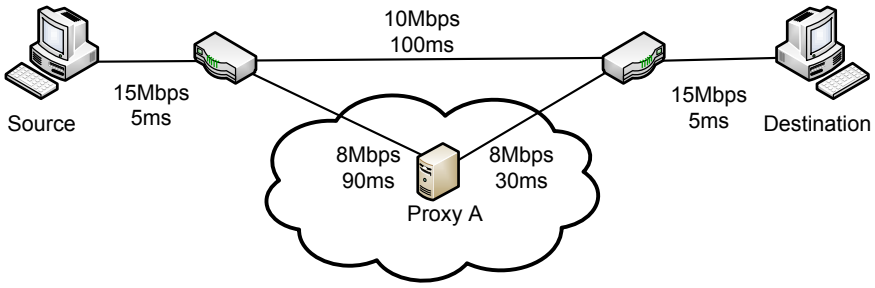


Figure 3.3: An example of bottleneck oscillation.

stead of dropping the path completely in order to prevent *bottleneck oscillation*. We illustrated this with an example in Figure 3.3. In this example, the transmission is initially limited by the 10 Mbps core link bottleneck on the direct path. As new proxied paths that can route around the core link bottleneck are found (e.g. proxy A), the common access link with capacity 15 Mbps will become the new bottleneck. When a correlated packet loss is detected at the new bottleneck, the naive approach of dropping proxy A completely will cause the bottleneck to shift back to the core. If mPath then uses proxy A (or some other good proxy) to improve throughput, the bottleneck will eventually shift back to the 15 Mbps access link and the system will oscillate. By aggregating the congestion windows of the proxied paths to the direct path, the stability of the system is improved.

Handling Timeouts. Like TCP, mPath detects timeout for all paths with a mechanism based on the estimated RTT for each path. The direct path reacts to a timeout by reverting to the slow-start state. However, when a timeout occurs for a proxied path, we drop it instead of reverting to slow-start since it is easy for mPath to either find a replacement among the unused proxied paths or redistribute the load across existing paths. The path can be dropped temporarily or permanently depending on the historical contribution of the path. If the path's contribution to

the throughput is significantly below average compared to other proxied paths in its lifetime, it will be marked as unavailable and dropped permanently. Otherwise, it will only be dropped temporarily and may be reused at a later time.

3.2 Analysis of Multipath AIMD

In this section, we extend the classic Chiu and Jain AIMD model [26] to analyze multipath AIMD. We show that our algorithm (i) is TCP-friendly, (ii) maximizes the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) rapidly eliminates any redundant proxied paths.

Following the notation in [26], we obtain the multipath model for a single user, which is illustrated in Figure 3.4. The user imposes a total load of x on the system, with a load of x_i on each path i. Path 0 refers to the direct path, which has a core link capacity of X_{goal_0}, while paths 1 to n are the proxied paths available to the user. Without loss of generality, we assume that k out of the n proxied paths are limited by core (or AS-AS peering) link bottlenecks, each with a capacity of $X_{goal_i}, 1 \leq i \leq k$, while the remaining paths from $k+1$ to n are free of congestion and can accept more load. The capacity at the access link bottleneck is denoted with X_{goal}. We also define Y to be the feedback vector to the user, which is a tuple comprising of binary feedback values y_i for each path i. A positive feedback ($y_i = 0$) implies that there is no packet loss on path i, while a negative feedback ($y_i = 1$) implies that congestion has occurred.

When a negative feedback is received, mPath will halve the load on the associated path, and if it is a correlated packet loss, mPath will perform load aggregation. When a positive feedback is received, mPath will increase the load by one with

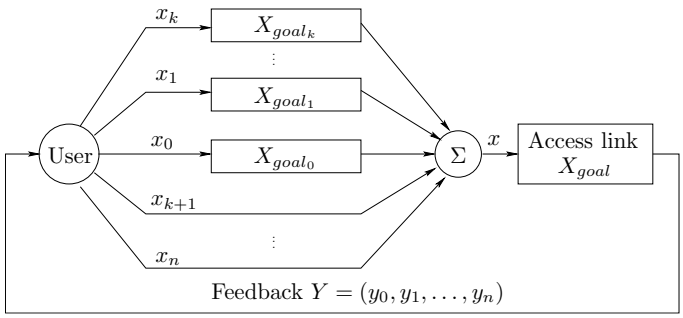


Figure 3.4: Model for a single user using multiple paths.

probability $\frac{x_i}{\sum_{i=0}^{n} x_i}$, denoted by γ_i. Clearly, γ_i is less than 1 for each path i and $\sum_{i=0}^{n} \gamma_i = 1$ at the access link bottleneck where all paths are aggregated. Thus, for one mPath flow, the additive-increase value on any path is always less than or equal to that of TCP and the multiplicative-decrease value is always equal to that of TCP. This implies that mPath is TCP-friendly. In addition, because the behavior of mPath will be similar to that of TCP at the access link, mPath will compete for resources fairly and efficiently [26].

There are three scenarios under which a bottleneck can occur: (i) the direct path is limited by a core link bottleneck but there is insufficient free capacity on the available proxied paths to saturate the access link; (ii) the direct path is limited by a core link bottleneck and enough free capacity via other paths can be found to saturate the access link; or (iii) the bottleneck is entirely at the access link.

(i) Core Link Bottleneck, Insufficient Capacity. The first case is where the direct path experiences congestion on a core link and mPath cannot find a set of alternative paths to fully saturate the common access link. Load aggregation will ensure that any proxied paths sharing the same core link bottleneck as the direct path will eventually be dropped, i.e. only the proxied paths that do *not* share a bottleneck with the direct path will be retained. In the steady state, the congestion

windows of the active paths would be oscillating in a manner that is equivalent to the *semicoupled algorithm* [97] for $a = 1$, where a is a constant that determines the aggressiveness of the congestion control algorithm. Raiciu et al. [97] determined that only the paths with loss rates satisfying the following condition will be used:

$$(1 - p_r)\frac{1}{\hat{x}} = p_r\frac{\hat{x_r}}{2} \tag{3.2}$$

where $\hat{x}$ is the average load of the user, and p_r and $\hat{x_r}$ are the loss rate and average load on path r respectively. This condition follows from the intuition that paths must either reach an equilibrium for the average increase and decrease of their load or converge to zero and get dropped. In other words, mPath will distribute as large a load as possible to paths with low loss rates and drop all the paths that have loss rates too high to satisfy condition (3.2).

(ii) Core Link Bottleneck, Excess Capacity Sufficient. The second case is where mPath is most effective. When there is a bottleneck on the direct path and sufficient core link capacity exists, the additive-increase phase will fully saturate the access link by utilizing the free capacity on proxied paths $k+1$ to n. This occurs even with a small additive-increase value of $\gamma_i < 1$, for $k+1 \leq i \leq n$, since these paths experience minimal packet loss. On the other hand, the congested paths 1 to k will experience packet loss and their loads will undergo multiplicative decrease, eventually converging to zero.

(iii) Access Link Bottleneck. Finally, if the flow is limited by an access link bottleneck, using multiple paths will not help. The proxied paths will experience correlated packet losses with the direct path, and, over time, load aggregation will move most of the traffic onto the direct path and cause the load for all the proxied

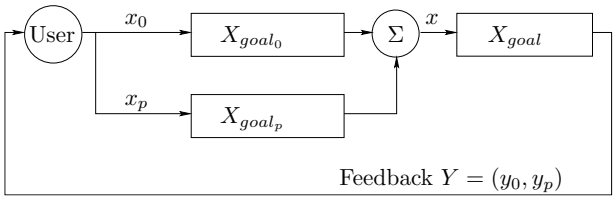


Figure 3.5: Model of a shared access link bottleneck.

paths to converge to zero.

We further analyze the performance of mPath for scenarios (ii) and (iii) in more detail. We consider a simple flow where there is only one proxied path, with a core link capacity of X_{goal_p}, sharing a common access link with the direct path, as shown in Figure 3.5. We define the load at time t on the direct path's core, proxied path's core and access link to be $x_0(t)$, $x_p(t)$ and $x(t)$ respectively. Depending on the feedback values received at time t, given by $Y(t) = (y_0(t), y_p(t))$, mPath will derive the loads at time $t+1$ as follows:

1. $Y(t) = (0,0)$. When positive feedback is received on both paths, the load on the direct and proxied path is increased probabilistically:

$$\begin{aligned} x_0(t+1) &= x_0(t) + 1 \\ &\quad \text{with probability} \quad \gamma_0 = \tfrac{x_0(t)}{x_0(t)+x_p(t)} \\ x_p(t+1) &= x_p(t) + 1 \\ &\quad \text{with probability} \quad \gamma_p = \tfrac{x_p(t)}{x_0(t)+x_p(t)} \end{aligned} \tag{3.3}$$

2. $Y(t) = (1,0)$. When only the direct path has negative feedback, we obtain:

$$\begin{aligned} x_0(t+1) &= \tfrac{x_0(t)}{2} \\ x_p(t+1) &= x_p(t) + 1 \end{aligned} \tag{3.4}$$

3. $Y(t) = (0,1)$. When only the proxied path has negative feedback, we obtain:

$$\begin{aligned} x_0(t+1) &= x_0(t) + 1 \\ x_p(t+1) &= \tfrac{x_p(t)}{2} \end{aligned} \tag{3.5}$$

4. $Y(t) = (1,1)$. When negative feedback is received on both paths, the loads will be halved and load aggregation (with parameter α) will be performed to shift part of the load from the proxied path to the direct path:

$$\begin{aligned} x_0(t+1) &= \tfrac{x_0(t)}{2} + \alpha\tfrac{x_p(t)}{2} \\ x_p(t+1) &= (1-\alpha)\tfrac{x_p(t)}{2} \end{aligned} \tag{3.6}$$

Now assume that X_{goal} is saturated at time t. This implies that there will be shared loss (i.e. $Y(t+1) = (1,1)$), which will cause the overall load to be halved ($x(t+1) = x(t)/2$) and load aggregation to be performed according to Equation (3.6). If bottleneck conditions do not change and there are no other losses, it will take approximately $x(t)/2$ time intervals for X_{goal} to be saturated again to produce another shared loss, since the average increase for $x(t)$ is 1 at each time interval, as described by Equation (3.3). We can then deduce that after k intervals of shared losses under these conditions, at time $u = t + (k-1)x(t)/2 + 1$, the loads would be:

$$\begin{aligned} x_0(u) &= \tfrac{x_0(t)}{2} + [1 - (1-\alpha)^k]\tfrac{x_p(t)}{2} \\ x_p(u) &= (1-\alpha)^k \tfrac{x_p(t)}{2} \end{aligned} \tag{3.7}$$

For scenario (ii), where the bottleneck is in the core and sufficient capacity exists (i.e. X_{goal_p} is large), x_0 and x_p would increase until either X_{goal} or X_{goal_0}

is achieved. If X_{goal} is achieved first, correlated packet losses will be detected and load will aggregate to the direct path at an exponential rate until there are no more correlated losses as described by Equation (3.7). This behavior ensures that the direct path is never under-utilized. The overall performance in this case is the same as that for a normal TCP flow competing for resources constrained by the access link bottleneck. If X_{goal_0} is achieved first, mPath has over-utilized the direct path and caused congestion, so x_0 will be halved and x_p will increase (as described by Equation (3.4)) until X_{goal} is achieved. This means that mPath will always maximize the utilization of the access link without under-utilizing the direct path when sufficient core link capacity is available.

For scenario (iii), where the access link is the bottleneck, the proxied path is redundant. All losses will occur at the shared access link and load aggregation will always be performed to shift traffic to the direct path. Thus, as described by Equation (3.7), the load of the proxied path (x_p) will be aggregated to the direct path at an exponential rate until x_p is reduced to zero (i.e. the path is dropped).

Like the classic AIMD model [26], we assumed a synchronous feedback/control loop and omitted the RTTs of the paths in our analysis in order to keep the model tractable. We can extend the model to consider paths with different RTTs, but doing so will not shed significantly more insight. An extension of the classic AIMD model incorporating the RTTs [101], showed that the system would simply be biased against paths with longer RTTs. As mPath is a variant of AIMD, we can expect similar results. We show in Section 3.3.1 that even with paths of varying RTTs, mPath behaves as expected.

3.3 Performance Evaluation

We evaluated mPath by running experiments on both Emulab and PlanetLab to show that mPath (i) behaves in a manner consistent with the model described in Section 3.2, (ii) is practical and can often achieve significant improvements in throughput, and (iii) is scalable and that end-hosts can be used as proxies. We also investigated the trade-off associated with the choice of system parameters.

In our experiments, we compared mPath with TCP by sending continuous streams of randomly generated data. However, we did not use the native implementation of TCP due to two issues: (i) bias towards different transport protocols (e.g. by firewalls or routing policies) may skew the results; and (ii) PlanetLab limits the TCP window size, which limits the maximum throughput attainable. Therefore, we used an implementation of TCP (New Reno[43]) based on UDP, which is available in UDT [36]. This implementation has been shown by its authors to have similar performance to native TCP.

3.3.1 Is our model accurate?

We first verify the behavior of mPath with a series of experiments on Emulab, to show that mPath (i) can significantly improve throughput when there is sufficient core capacity and automatically distribute load over the paths according to the available path capacities, performs no worse than TCP when (ii) there is insufficient capacity in the core or (iii) the access link is the bottleneck, (iv) is TCP-friendly, and (v) can dynamically adapt to changing network conditions.

(i) Core Link Bottleneck, Excess Capacity Sufficient. Our first set of experiments was conducted on a topology containing a core link bottleneck as shown

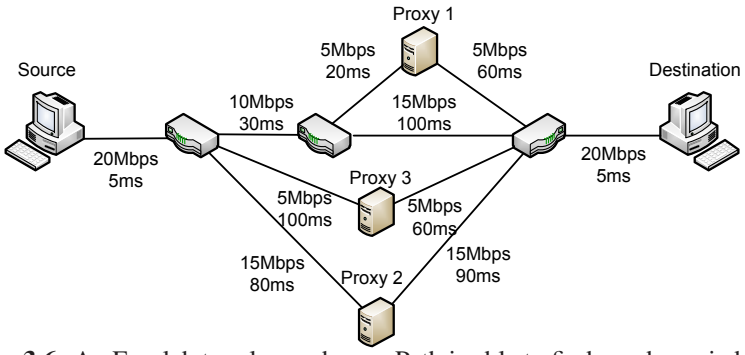


Figure 3.6: An Emulab topology where mPath is able to find good proxied paths.

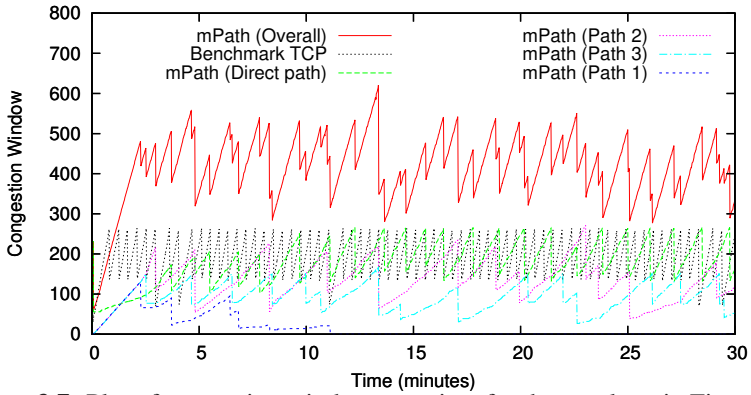


Figure 3.7: Plot of congestion window over time for the topology in Figure 3.6.

in Figure 3.6. In this topology, we created three proxied paths, with one proxy sharing a 10 Mbps core link bottleneck and all three proxies sharing a 20 Mbps access link bottleneck along the direct path. We ran mPath for 30 minutes and compared its performance to a TCP benchmark that was run for the same duration. As expected, mPath achieved an average throughput of 14.31 Mbps, and the benchmark TCP flow achieved an average throughput of 7.21 Mbps.

The congestion windows of the various paths (direct path and three proxied paths) used by mPath are shown in Figure 3.7. The congestion windows of the proxied paths are labeled according to the proxies that they pass through. We also plot the congestion window for the benchmark TCP flow and the overall mPath congestion window over time for reference.

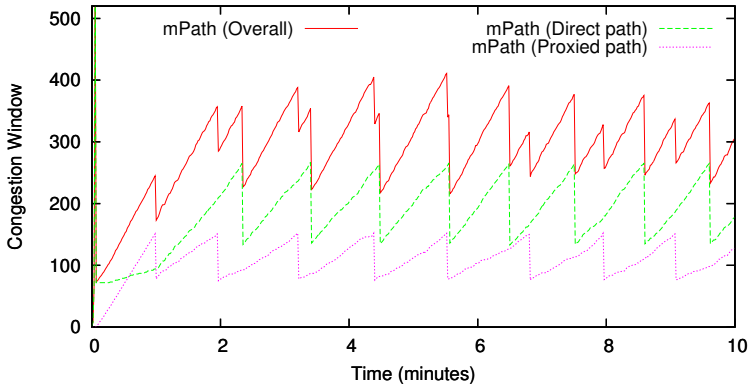


Figure 3.8: Plot of congestion window over time for the topology in Figure 3.6 when only proxy 3 is used.

Initially, the three proxied paths compete for bandwidth, with the congestion window of the direct path increasing slowly to allow those of the proxied paths to fully expand. In the process, mPath detects the shared bottleneck between path 1 and the direct path and applies load aggregation, causing path 1 to be dropped 11 minutes into the transfer when its congestion window is reduced to zero. This leaves the system in a stable state where paths 2 and 3 efficiently exploit the access link capacity that cannot be utilized by the direct path alone. Path 2 carries more traffic because it has a larger core link capacity. Observe that load aggregation also ensures that the direct path is fully utilized throughout the transfer, handling a load of about 7 Mbps.

(ii) Core Link Bottleneck, Insufficient Capacity. To investigate how mPath performs when there is insufficient core link capacity to saturate the access link, we use the topology shown in Figure 3.6 but with proxy 1 and 2 removed. That is, we have a 20 Mbps access link, a 10 Mbps core link bottleneck on the direct path and a 5 Mbps alternative path via proxy 3. Our results from running mPath and TCP individually for 10 minutes are shown in Figure 3.8. Clearly, mPath can still effectively utilize all available capacity on both direct and proxied paths.

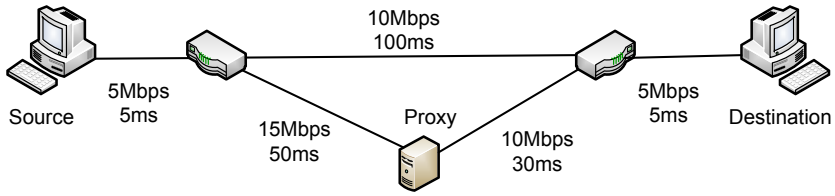


Figure 3.9: An Emulab topology where the access link is the bottleneck and the proxied path is useless.

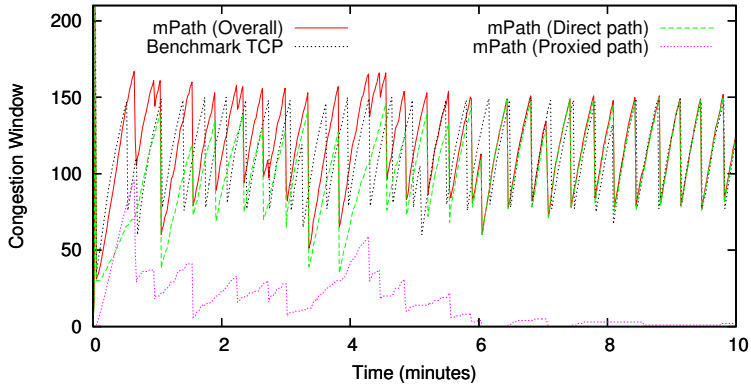


Figure 3.10: Plot of congestion window over time for the topology in Figure 3.9.

(iii) Access Link Bottleneck. For this scenario, we designed a simple topology where the access link is the only bottleneck in the system, as shown in Figure 3.9, and ran mPath and TCP individually for 10 minutes. The results in Figure 3.10 show that mPath and TCP produce similar patterns for their congestion windows and achieve about the same throughput: 4.36 Mbps for mPath and 4.40 Mbps for TCP. Some traffic is sent along the proxied path at first, but this drops significantly when mPath determines it is of no benefit and the proxied path's congestion window is aggregated to the direct path. This example verifies that when a good proxies are not available due to an access link bottleneck, mPath behaves much like TCP.

(iv) TCP-friendliness. We ran another experiment on the topology in Fig-

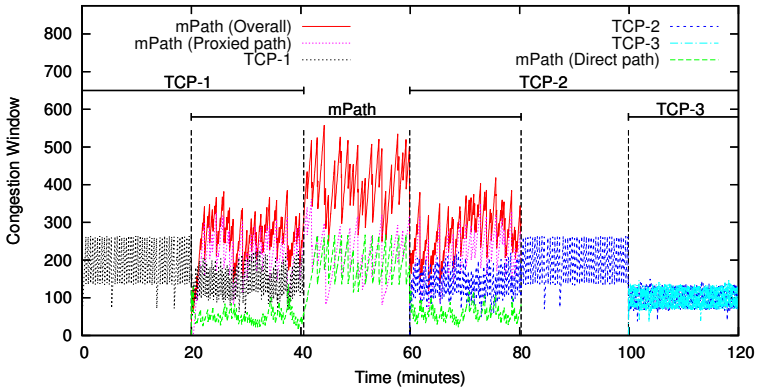


Figure 3.11: Plot of congestion window over time with competing mPath and TCP flows for the topology in Figure 3.6.

ure 3.6 to evaluate mPath in the presence of competing TCP flows. We started a TCP flow (TCP-1) in the background, after which we started mPath to observe its influence on TCP-1. Next, TCP-1 is terminated and another TCP flow (TCP-2) is started to observe how mPath reacts to the new flow. Finally, mPath is stopped completely and a third TCP flow (TCP-3) is started to provide us with a benchmark for two competing TCP flows. After starting/stopping a flow, we give the system 20 minutes to stabilize before making the next change. The congestion windows of the various flows over time are shown in Figure 3.11.

We found that running mPath in parallel with TCP causes the TCP congestion window to drop by about 25% on average, which is better than the TCP-3 benchmark which causes a 50% drop as expected. When TCP-1 is terminated, the direct path for mPath quickly soaks up all the excess bandwidth freed by the departure of TCP-1. When TCP-2 is started, it is able to achieve a steady state congestion window that is equivalent to that for TCP-1. The congestion windows of the proxied paths remain fairly stable throughout because they do not share the core link bottleneck with the direct path. These results show that, in the presence

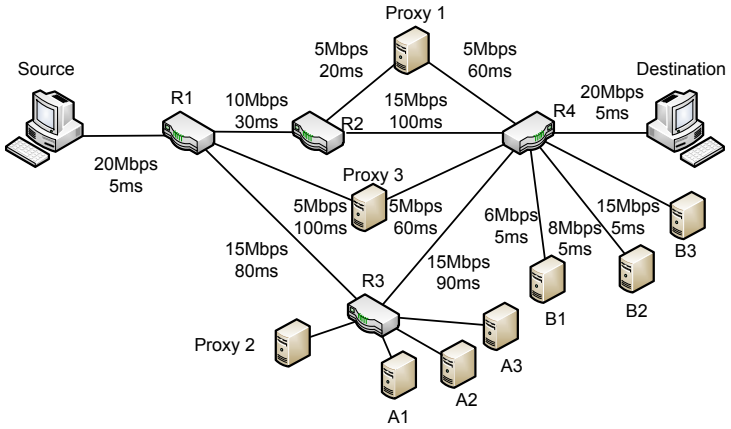


Figure 3.12: An Emulab topology to investigate how mPath reacts to changing path conditions.

of good proxied paths, mPath can achieve an overall throughput surpassing that of TCP while remaining TCP-friendly.

(v) Adapting to changing proxied path conditions. To show that mPath can dynamically adapt to congestion on the proxied paths, we created a new topology by adding some new nodes to the topology in Figure 3.6, as shown in Figure 3.12. Under this new scenario, path 2 of the mPath flow is disrupted by incrementally introducing three TCP flows that all use the path segment R3 to R4 to reduce the available capacity of the segment. The results are shown in Figure 3.13. The mPath flow is given 15 minutes to stabilize before we start the first TCP flow from A1 to B1, which has an access link capacity of 6 Mbps. In this state, proxied path 2 still has sufficient capacity to allow mPath to saturate the access link. After another 15 minutes, we add a second TCP flow with an access link capacity of 8 Mbps from A2 to B2. Now path 2 does not have sufficient capacity and mPath automatically redistributes some load to path 3. mPath's overall throughput in the steady state drops from 12.0 Mbps to 11.4 Mbps. When the final TCP flow from A3 to B3 is started, the segment R3 to R4 becomes highly congested and causes

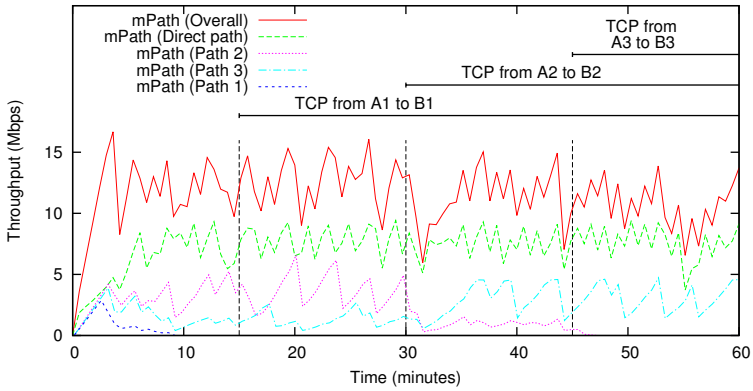


Figure 3.13: Plot of throughput over time with interfering TCP flows on proxied path 2 for the topology in Figure 3.12.

significant packet losses on path 2. This leads to path 2 being dropped completely within 2 minutes and an overall drop in the steady state throughput of mPath from 11.4 Mbps to 10.6 Mbps.

3.3.2 Does mPath work over the Internet?

To evaluate the performance of mPath over the Internet, we ran a series of experiments on PlanetLab using approximately 450 proxies. In this section, we present results from two representative experiments that demonstrate mPath (i) can improve throughput while maintaining TCP-friendliness when good proxied paths exist and (ii) performs no worse than TCP when a good proxied path cannot be found.

Performance with good proxied paths. The first experiment follows the same procedure as that of the Emulab experiment for TCP-friendliness described in Section 3.3.1. The only difference is that we now give mPath an additional 20 minutes to stabilize, since we now have significantly more proxied paths and it might take longer for good proxied paths to be found. The results from running

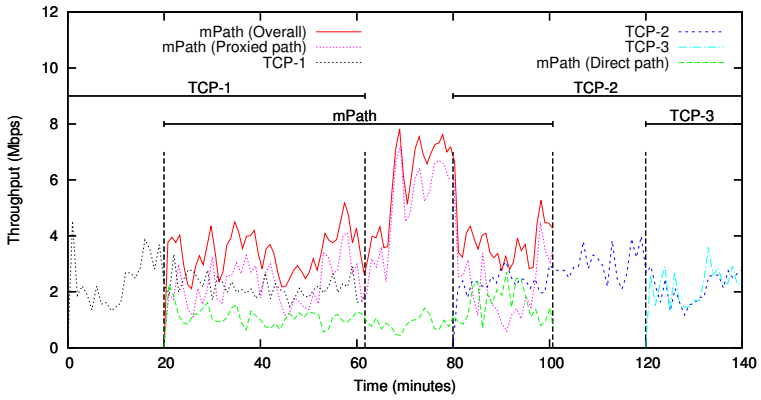


Figure 3.14: Plot of throughput against time for the path from `pads21.cs.nthu.edu.tw` to `planetlab1.cs.uit.no`.

this experiment on PlanetLab nodes `pads21.cs.nthu.edu.tw` and `planetlab1.cs.uit.no` are shown in Figure 3.14.

We observed similar behavior as that for the earlier corresponding Emulab experiment. In both cases, mPath achieves a relatively large increase in throughput. However, when TCP-1 terminates, the increase in throughput occurs on the proxied paths, rather than on the direct path as observed on Emulab. This is because the sender is given exclusive access to the topology in Emulab, while many users may have flows passing through the same core link bottleneck on the Internet for the PlanetLab experiment. Flows that pass through the same link will be given a share of the freed bandwidth when a flow leaves. In this case, stopping the flow of TCP-1 will only increase mPath's share of the direct path by a small amount. However, stopping TCP-1 also frees up bandwidth on the access link, which can then be used to increase the congestion windows of the proxied paths. Since the direct path is unable to supply enough bandwidth to fully utilize the access link, the proxied paths will take up most of the slack.

To better understand the improvement in throughput achieved by mPath, we

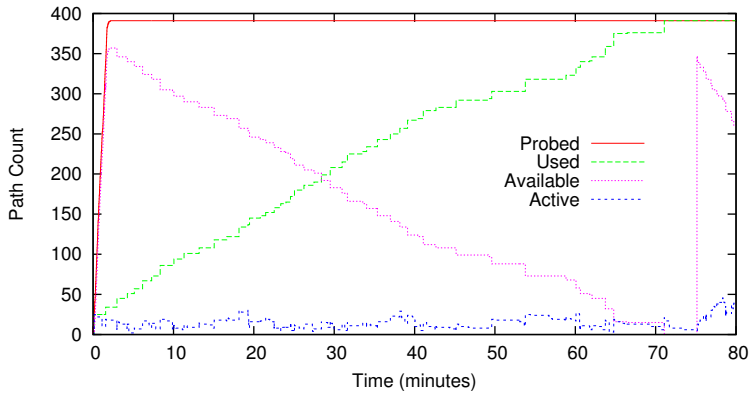


Figure 3.15: Plot of proxied path usage over time.

did *traceroutes* for the direct and proxied paths used in the experiment. The default route from `pads21.cs.nthu.edu.tw` to `planetlab1.cs.uit.no` used a direct path from `211.79.48.190` to `109.105.98.41`, which is a route across the Indian Ocean to Europe. We found that the proxied paths that contributed most to the throughput did not intersect with this route. In particular, most of the proxied paths used crossed the Pacific Ocean, continental America, and the Atlantic Ocean before reaching Europe. To some extent, this is not surprising because the Earth is round and there are generally two ways to connect any two points on the planet: clockwise and anti-clockwise.

We also examined how mPath finds and uses the proxies in the system to establish a stable set of proxied paths. Figure 3.15 is a plot of the distribution of the proxies over time. The probing phase to determine the available proxies completes relatively quickly and takes approximately 2 minutes to build an available list of approximately 400 proxies out of the 450 registered proxy nodes. mPath attempts to use all the available proxies in 75 minutes while maintaining an active set of between 10 to 20 proxies at any one time. Comparing these results

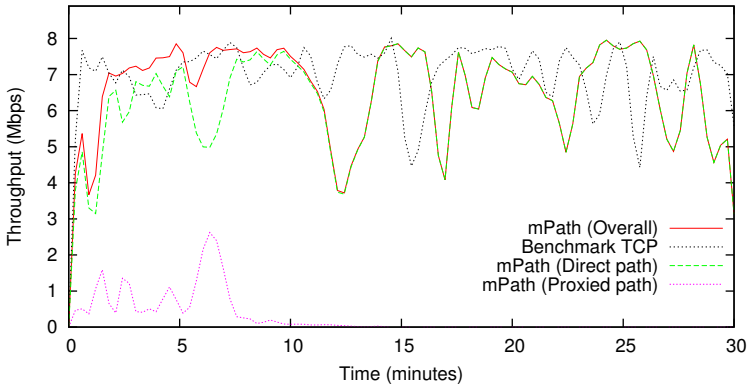


Figure 3.16: Plot of throughput against time for the path from `planetlab2.cs.ucla.edu` to `planetlab2.unl.edu`.

with the evolution of throughput in Figure 3.14, it is clear that the system finds a good working set of proxies long before it tries out all the available proxies. In fact, enough good proxied paths were found almost immediately after starting the transfer.

Performance without good proxied paths. In the second experiment, we used a pair of nodes (`planetlab2.cs.ucla.edu` and `planetlab2.unl.edu`) for which mPath failed to find any good proxied paths. The throughput achieved in this experiment is shown in Figure 3.16. For this pair of nodes, we found that all the proxied paths experienced a bottleneck at the same access link. We can see from Figure 3.16 that the throughput achieved by mPath and TCP are similar. mPath achieves its steady state throughput within 2 minutes and spends only about 12 minutes assessing the 450 available proxies before giving up.

3.3.3 How often and how well does mPath work?

We investigated the throughput achieved by mPath for approximately 500 source-destination pairs (distinct from the proxies) on PlanetLab and compared it to the

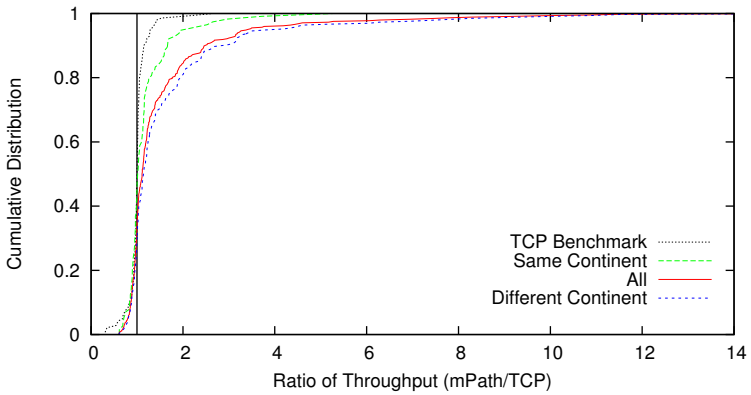


Figure 3.17: Cumulative distribution of the ratio of mPath throughput to TCP throughput for 500 source-destination pairs.

throughput achieved by TCP. Figure 3.17 shows the cumulative distribution of the ratio of the throughput achieved by mPath to that of TCP over all the node pairs tested. Each data point was obtained by running mPath for 30 minutes followed by TCP for another 30 minutes on each pair of nodes.

We found that mPath performs at most 20% worse than TCP for a small number of node pairs, which we believe can be attributed to the natural temporal variance of the available bandwidth on the Internet due to congestion and cross-traffic. To verify this, we ran a large number of TCP flows back-to-back for 30 minutes on random node pairs and plot the ratio between these two flows in Figure 3.17 as "TCP benchmark". The line provides us with a benchmark for what would be considered performance equivalent to TCP. In this light, we consider mPath to have achieved an improvement over TCP if it achieves a distribution that is to the right of this benchmark line. We see that about 40% of the node pairs seem to achieve an non-trivial improvement in throughput, with about half of these pairs achieving more than twice the throughput achieved by TCP. This is a significant proportion and it verifies our hypothesis that many of the bottlenecks for the di-

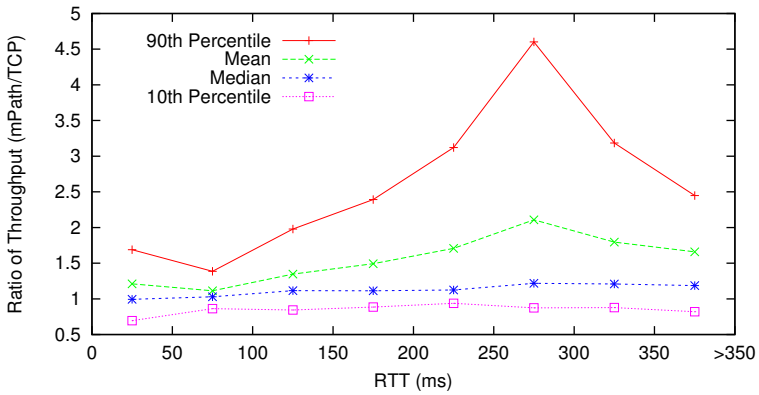


Figure 3.18: Plot of ratio of mPath throughput to TCP throughput against RTT.

rect paths are in the core, at least for PlanetLab nodes. From our observations, we found that the remaining 60% of node pairs could not improve their throughput using mPath, possibly because they were limited by their access links. Even for cases where mPath seems to perform more poorly than TCP, the distribution is still on the right of the TCP benchmark line, suggesting that even when network conditions deteriorate, mPath is likely able to ameliorate the degradation.

Intuitively, the distance between the sender and receiver would have a significant impact on how much mPath can improve the throughput. We expect that if the sender and receiver are very close (e.g. in the same AS), the throughput gains would only be marginal. This is evident in Figure 3.17, where we plot the improvement ratio of node pairs that have been categorized according to whether they are located on the same continent or on different continents. From these results, it is clear that the pairs located on different continents can achieve larger improvements and this conforms with our intuition. In addition, we plot the throughput improvement ratio against the direct RTT between the sender and receiver (for 500 node pairs) in Figure 3.18. As expected, node pairs with a higher RTT have a

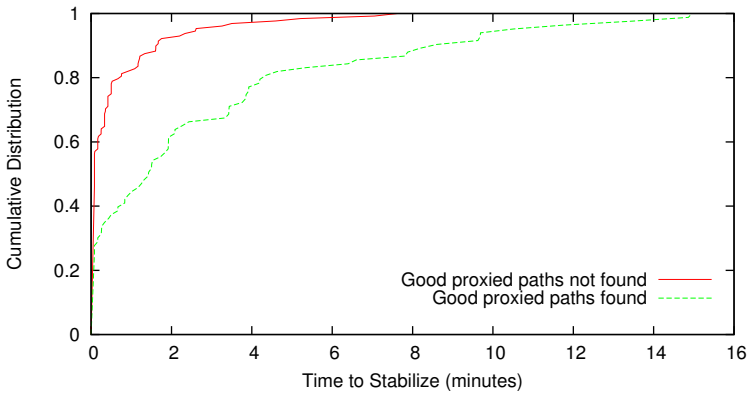


Figure 3.19: Cumulative distribution of the time taken for mPath to stabilize.

greater chance of benefiting from mPath and this suggests that mPath should use proxied paths more aggressively if the direct path RTT is larger, but this remains as future work.

To determine how quickly mPath can find a good set of proxies, we plot the time taken for mPath to reach its steady state throughput in Figure 3.19. We see that if good proxied paths exist, mPath can find them within 5 minutes for 80% of the node pairs. If good proxied paths cannot be found, mPath gives up within 1 or 2 minutes 90% of the time.

Based on these results, we see that the practicality of using mPath largely depends on the location of the bottleneck and the duration of transmission. If the bottleneck is at the core, and the transmission takes longer than mPath's stabilization time, we can expect the throughput improvements shown earlier. On the other hand, if the bottleneck is at the access link, or if the transmission duration is too short, throughput gains with mPath will be marginal (if any). This suggests that mPath is only suitable for high-volume data transfers.

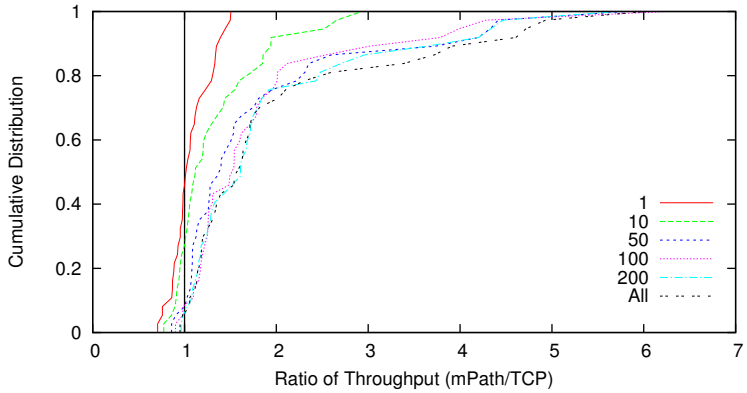


Figure 3.20: Cumulative distribution of the ratio of mPath throughput to TCP throughput when different numbers of proxies are provided by the RS.

3.3.4 How many proxies are minimally required?

We are also interested in the number of candidate proxies required for mPath to find a good proxy set. In this experiment, we limited the size of proxy lists returned by the RS and compared the throughput achieved by mPath to TCP. The sizes of proxy sets investigated are 1, 10, 50, 100, 200, and all the available proxies (approximately 450). As shown in Figure 3.20, the performance of mPath improves up till about 50 proxies, after which the performance gains of having more available proxies become negligible. This suggests that the RS should provide source nodes with at least 50 proxies. As these results are for a system where only one mPath flow is active, it is possible that more proxies will be required in practice.

3.3.5 Is mPath scalable?

Since mPath is expected to support a large number of users, we want to understand how the performance of mPath will scale as the number of users in the system

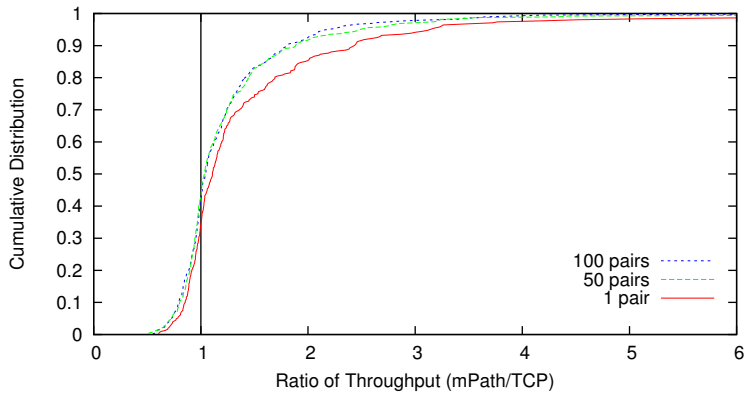


Figure 3.21: Cumulative distribution of mPath throughput to TCP throughput with n disjoint source-destination pairs transmitting simultaneously when proxies and end-hosts are distinct nodes.

increases. We used 200 PlanetLab nodes (distinct from the proxies), partitioned them into 100 disjoint pairs of senders and receivers, and ran experiments with 1, 50 and 100 pairs of nodes transmitting to each other simultaneously. By using four different random partitions, we obtained 400 data points for each of the three scenarios. These results are shown in Figure 3.21.

Since mPath is useful only if it improves the throughput for a node pair significantly, we focus on the proportion of node pairs for which mPath can achieve throughput that is at least twice that of TCP. We had shown earlier that when there is only one user, about 20% of the source-destination pairs can achieve twice the throughput of TCP. As shown in Figure 3.21, for 50 and 100 concurrent users, this number drops to about 10% of the users. This can be explained as follows: mPath consumes the unused bandwidth of proxies to improve throughput and users who are concurrently sending or receiving data would compete for this same bandwidth. If the amount of unused bandwidth is kept constant, then as the number of concurrent users increases, the number of users who see an improve-

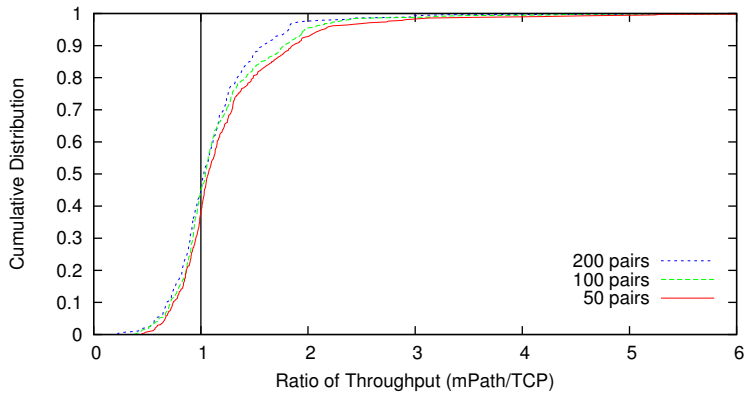


Figure 3.22: Cumulative distribution of mPath throughput to TCP throughput with n disjoint source-destination pairs transmitting simultaneously when the end-hosts are themselves proxies.

ment in throughput would decrease. Hence, the scalability of mPath depends on the amount of unused bandwidth available in the system, as expected. As we are limited by the PlanetLab nodes available, we are not able to conduct larger scale experiments to study this in greater detail.

Using client nodes as proxies is one potential way of improving the scalability of mPath. To evaluate the feasibility of this approach, we devised an experiment where 450 PlanetLab nodes are used as proxies and 50, 100 and 200 source-destination pairs are randomly selected from these proxies to concurrently send/receive data. As shown in Figure 3.22, mPath is still able to improve the throughput for some node pairs. For 50 and 100 pairs, about 10% of the pairs can achieve at least twice the throughput of TCP. This drops to a very small number for 200 pairs, but this is not entirely surprising as this means that some 400 out of the 450 proxies are sending/receiving data. Since these proxies do not have much unused bandwidth to be exploited, mPath is unlikely to be able to use them to improve throughput. Our results suggests that if the number of users concurrently sending/receiving data is less than 50% of the total number of proxies, it is

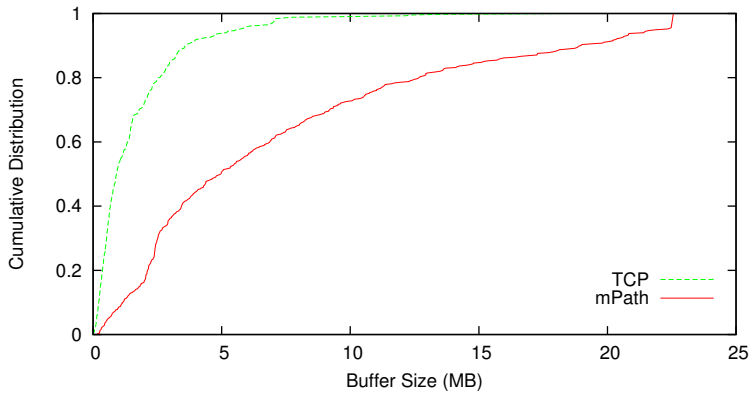


Figure 3.23: Cumulative distribution of the maximum buffer size required for 500 source-destination pairs.

feasible to use client nodes as proxies.

3.3.6 How serious is reordering in mPath?

The amount of reordering in the packets received directly affects the size of the buffer required at the receiver. In this light, we quantify the impact of reordering by recording the amount of buffer space used at the receiver (for both mPath and TCP) during our experiments. In Figure 3.23, we plot the buffer usage for all the node pairs evaluated in Section 3.3.3. From these results, we can see that a 25 MB buffer is sufficient to handle the reordering in mPath. Since most modern desktops have at least 4 GB of RAM, 25 MB is insignificant.

3.3.7 How should the parameters be tuned?

mPath is characterized by the parameters α, β and τ. In this section, we investigate the trade-off for each of these parameters. We investigate the effect of α on Emulab because of the controlled environment and the effect of β and τ on PlanetLab

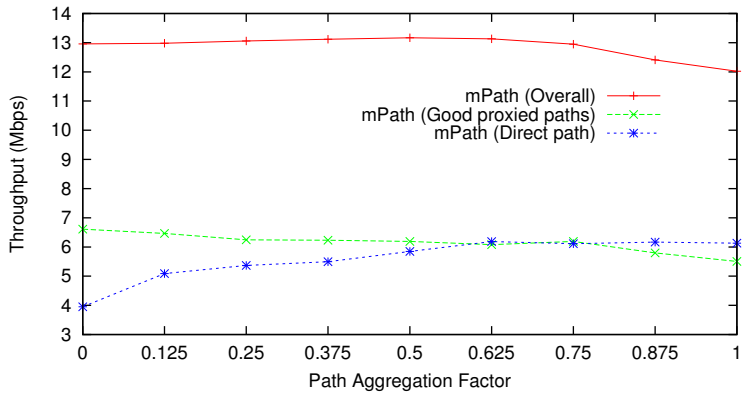


Figure 3.24: Plot of throughput against load aggregation factor α.

because it was not practical to create hundreds of proxies on Emulab.

Load Aggregation (α). The first parameter, α, is the proportion of the congestion window moved from a proxied path to the direct path when a correlated packet loss is detected. If α is too large, it may result in low utilization of good proxied paths and reduced throughput; if α is too small, it may take a long time for bad proxied paths to converge to zero and cause the direct path to be underutilized. As shown in Figure 3.24, our experiments indicate that $\alpha > 0.75$ would lead to a decrease in overall throughput. We also found that when $\alpha \leq 0.25$, mPath would take more than 30 minutes to eliminate the bad proxied paths. Thus, we set $\alpha = 0.5$.

Proxied Path Creation (β). Next, we investigate β, the number of new paths created as a fraction of the direct path congestion window when loss is detected. This number trades off the time taken by mPath to find a good proxy set against the utilization of the direct path. Using a pair of nodes observed to have good proxied paths (`planetlab6.goto.info.waseda.ac.jp` to `planetlab2.wiwi.hu-berlin.de`), we perform data transfers lasting 30 minutes and plot the through-

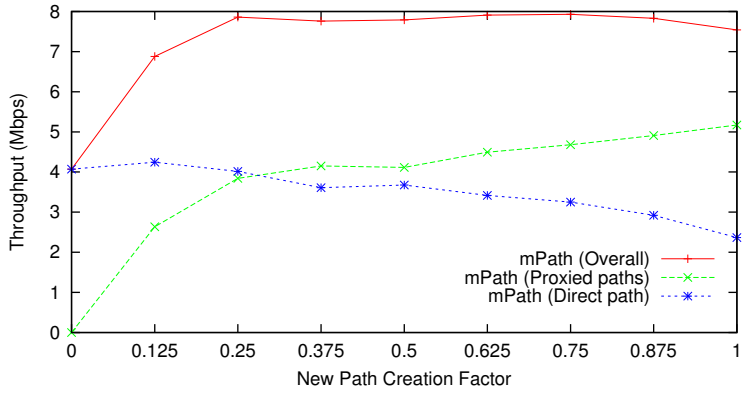


Figure 3.25: Plot of throughput against new path creation factor β.

put achieved against β in Figure 3.25. The graph shows that when $\beta \geq 0.5$, the utilization of the direct path decreases, and when $\beta > 0.75$, there is even a slight drop in the overall throughput. We also found that when $\beta < 0.25$, the time taken to find good proxies increases significantly, and that beyond $\beta = 0.25$, there is no substantial reduction in this time. Therefore, we set $\beta = 0.25$.

Maximum Allowable Proxied Path RTT (τ). Intuitively, the maximum allowable RTT for the proxied paths is directly related to packet reordering and the number of proxied paths that can be used, and these factors will affect the achieved throughput. Figure 3.26 shows the effect of τ on the maximum buffer size required at the receiver. Clearly, increasing τ results in greater reordering and thus larger buffering requirements. Figure 3.27 shows the number of usable proxied paths as we increase τ. We pick $\tau = 2 \times RTT_0$ because this provides a sufficient number of proxies and because we found that increasing it beyond this value did not yield significant improvements in throughput.

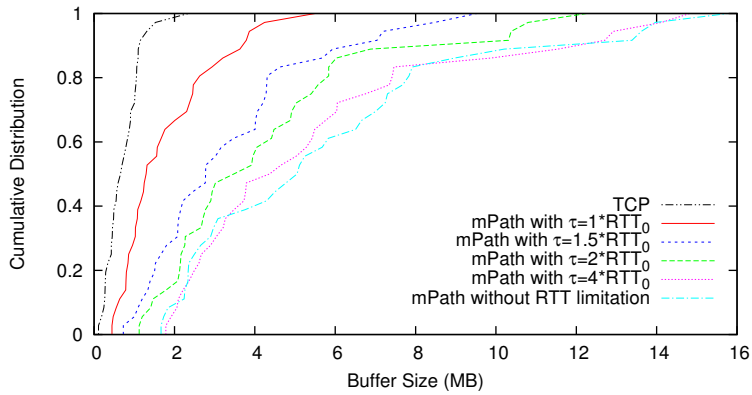


Figure 3.26: Cumulative distribution of the maximum buffer size required for different maximum proxied path RTTs τ.

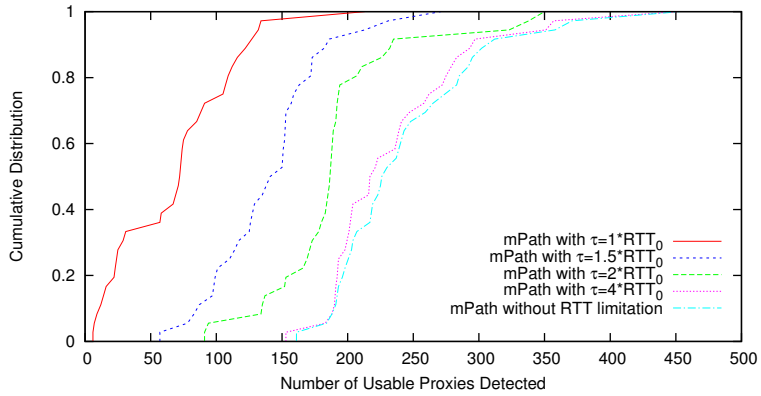


Figure 3.27: Cumulative distribution of the number of usable proxies detected for different maximum allowable proxied path RTTs τ.

3.4 Summary

In this chapter, we proposed and evaluated mPath, a practical massively-multipath source routing system, that (i) is TCP-friendly, (ii) will maximize the utilization of the access link without under-utilizing the direct path when there is free core link capacity, and (iii) will rapidly eliminate any redundant proxied paths. This is achieved with a modified AIMD congestion control algorithm that uses loss intervals to infer shared bottlenecks and incorporates a load aggregation mechanism

to maximize direct path usage. We show with evaluation that mPath is able to mitigate the problem of the core link congestion effectively. In the next chapter, we will discuss the characteristics of cellular data networks and in particular, we will discuss one of the access link congestion problem, namely "self-inflicted" congestion problem.

Chapter 4

Measurement Study of Cellular Data Networks

In recent years, cellular data networks are carrying an increasing amount of traffic with their ubiquitous deployments [1]. However, networks such as HSPA and LTE have very different link-layer protocols from conventional wired and WiFi networks. It is thus important to have a better understanding of the characteristics and behavior of cellular data networks.

In this chapter, we investigate and measure the characteristics of the cellular data networks for the three ISPs in Singapore with experiments in the laboratory as well as with crowd-sourced data from real mobile subscribers. We observed several interesting behaviors that are different from conventional wired and WiFi networks: i) transmitted packets tend to arrive in bursts; ii) there can be large variations in the instantaneous throughput over a short period of time; iii) large separate downlink buffers are typically deployed in mobile ISPs; and iv) mobile ISPs typically implement some form of fair queuing policy.

These findings confirm that cellular data networks behave differently from conventional wired and WiFi networks, which may be overlooked by the researchers. These results also suggest that more can be done to optimize protocol performance in existing cellular data networks. For example, the fair scheduling in such networks might effectively eliminate the need for congestion control if the cellular link is the bottleneck link. We have also found that different ISPs and even different devices use different buffer configurations and queuing policies. Whether these configurations are optimal and what makes a configuration optimal are candidates for further study.

We further investigate the performance issues when there is concurrent uploads and downloads in cellular data networks. Our measurement study shows that in the presence of a simultaneous background upload, 3G download speeds can be drastically reduced from more than 1,000 Kbps to less than 100 Kbps. We identify this problem as the "self-inflicted" congestion problem.

We organize this chapter as follows: in Section 4.1, we first describe our methodology of the measurement studies. In Section 4.2, we investigate the packet flow characteristics of cellular data networks, in particular, the bursty arrival pattern of the packets and the variation of throughput. In Section 4.3, we measure the buffer setting and queuing policy of cellular data networks. In Section 4.4, we investigate the performance issues caused by a saturated upstream buffer in cellular data networks.

4.1 Methodology

We first describe our measurement study methodology. Our experiments were conducted on the cellular data networks of the three local ISPs in Singapore, which we anonymize as A, B and C. Some measurements were taken in our laboratory at the National University of Singapore, while the rest were crowd-sourced with the assistance of real users using their personal mobile devices. For the laboratory experiments, we purchased 3G/LTE cellular data plans from each ISP and took measurements with different models of smartphones and USB modems. The LTE data plans were backward-compatible with the older HSPA and HSPA+ networks and allowed us to also access these older networks and use non-LTE-enabled mobile devices.

To obtain crowd-sourced measurements, we developed and published a measurement application, *ISPCheck* [2], on the Android Play Store. To date, it has about 50 installations and the data presented in this paper was obtained over a 5-month period from April to August 2013. During this period, 6,048 sets of experiments from 23 different users were collected, with 2,301 sets for HSPA networks and 3,747 sets for the faster HSPA+ networks. We did not include the data for LTE networks because we had relatively little data for these networks, since the LTE networks in Singapore are relatively new and the majority of subscribers have not yet upgraded to LTE.

In our experiments, the measured UDP throughput was never lower than the measured TCP throughput. This suggests that the local ISPs do not throttle UDP flows, unlike the ISPs for other countries[92]. As such, we decided to use UDP flow in all our experiments because using UDP provides us with full control over

the packet size and sending rate. Also, unless otherwise stated, the packet size for our experiments was 1,420 bytes (including IP headers), since we found that this was the default MTU negotiated by TCP connections in the local networks. For the experiments conducted in the laboratory, we synchronized the clock of the mobile phones to that of our server by pinging the phone over a USB connection with our server. By using pings with RTTs that are less than 2 ms, we were able to synchronize the clocks to within 1 ms accuracy. This allows us to count the packets in flight and determine the exact one-way delay in our measurements precisely. While `tcpdump` was used to log the packets in our laboratory experiments, we could not use it in ISPCheck because it requires root access to the device. So ISPCheck simply logs packet traces at the application layer.

4.2 Packet Flow Measurement

In this section, we investigate the packet flow characteristics of cellular data networks. In particular, we demonstrate that the arrival pattern of cellular data packets is bursty, and it is thus necessary to take this pattern into account when we try to estimate the instantaneous throughput for cellular data networks. Finally, we investigate how the instantaneous throughput of cellular data networks vary over time and find that it can vary by as much as two orders of magnitude within a 10-minute interval.

4.2.1 Burstiness of Packet Arrival

In cellular data networks, packets are typically segmented and transmitted over several frames in the network link and then reconstructed at the receiver. Such

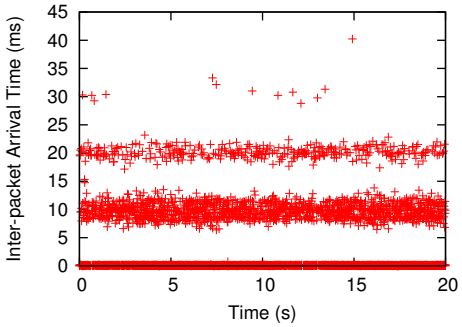


Figure 4.1: Trace of the inter-packet arrival times of a downstream UDP flow in ISP C.

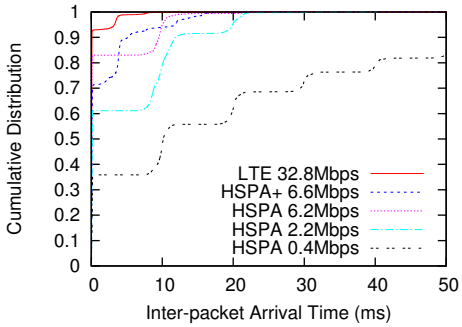
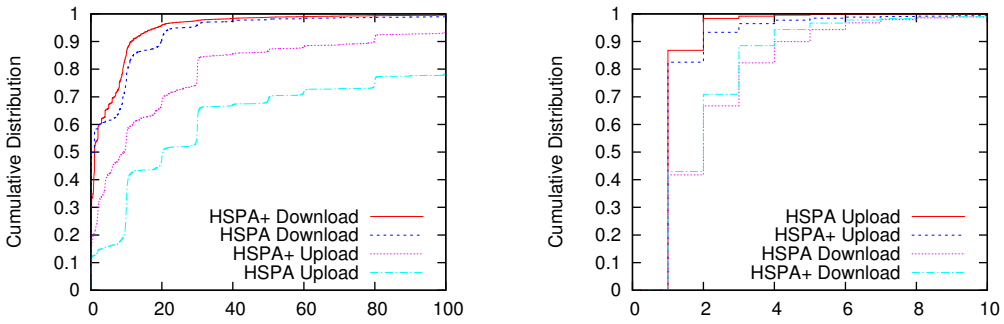


Figure 4.2: Cumulative distribution of the inter-packet arrival times for ISP C.

networks also incorporate an ARQ mechanism that automatically retransmit erroneous frames, and this can cause packets to be delayed or reordered. To investigate the effect of the link layer protocols on the reception pattern of IP packets, we saturated the mobile link by sending UDP packets from our server to a mobile device at a rate that is higher than the receiving rate. A HTC Desire (HSPA-only) phone was used to measure existing HSPA networks and a Samsung Galaxy S4 phone was used to measure existing HSPA+ and LTE networks. We cannot use the Galaxy S4 to measure HSPA networks because it would always connect to existing HSPA+ networks by default.

One key observation is that packets tend to arrive in bursts. In Figure 4.1, we plot the inter-packet arrival times of a representative trace from one of our experiments. We can clearly see that packets tend to arrive in clusters at 10 ms intervals, and that most packets tend to arrive within bursts of less than 1 ms interval. In Figure 4.2, we plot the cumulative distribution of the inter-packet arrival times for 5 traces from different networks and bandwidths. From these results, we can see that packet arrival is bursty at 10 ms intervals in HSPA networks and at 4 ms in the faster HSPA+ and LTE networks.



(a) Inter-packet arrival time (ms) **(b)** Number of packets in one burst

Figure 4.3: Inter-packet arrival times and number of packets in one burst for ISPCheck.

In Figure 4.3(a), we plot the cumulative distribution of the inter-packet arrival times for the crowd-sourced data collected with ISPCheck. In total, the ISPCheck data set consisted of more than 1 million downstream packets and over 400,000 upstream packets. Again, we can see that the packets arrive in distinct bands even when the packet traces are recorded at the application layer. We consider packets that arrive within 1 ms of each other to constitute a burst, and plot the cumulative distribution of burst sizes in Figure 4.3(b). We can see that the majority of downstream packets arrive in bursts. This is likely because the downlink of cellular data networks allows for the parallel transmission of frames which could result in multiple packets being reconstructed at the same time at the receiver.

The arrival of packets at distinct intervals of either 10 ms or 4 ms is likely due to the polling duty cycle of the radio driver in the mobile devices. We noticed that older (and slower) phones like the HTC Desire had a longer interval of 10 ms, while the newer Galaxy S4 has an interval of only 4 ms. To ascertain that this was independent of the kernel tick interval, we performed the same experiments over a 802.11g WiFi network, and we found that the arrival of packets was spread uniformly over time and there was no distinct banding of packets.

4.2.2 Measuring Instantaneous Throughput

Our observation of bursty packet arrivals suggests that traditional bandwidth measurement techniques involving packet pairs [56, 72] or packet trains [48, 79] will not work well for cellular data networks. In order to obtain a reasonably good estimate of the instantaneous throughput, we would likely have to observe at least two bursts worth of packets, but even that might not be sufficient because of the coarse granularity of the clock.

To investigate the effect of the bursty packet arrival on instantaneous throughput estimation, we initiated a large number of saturating downstream flows of UDP packets for 30 s over an extended period, until we found a trace where the flow seemed to be stable and the average throughput over the entire period was 6.9 Mbps. Since the maximum speed of this data plan was 7.2 Mbps, we know that this trace is one where there was very little interference from other users and network traffic. Hence, any variations could be attributed to the burstiness of the packet arrivals and the transmission medium.

The packet arrivals in the trace were segmented into bursts of packets all arriving within 1 ms of each other. As before, the interval between the bursts was approximately 10 ms. Next, we estimated the instantaneous throughput by using a consecutive number of n bursts. That is, we ignored the first burst and divided the data in the last $n-1$ bursts over the total time elapsed between the n bursts. We computed all possible windows of n-bursts in the flow and plot the standard deviation and error between the estimates and the long-term average throughput of 6.9 Mbps (normalized against 6.9 Mbps) in Figure 4.4 for the estimates obtained as n varies from 2 to 1,000.

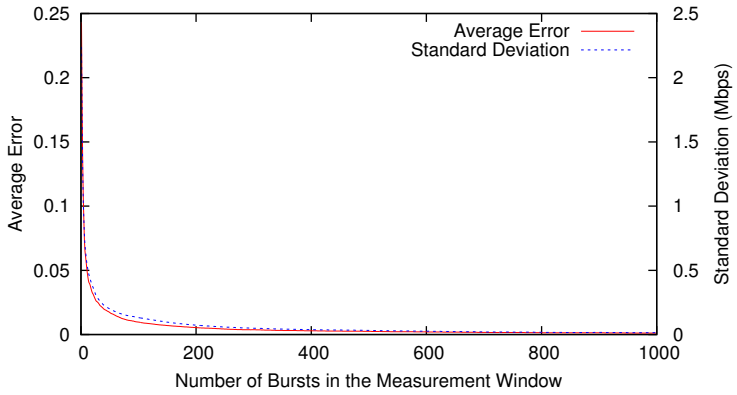


Figure 4.4: The accuracy of throughput estimation with different window.

As expected, the accuracy and the standard deviation of our estimates will improve if we use a larger number of bursts. However, it is not feasible to use too much data because doing so is not only costly, it might cause the measurement to take too long and the resulting instantaneous measurement might not be too meaningful. Our results in Figure 4.4 suggest that using 50 bursts of packets achieves a reasonable trade-off between accuracy and data required. This translates to about 100 KB and 300 KB of data respectively, or at least 400 ms and 325 ms respectively in terms of time, for measuring upstream and downstream throughput for 2 Mbps upstream/7.2 Mbps downstream HSPA networks.

4.2.3 Variations in Mobile Data Network Throughput

It is well-known that there is a large variance in the performance of cellular networks [66, 90]. In this section, we present our findings on the variations in the networks that we investigated.

In Figure 4.5, we plot the cumulative distribution of the crowd-sourced data obtained from ISPCheck. As expected, HSPA+ networks are generally faster than HSPA networks. While HSPA+ can in principle achieve speeds higher than

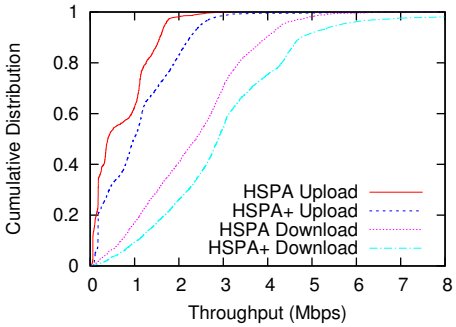


Figure 4.5: Plot of cumulative distribution of the throughput for data from ISPCheck.

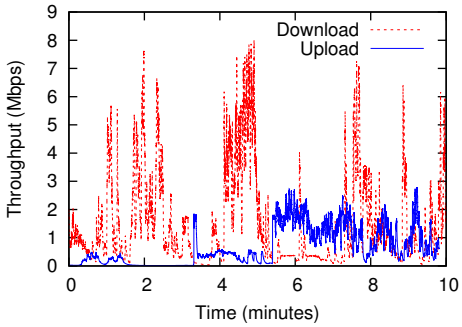


Figure 4.6: The huge variation of the download and upload throughput.

7.2 Mbps, we rarely found speeds higher than that because most of the local data plans have a maximum rate limit of 7.2 Mbps. Overall, we see significant asymmetry in the upstream and downstream data rates and also that the actual throughput achieved by the local subscribers can vary significantly from a few Kbps to several Mbps.

To understand temporary variation, we initiated a 10-minute long UDP flow in the HSPA+ network of ISP C and maintained a constant number of packets in flight to keep the buffer filled and ensure that the cellular link is always busy. We estimated the instantaneous throughput over the entire period using windows of 50-packet bursts, as discussed in Section 4.2.1. We plot the estimated instantaneous throughput for both an upstream flow and a downstream flow in Figure 4.6. We can see that not only does the throughput change fairly quickly, it also varies by as much as over two orders of magnitude several times within a 10-minute interval. This corroborates the claims of previous work [90, 96]. With such huge variations over such short periods, protocols like TCP and HTTP might not work well [46]. In fact, several unrelated experiments that we attempted with TCP on these cellular data networks also suffered from serious under-utilization and time-

outs.

4.3 Buffer and Queuing Policy

This section highlights our measurements of the buffer configurations on both ends of cellular data networks and our investigation into the queuing policies of the three local ISPs.

Downlink Buffer Size. We estimate the buffer size by sending UDP packets at a rate higher than the receiving rate, which causes the buffer to fill over time with packets and eventually overflow. We can accurately determine the number of the outstanding packets in the network, or the packets in flight, by synchronizing the clock of our mobile phones to that of the server. Finally, we can estimate the buffer size by subtracting the measured bandwidth-delay product from the total packets in flight. Interestingly, we found that instead of being conventionally sized in bytes, the downstream buffers at the ISPs are sized in packets. In these experiments, we vary the size of a packet from 200 to 1,420 bytes. We could not use packets smaller than 200 bytes because our receiving devices and `tcpdump` are not able to process smaller packets fast enough when we try to saturate the networks to measure the buffer size.

Figure 4.7 shows the plot of packets in flight against time for one of our experiments using different packet sizes over ISP C's HSPA network. We can see that the number of the packets in flight plateaus at the same value for different packet sizes. In this instance, the bandwidth delay product was small ($\approx$ 50 packets), and so we deduced that the buffer size was fixed at about 2,000 packets. We observed similar behavior in the downstream buffers for all the networks studied, with the

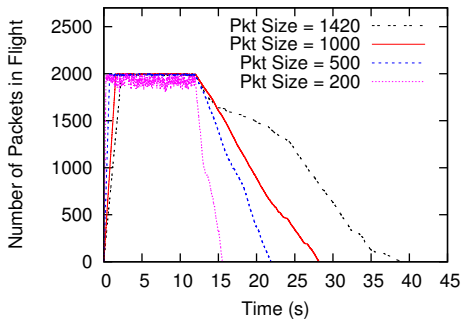


Figure 4.7: The number of packets in flight for downloads with different packet size.

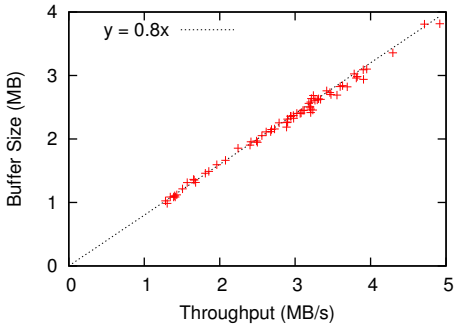


Figure 4.8: In ISP A's LTE network, the buffer size seems to be proportional to the throughput.

exception of ISP A's LTE network.

We found that the downstream buffer for ISP A's LTE network behaved quite differently from the rest. As shown in Figure 4.8, the buffer size seems to be a linear function of the throughput (c.f. $y = 0.8x$). In other words, the size of the buffer appears to vary proportionally to the throughput in a way that keeps the maximum queuing delay constant at 800 ms. We suspect that ISP A might have implemented a Codel-like [71] mechanism in their network, i.e., packets are timestamped when they arrive, and checked at the head of the queue. If they spent longer than 800 ms in the buffer, they would be dropped. While there is certainly an absolute limit of the buffer in terms of physical memory space, we were not able to exceed that even when we send packets at the maximum supported data rate. A summary of the estimated buffer sizes for all three local ISPs is shown in Table 4.1.

Overall, we observed that the downstream buffers for most of the ISP networks are fairly large. Because the variation in the throughput can be very large, it is possible on occasion for the latency to become very high when throughput is too low to drain the buffer fast enough [51]. By controlling the maximum time that a

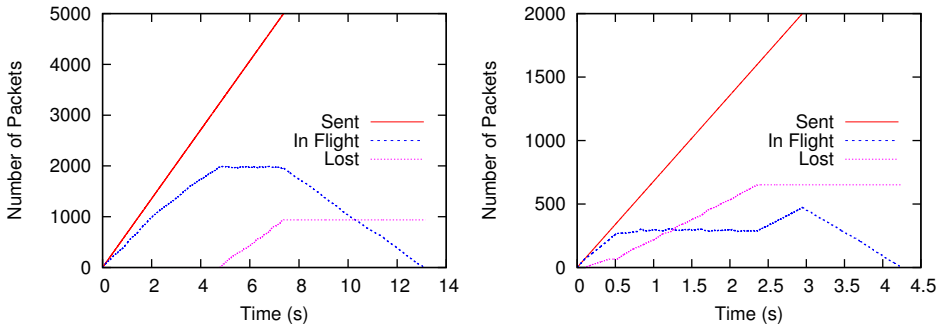
Table 4.1: Downlink buffer characteristics for local ISPs

ISP	Network	Buffer Size	Drop Policy
ISP A	HSPA(+)	4,000 pkts	Drop-tail
	LTE	800 ms	AQM
ISP B	HSPA(+)	400 pkts	Drop-head
	LTE	600 pkts	Drop-tail
ISP C	HSPA(+)	2,000 pkts	Drop-tail
	LTE	2,000 pkts	Drop-tail

packet can spend in the buffer(like in ISP A's LTE network), the maximum latency can however be kept at a consistent and stable value (about 800 ms for ISP A's LTE network) independent of the throughput.

Drop Policy. In addition to the buffer size, we also investigated the drop policy of the various ISPs by studying the traces of the packet losses and found that a drop-tail policy was implemented in all the networks except for ISP B's HSPA(+) network and ISP A's LTE network. We repeated our experiments several times with different parameter settings and at different physical locations, and consistently obtained similar results, which are summarized in Table 4.1.

We explain how we inferred the drop policies with the following examples: in Figure 4.9(a), we plot the number of packets sent, packets lost and packets in flight over time for ISP C's HSPA(+) network, and in Figure 4.9(b), we plot a corresponding trace for ISP B's HSPA(+) network. Because the traces are analyzed offline, we could determine the lost packets by observing that they were sent but never received. However, we cannot determine precisely when the packet losses happened. Hence, the "Lost" line in our graphs refers to the time when the lost packets were sent and not when they were actually dropped. We see in Figure 4.9(a), that for ISP B's network, packet losses only occur to packets sent



(a) ISP C HSPA(+) **(b)** ISP B HSPA(+)

Figure 4.9: Trace of the packets sent, lost and in flight in a UDP downstream flow.

after time $t = 5$. This also coincides with the start of a plateau in the number of packets in flight because we exclude known lost packets when plotting the number of packets in flight. Thus, we can infer that Figure 4.9(a) suggests a drop-tail queue, where the buffer is fully saturated around time $t = 5$ and newly sent packets are dropped until no more packets are sent at time $t = 7.2$ and the buffer starts to empty.

In contrast, Figure 4.9(b) paints a very different picture for ISP C's network. We see that packet losses start to occur very early in the trace and stop after time $t = 2.4$, i.e., there were no losses for the final batch of 400 packets sent after time $t = 2.4$. This suggests a drop-head queuing policy. In addition, the packets in flight plateaus at a lower value before increasing to a peak from time $t = 2.4$ to $t = 3$. The likely explanation for this phenomenon is that we exclude lost packets from the packets in flight by assuming they were lost the moment they were sent. However, for a drop-head queue, these lost packets would have occupied some space in the buffer before they get dropped at the head of the queue. Thus, as long as packets are being dropped in a drop-head queue, our estimate of the packets in flight is likely an underestimate of the actual value. From time $t = 2.4$ to $t =$

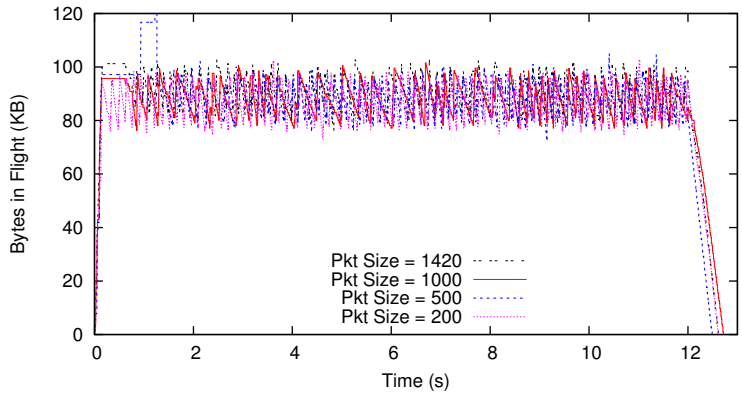


Figure 4.10: The bytes in flight for uploads with different packet sizes.

3, the older packets in the buffer are still being dropped but no new packet are lost. Hence, the proportion of packets dropped decreases, which explains why our estimate of the packets in flight continues to increase.

Uplink Buffer Size. The uplink buffer is at the radio interface of the mobile device, and for all the mobile phones we tested, the buffer is sized in terms of bytes rather than number of packets like the downlink buffer. In Figure 4.10, we plot the bytes in flight over time for the experiments carried out on a HTC Desire phone. We see that the number of bytes in flight remained constant for different packet sizes. On the other hand, the Huawei USB modems we tested had a buffer that is sized in terms of number of packets. Our results are summarized in Table 4.2.

Another interesting finding is that the newer Samsung Galaxy S3 LTE and Galaxy S4 phones seem to buffer packets in the kernel (which is sized in packets), in addition to the regular buffer in the radio interface (which is sized in bytes). Our ISPCheck application is blocked from sending UDP packets once there are about 200 packets in the kernel buffer. This behavior is unexpected because we do not typically expect UDP packet transmissions to be blocked and indeed, this is not observed in the older Android phones. It is plausible that the phone manu-

Table 4.2: The radio interface buffer size of different devices

Device Type	Model	Network	Buffer Size
Android Phone	HTC Desire	HSPA	64 KB
	Galaxy Nexus	HSPA+	1.5 MB
	Galaxy S3 LTE†	HSPA+	200 KB
		LTE	400 KB
	Galaxy S4†	HSPA+	200 KB
		LTE	400 KB
USB Modem	Huawei E3131	HSPA+	300 pkts
	Huawei E3276	LTE	1,000 pkts

†These devices have additional buffering of 1,000 packets in the kernel.

facturers have come to realize that because the uplink bandwidth can sometimes be very low, not blocking UDP transmissions would likely cause packets to be dropped even before the phone can get a chance to transmit them, and thus have modified the kernel to implement blocking even for UDP transmissions. To further investigate this phenomenon, we tethered the phone to a desktop computer via USB and used the desktop as the packet source, instead of an Android application. By running `tcpdump` on the USB and the radio interfaces of the phone, we can directly observe the flow of packets through the phone. In these experiments, we found that the buffering in the kernel was 1,000 packets for both the Galaxy S3 LTE and S4. There was no evidence that packets were buffered in the kernel for the other Android phone models that we investigated.

Separate Downlink Buffers. Winstein et al. claimed that ISPs implement a separate downlink buffer for each device in a cellular data network [96]. To verify this claim, we performed an experiment where we started saturating UDP flows to two mobile phones concurrently connected to the same radio cell. If there was a common buffer, we will likely see differences as the packets for the two flows

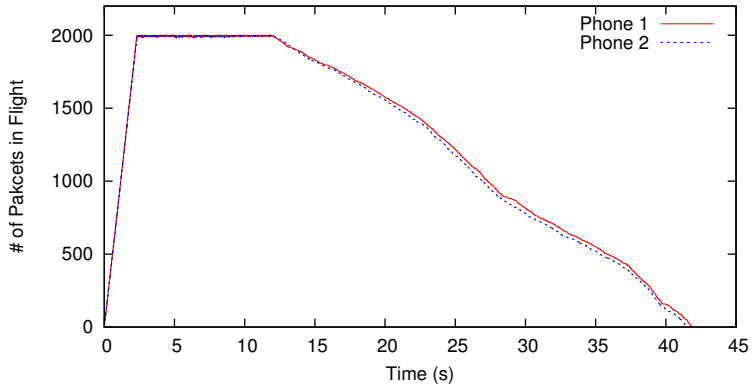


Figure 4.11: The number of packets in flight for two concurrent downloads.

jostle for a place in the common buffer. Instead, in Figure 4.11, we can see that the packets in flight reaches the same and constant value for both phones, indicating that the buffer is not shared between the devices. We observed the same behavior for all the three ISPs.

Queuing Policy and Fairness. To investigate if the ISPs implement a fair scheduling algorithm such as Round Robin, Maximum C/I and Proportional Fair as specified in [91], we ran the following experiment: using two mobile phones connected to the same cell with the same signal strength, we sent a UDP flow to one of the phones at the constant rate of one 50-byte packet every 10 ms. After 2 minutes, we started a saturating UDP flow to the other phone using 1,420-byte packets and saturated the buffer by maintaining 1,000 packets in flight. The first flow mimics a low-throughput, delay-sensitive application, while the second mimics a high-throughput application. In Figure 4.12, we plot the downstream one-way delay of both flows together with the throughput of the second saturating UDP flow. If the queuing policy were FIFO, we would expect that since flow 2 saturates the buffer, the one-way delay for flow 1 would greatly increase. Instead, our results show that the delay of flow 1 remains low and stable throughout.

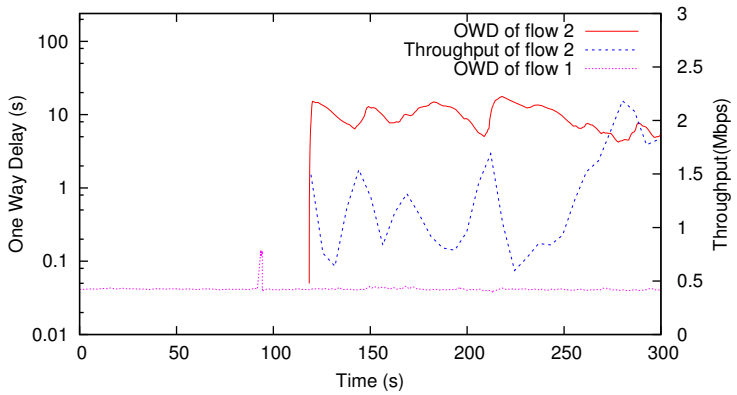


Figure 4.12: Comparison of delay-sensitive flow and high-throughput flow.

To investigate if the scheduling policy is fair among devices, we designed another experiment using three HTC Desire mobile phones connected to the same cell with similar signal strength. A downstream flow was initiated to each phone: i) a UDP flow that maintains 1,420 KB of data in flight, ii) a UDP flow that maintains 64 KB of data in flight, and iii) a TCP flow whose maximum receiver window is set at 64 KB. In Figure 4.13, we plot the throughput of all three flows with the number of packets in flight. It turns out that the throughput is fairly distributed among the three devices, independent of the number of packets or bytes of data in their buffer. We repeated this experiment for the HSPA(+) networks of all three local ISPs and found similar results.

We make several observations from the results of our experiments. First, all the ISPs clearly implement some form of fair queuing and unlike in the core Internet, UDP and TCP traffic seem to be treated equally by our local mobile ISPs. Second, having more data in flight may not help increase throughput because flows are effectively separated and do not compete for the same buffer space at a cellular base station. Instead, if the throughput is low, saturating the buffer will only result in increased latency. Third, since the fairness among connected mobile devices

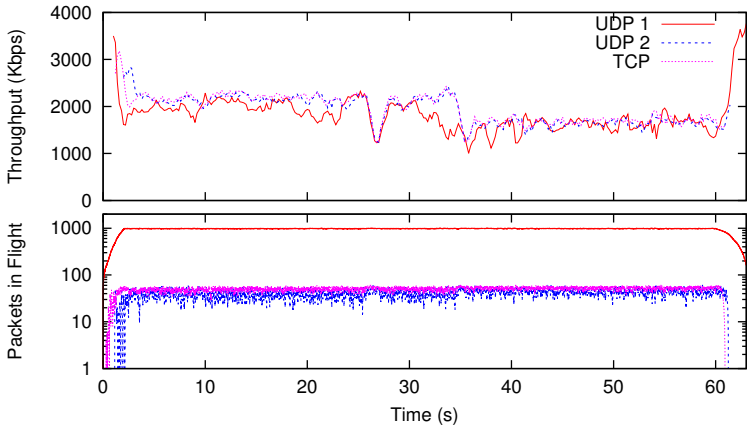


Figure 4.13: The throughput and packets in flight of three downlink flows in ISP C.

is enforced by a scheduling policy, congestion control at the transport layer (i.e. TCP) may not be necessary across a cellular link. This suggests that if the cellular link is the bottleneck link, which is common in the older HSPA networks, an end-to-end approach to congestion control may be possible [96]. Also, it is possible for an end-to-end flow to be split at the gateway of the cellular data network and a more efficient protocol can be used on the cellular link [99, 65].

4.4 The Problem of Saturated Uplink

As shown in previous sections, the throughput of 3G network can vary by as much as two orders of magnitude from a few Kbps to several Mbps and the buffer size can be set to quite large. Inspired by the *Data Pendulum* effect [42], we are interested in how these two properties will influence the performance of TCP when there is concurrent uploads and downloads. In this section, we demonstrate and explain why a concurrent upload can cause significant degradation to the download throughput with laboratory experiments. We focus on the 3G network in this section because the 4G/LTE network in our country is still not widely used and the

network is relative free and not yet congested. We believe our results can apply to the 4G/LTE network when the number of 4G/LTE subscribers increases and the congestion happens in the network.

Impact of Saturated Uplink. To verify that a saturated uplink can negatively impact downstream performance, we ran an experiment with three independent sets of TCP flows: (i) a download-only flow (d_0) that downloads 1 MB of data from a server, (ii) an upload-only flow (u_0) that sends 1 MB of data to the server, and (iii) an upload flow (u_1) that continuously sends random data to the server while a concurrent download flow (d_1) downloads 1 MB of data from the server. In a single run, these three sets of flows are run immediately one after another to minimize temporal variance. This experiment was conducted continuously over several days at 15-minute interval.

In Figure 4.14, we plot the performance of d_1 against d_0 and observe that a saturated uplink can significantly increase RTTs and reduce the downlink utilization. However, this phenomenon is not observed for all the experiments. In particular, ISP B experiences very little degradation in performance even when the uplink is saturated. We further investigated the relationship between the upload throughput and the downlink performance degradation by plotting the performance ratio, d_1/d_0, against the throughput of u_0 in Figure 4.15. We observed that the downstream performance is particularly poor when the upload speeds are low. For instance, ISP A consistently has low upload throughput and hence the downstream performance is severely degraded. We show later in this section that this poor downlink utilization can partially be explained by ACKs being delayed at the uplink buffer.

We found it interesting that even when an ISP has a consistently high up-

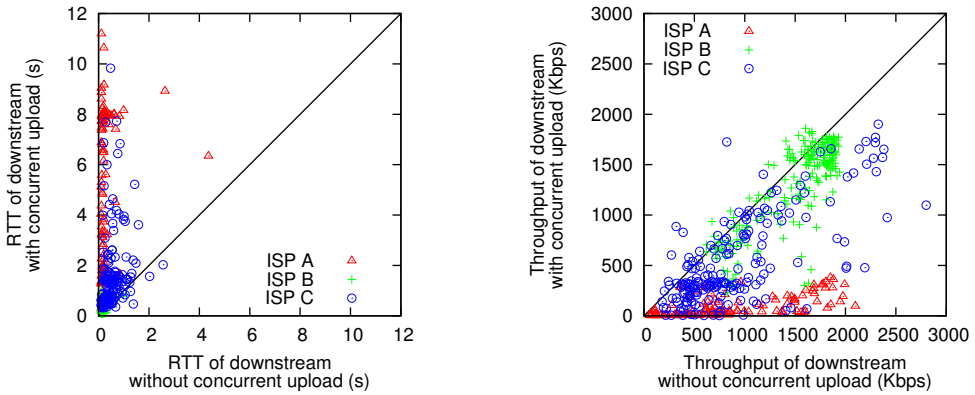


Figure 4.14: Comparison of RTT and throughput for downloads with and without uplink saturation.

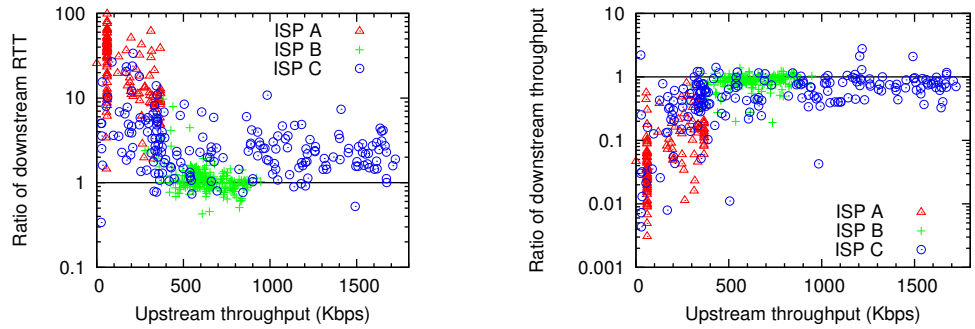


Figure 4.15: Plot of ratio of downstream RTT and throughput, with and without upload saturation, against the upload throughput.

load speed, as in the case of ISP B, the downlink utilization is still reduced by a small amount. We investigated this further by repeating our experiments with UDP flows. Even with UDP, the throughput of a download flow is degraded by a concurrent upload. Since there are no ACKs for UDP flows, this suggests that mobile downloads are naturally degraded by a concurrent upload, possibly because of ISP-level air time scheduling. As this phenomenon is beyond our control, the focus of this paper is on mitigating the problem of ACK delays, which clearly significantly exacerbates the degradation when upload speeds are low.

Measuring One-Way Delays. To confirm our hypothesis that delayed ACKs is a major reason for the poor downlink utilization, we set up an experiment in a

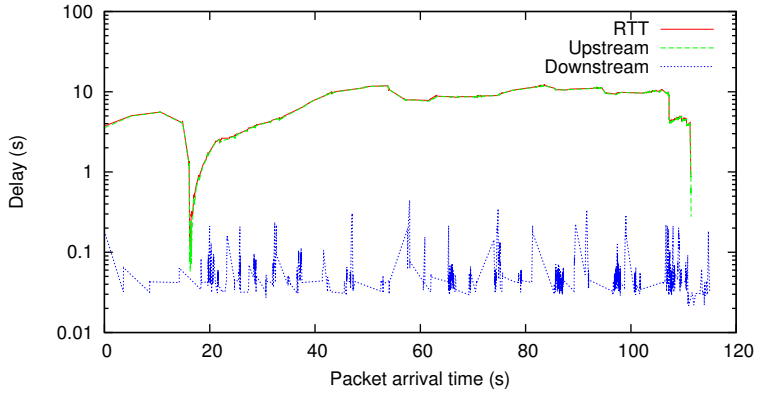


Figure 4.16: The breakdown of the downstream RTT into the one-way upstream delay and the one-way downstream delay.

loopback configuration. In this configuration, an Android phone was tethered to a server machine via USB. Next, upload and download TCP flows were initiated on the server and these flows were routed through the phone's 3G link via the USB connection and back to the server via the wired Internet. As the server is both the source and destination of all the TCP packets, the timestamps are fully synchronized and we can measure the one-way delay of the downlink (for data packets from the server to the phone), and the one-way delay of the uplink (for ACK packets from the phone to the server). In Figure 4.16, we plot the results of this experiment conducted on ISP A. We found that there is significant asymmetry in the delays and that the uplink one-way delay can be up to two orders of magnitude larger than the downlink one-way delay.

The reason for such a huge uplink one-way delay is that the uplink buffer is too large for the current uplink speed and it may take several seconds for the 3G link to transmit all the data packets in the buffer. For example, it takes around 10 seconds to send 64 KB (which is the uplink buffer size of HTC desire as shown in section 4.3) when the current uplink speed is 50 Kbps. In this sense, we propose

a method to regulate the uplink buffer in next section.

4.5 Summary

In this chapter, we described a measurement study of cellular data networks in Singapore, and highlighted several interesting characteristics. We showed that the packet arrivals tend to be bursty and that this burstiness needs to be taken into account when estimating instantaneous throughput. We verified that the throughput of existing cellular data networks can vary by as much as two orders of magnitude within a 10-min interval, and found that mobile ISPs often maintain large and separate downlink buffers for each user. The ISPs also implement some form of fair queuing, but for different networks, the buffer management policies may be quite different. We further investigated the performance issues when there are concurrent uploads and downloads in cellular data networks, which we identify as a "self-inflicted" congestion problem. In the next chapter, we propose a receiver-side flow control algorithm to address this "self-inflicted" congestion problem.

Chapter 5

Receiver-Side Flow Control

In this chapter, we propose and evaluate *Receiver-side Flow Control* (or RSFC), a method to dynamically control the uplink buffer of the sender from the receiver, to solve the "self-inflicted" congestion problem caused by saturated uplink buffer. The technique of using `rwnd` to control a TCP flow has been employed in other contexts [35, 87, 57, 54, 12, 24, 51]. However, to the best of our knowledge, we are the first to apply this technique to improve the utilization of a 3G mobile downlink in the presence of concurrent uploads.

The key challenge is for the TCP receiver to accurately estimate the current uplink capacity and to determine the appropriate `rwnd` to be advertised so that the number of packets in the uplink buffer is kept small without causing the uplink to become under-utilized. To solve this challenge, our approach uses the TCP timestamp to continuously estimate the one-way delay, queuing delay and RTT. Then, our approach uses a feedback loop to continuously estimate the available uplink bandwidth and advertises an appropriate TCP receiver window (`rwnd`) according to the current congestion state. This approach can dynamically adapt to

the variations of the cellular link.

We organize this chapter as follows: in Section 5.1, we first describe RSFC and explain how it can dynamically control the size of the uplink buffer of a mobile device. In Section 5.2, we evaluate the effectiveness of RSFC with experiments on three local cellular data networks.

5.1 Receiver-Side Flow Control

In Section 4.4, we showed that the downstream performance can be severely degraded at a mobile sender when the uplink bandwidth is low and the ACKs for the downstream data are delayed in the uplink buffer. One straightforward way to eliminate the delay would be to use a small uplink buffer. However, doing so will cause the uplink to be under-utilized when the available uplink bandwidth increases at a later time. Another possible approach is to control the total amount of data in flight so that the number of packets in the uplink buffer is large enough to fully utilize the uplink capacity and yet not so large as to inflate the uplink one-way delay unnecessarily. In fact, TCP Vegas [22] works in a similar manner by using packet delays to detect impending congestion (arising from packets building up in a buffer) in order to regulate the packet send rate. However, TCP Vegas does not contend well with the common default TCP implementations and it is thus not likely to be adopted as the default TCP implementation for mobile devices anytime soon.

In our approach, which we call *Receiver-side Flow Control* (RSFC), we regulate the number of packets in the uplink buffer of a mobile sender at the receiver by dynamically adjusting the advertised TCP receiver window (`rwnd`). The key

challenge is for the TCP receiver to accurately estimate the current uplink capacity and to determine the appropriate `rwnd` to be advertised so that the number of packets in the uplink buffer is kept small without causing the uplink to become under-utilized.

Similar to previous work [61], we estimate delays using TCP timestamps, i.e. using the `TSval` and `TSecr` fields in a received TCP packet. Our approach relies only on the relative differences in the received timestamps and does not require synchronization between the TCP sender and receiver. One caveat is that the granularity of TCP timestamps is not standardized and different devices may increment the timestamps at different rates. In this paper, we work with a granularity of 10 ms on both the mobile sender and the receiver. We expect this issue to be less of a concern in the future because an IETF working group is currently working on the standardization of the TCP timestamp granularity [82].

In the following sections, we first describe how our algorithm works in the general case and then we analyze how the uplink buffer at the mobile sender behaves. Next, we explain how RSFC adapts to changes in the underlying delays of the network. Finally, we briefly discuss how our algorithm can be deployed in practice.

5.1.1 RSFC Algorithm

A typical sequence of packets in an upstream TCP flow is illustrated in Figure 5.1. At time $t = t_{s_0}$, a packet is sent and it is received at time $t = t_{r_1}$ after a transmission delay t_u. Note that t_s and t_r are recorded with respect to the sender's and receiver's local clocks respectively. The receiver then sends an ACK packet with timestamp

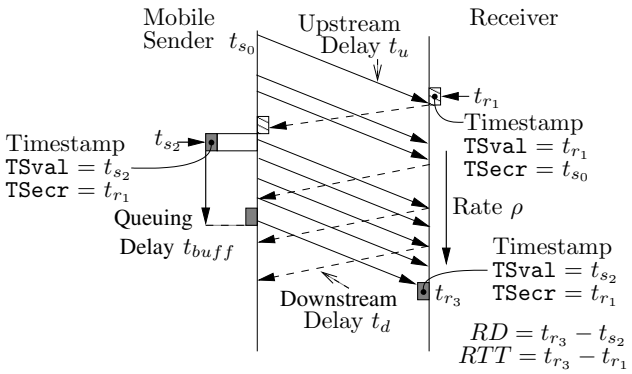


Figure 5.1: Packet flow diagram illustrating the various metrics. Solid lines represent data packets, while dotted lines represent ACK packets.

fields $\texttt{TSecr} = t_{s_0}$ and $\texttt{TSval} = t_{r_1}$. The sender receives the ACK packet after the transmission delay t_d at time $t = t_{s_2}$ and immediately pushes a new data packet into the buffer, with $\texttt{TSecr} = t_{r_1}$ and $\texttt{TSval} = t_{s_2}$. After spending some time t_{buff} in the uplink buffer, the packet is sent and after the transmission delay t_u, the packet is received at time $t = t_{r_3}$. Clearly, there is also a buffer at the downlink, but we do not consider it in our model because as shown in Figure 4.16, the downlink one-way delay is negligible and so the amount of time that a packet will spend in the downlink buffer is also negligible.

The receiver tracks the following metrics for each mobile sender as soon as packets are received: (i) the relative one-way delay ($RD = t_{r_3} - t_{s_2}$), (ii) the round-trip time ($RTT = t_{r_3} - t_{r_1}$), and (iii) the rate at which packets are received (ρ). RD and RTT will vary for different packets and the smallest RD and RTT observed are recorded as RD_{min} and RTT_{min} respectively. Clearly,

$$RD = t_{buff} + t_u + t_{offset} \tag{5.1}$$

$$RTT = t_{buff} + t_u + t_d \tag{5.2}$$

where t_{offset} is the clock offset between the sender's and the receiver's clocks. If the transmission delays t_u and t_d are constant, it is reasonable to assume that RD_{min} and RTT_{min} are obtained when $t_{buff} \approx 0$, i.e.

$$RD_{min} \approx t_u + t_{offset} \tag{5.3}$$

$$RTT_{min} \approx t_u + t_d \tag{5.4}$$

In other words, $RD - RD_{min}$ is a good estimate $\hat{t}_{buff}$, the time that a packet spends in the uplink buffer. Note that we only estimate $\hat{t}_{buff}$ from RD and RD_{min} when there are no packet losses and no re-ordering.

The effective TCP sliding window at the mobile sender is the minimum of the TCP congestion window (`cwnd`) and the `rwnd`. By adjusting the value of `rwnd`, the receiver can cap the growth of the sliding window. Recall that our objective is to regulate the uplink buffer at the mobile sender so that the available capacity is utilized while simultaneously minimizing the uplink delay. Therefore, using this strategy, we advertise an appropriate value of `rwnd` based on the estimated $\hat{t}_{buff}$.

If $\hat{t}_{buff}$ is larger than a threshold T, we say that the system is in the *fast* state because there are too many packets in the uplink buffer. Conversely, if $\hat{t}_{buff} \leq T$, we say the system is in the *slow* state since we can possibly allow more packets to be sent or buffered. It remains for us to determine a value for T, which achieves a good trade-off between delay and link utilization. With a small value of T, we can achieve a lower delay at a higher risk of under-utilizing the upstream link. We evaluated different choices of T and found that setting $T = RTT_{min}$ keeps the RTT below 1 second 80% of the time, while keeping the upload throughput similar to that of TCP Cubic. Reducing T below RTT_{min} will reduce the throughput without

much reduction to the RTT. Thus, we set $T = RTT_{min}$ in our implementation.

Fast State. In the fast state, our algorithm will freeze the growth of the TCP buffer to prevent excessive delays, by advertising the `rwnd` as $min(rwnd, \lceil(\rho \times RTT_{min})/MSS\rceil \times MSS)$, where ρ is the rate at which packets are received, MSS is the Maximum Segment Size, and $rwnd$ is the original `rwnd` computed by the kernel. This sets the advertised window to the estimated bandwidth-delay product (BDP) for the transmission. Note that we cannot advertise more than what is originally allowed by the kernel to prevent overflowing the receiver's buffer. Upon receiving the new `rwnd`, the mobile sender will stop queuing more packets and its buffer will start to empty. At some point, $\hat{t}_{buff}$ will drop below T, causing the algorithm to enter the slow state.

Slow State. In the slow state, we want to send more packets to ensure that we fully utilize the available uplink capacity. To do so, we need to increase the advertised `rwnd`. Like TCP slow start, we increase the `rwnd` by one MSS after the receiver receives each data packet. Eventually, the TCP buffer at the sender would begin to fill and $\hat{t}_{buff}$ will start to increase until it exceeds T, and the algorithm returns to the fast state.

5.1.2 Maximum Buffer Utilization

As the algorithm oscillates between the fast and slow states, the number of packets in the uplink buffer would fluctuate over time. Hence, we need to ask and answer the following question: what is the maximum time that we expect a packet to stay in the uplink buffer? The answer will allow us understand the effect of selecting different values for the threshold T, and yield important insight into how RSFC

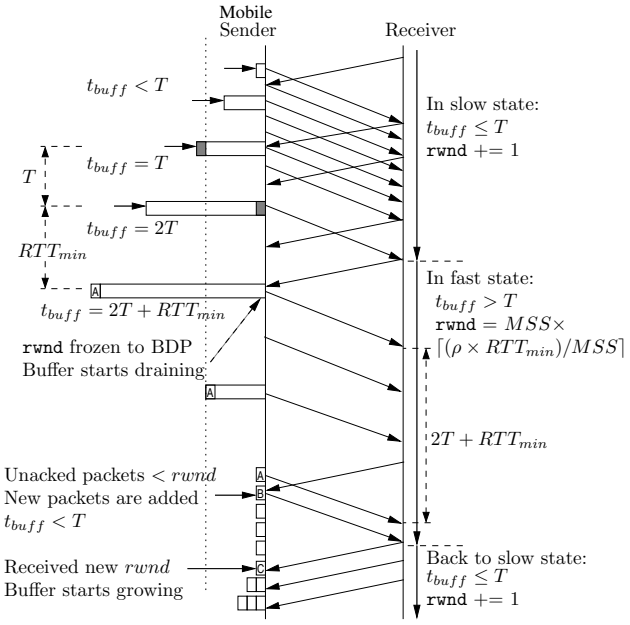


Figure 5.2: Packet flow diagram illustrating a typical scenario for buffer inflation.

adapts to changing network conditions. In Figure 5.2, we illustrate the change in the number of packets in the uplink buffer over time.

Suppose that the mobile sender starts in the slow state (i.e. $\hat{t}_{buff} < T$). The receiver will continuously increase the advertised `rwnd` by one *MSS* for every data packet received. The sender will add more packets to the buffer and at some point, it will add a packet that needs to spend a time $t_{buff} = T$ in the buffer before it is transmitted (See shaded packet in Figure 5.2). As this shaded packet moves to the head of the buffer, the sender will continue to receive ACKs for the data packets sent earlier, and the uplink buffer will continue to grow by one *MSS* for each ACK. When the shaded packet reaches the head of the buffer, by definition, the buffer queue would have reached a point such that the next packet queued will have a $t_{buff} = 2T$, since it took the shaded packet an amount of time equal to T to reach the head.

Once the receiver receives the shaded packet, it switches to the fast state and freezes the advertised `rwnd`. However, it will take a roundtrip time of RTT_{min} from the time of transmission of the shaded packet for this frozen `rwnd` to reach the sender. Therefore, until the new `rwnd` is received, the sender would keep queuing new packets as it receives ACKs with increasing `rwnd`. When the new `rwnd` is finally received, the buffer queue would be such that the next packet queued (see packet A in Figure 5.2) will have $t_{buff} = 2T + RTT_{min}$. This will be the maximum buffer delay because the `rwnd` is now capped at the estimated BDP $\lceil(\rho \times RTT_{min})/MSS\rceil \times MSS$. Since we set T as RTT_{min} in our implementation, the largest expected buffer delay t_{buff} is $3 \times RTT_{min}$.

At this point, the uplink buffer will start to drain and a new packet will only be pushed into the uplink buffer when the number of unacknowledged packets drops below `rwnd`. As `rwnd` is set to the BDP, which is the number of packets in flight, the buffer will eventually be completely emptied. When this happens, the new packet sent will spend a time $t_{buff} = 0 < T$ in the buffer. Hence, when the receiver receives this new packet, the algorithm will revert to the slow state and start increasing the `rwnd` by one MSS for each packet. Once again, it will take a time of RTT_{min} from the transmission of the packet for the new `rwnd` to reach the sender. Hence, in a typical scenario, the algorithm will spend at most $2T + 2RTT_{min}$ in the fast state before going back to the slow state.

5.1.3 Handling Changes in the Network

Because the 3G uplink is a shared resource, the base station will allocate an amount of air time to each mobile device depending on factors such as signal

quality and the number of connected devices. Hence, the available bandwidth and associated network delays are expected to change over time. In other words, we expect the variables ρ, t_d and t_u to vary over time and RSFC needs to adapt to these changes.

If there is a change only in the available bandwidth that results in a change of the receive rate ρ, there will not be any impact on the algorithm because ρ is only used to estimate the BDP in the fast state. When ρ changes, we simply update our estimate of the BDP with the new ρ. Also, the switching between the fast and slow states is independent of ρ and depends only on the delays t_u and t_d.

Similarly, if there is a decrease in network delays t_u and t_d, no special handling is required. A decrease in either t_u or t_d will cause RD or RTT to drop lower than the previously recorded values of RD_{min} or RTT_{min}, and these two variables will simply be updated. Even if we do not update RTT_{min} accordingly, the older (and larger) value of RTT_{min} will simply result in an over-estimation of the BDP. This means that the `rwnd` set in fast state will not allow the buffer to empty completely and $\hat{t}_{buff}$ will not be reduced to zero. As long as $\hat{t}_{buff}$ remains small, the uplink is still fully utilized and `rwnd` is still capped, there will be an upper bound on the delay.

The challenge arises when either of the network delays, t_u or t_d, increase. This will eventually cause the algorithm to enter the fast state, since the increasing `rwnd` in the slow state will cause the sender to queue packets until the $\hat{t}_{buff} > T$. In the fast state, the BDP will be under-estimated because the estimated $RTT_{min} < t_u + t_d$. This causes `rwnd` to be set at a value lower than the actual capacity of the link. The buffer will eventually empty, yet the low `rwnd` prevents new packets from being sent, which causes the link to be under-utilized. In other words, an increase

in t_u might be interpreted as an increase in $\hat{t}_{buff}$, which means that $\hat{t}_{buff}$ can never fall to zero. There are two possible situations:

(i) *Enters Slow State*. When the increase in t_u is not large and $\hat{t}_{buff} \leq T$, `rwnd` will start increasing, allowing the sender to send more packets for every ACK received. Since the link was previously under-utilized, this will result in ρ increasing at the receiver. Eventually the algorithm will return to the fast state. Hence, if we detect an increase in ρ in the most recent cycle of slow and fast states, we can safely assume that link utilization had dropped at some point and we update (increase) RD_{min} and RTT_{min} to the minimum RD and RTT values observed in the most recent cycle of slow and fast states.

(ii) *Stuck in Fast State*. If the increase in t_u is sufficiently large such that $\hat{t}_{buff} > T$, the algorithm will end up getting stuck in the fast state and the link will always be under-utilized. We showed earlier in Section 5.1.2 that the algorithm will typically spend at most $2T + 2RTT_{min}$ in the fast state before reverting to the slow state. Hence, we can deduce that something is wrong if the algorithm spends significantly longer than $2T + 2RTT_{min}$ in the fast state. In this light, if we find that the algorithm stays in the fast state for longer than $2T + 3RTT_{min}$, we will switch to a third special *monitor* state. Essentially, we add an extra RTT_{min} to provision for possible transient changes in delays.

Monitor State. Like in the slow state, `rwnd` is increased by 1 for every data packet received and we monitor ρ. There are two possible scenarios:

(a) If an increase is detected in ρ, we switch to the slow state. Simultaneously, RD_{min} and RTT_{min} is updated to the minimum RD and RTT observed while

in the monitor state and `rwnd` is updated to $\lceil(\rho \times RTT_{min})/MSS\rceil \times MSS$ accordingly with the new value of RTT_{min}.

(b) If ρ does not seem to increase even after RD increases by T, it suggests that the link is still fully utilized, i.e. there are still packets in the uplink buffer. We then halve both RD_{min} and RTT_{min} and switch to the fast state. This will likely force the buffer to empty and cause RSFC to switch back to monitor state. However, this time in the monitor state, we will likely end up in case (a) above and both RD_{min} and RTT_{min} will be set to the correct values.

We demonstrate in Section 5.2.5 that the monitor state is crucial for RSFC to achieve good link utilization in the presence of network variations.

5.1.4 Practical Deployment

RSFC requires minor modifications to the TCP stack at the TCP receiver, but *no modification* needs to be made to an existing mobile device, though it does require the TCP timestamp option to be enabled. A quick survey of the available smartphones suggests that the TCP timestamp option is enabled by default for both Android and iPhone (which together constitute about 93.2% of the global smartphone market [33]) and disabled by default for Windows Mobile phones. In this light, we believe that the majority of smartphones are RSFC-ready at present.

The current architecture of existing cellular data networks makes it relatively straightforward to deploy our algorithm. Wang et al. observed that the proxies or middleboxes are widely deployed in existing cellular data networks [95]. These proxies or middleboxes are used to improve 3G performance by caching commonly accessed web content (such as images on popular websites) and in some

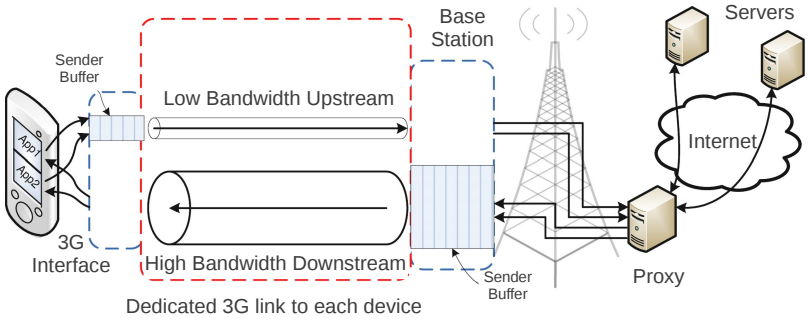


Figure 5.3: The bottleneck 3G link is virtually dedicated to each device. Multiplexing is done by the ISP in a schedule which is assumed to be fair.

cases, to perform QoS filtering on the traffic. We verified their observations that all our three local mobile ISPs implement a transparent web proxy that intercepts all HTTP connections, and effectively converts them into split TCP [67] connections. This situation suggests that we can deploy RSFC for an entire 3G network easily by modifying the TCP stack on these proxies as illustrated in Figure 5.3.

5.2 Performance Evaluation

In this section, we present our evaluation results for RSFC. We begin by investigating RSFC's effectiveness at reducing the RTT and in improving throughput. Next, we evaluate how RSFC performs in two possible application scenarios: (i) when users surf the web while there is a concurrent background upload, and (ii) when there are simultaneous uploads. Finally, we demonstrate the necessity of having to adapt to changing network conditions and also show that RSFC is compatible with other TCP congestion control algorithms.

All of the experiments in this section were conducted in our lab, on Android phones, on all three local ISPs. The congestion control algorithm used on the phones is TCP Cubic [39], which is the default TCP implementation in the An-

droid kernel. It is clear from our measurement results in Section 4.4 that ISP A has the poorest upload performance in our lab. As to be expected, RSFC achieves the greatest improvement in performance with ISP A's network. For ISP B and ISP C, where the upload speeds are relatively high, RSFC was less effective, though it does not perform worse than the default TCP Cubic. Because our goal is to show that RSFC can significantly improve downloads when mobile connectivity is less than ideal (and almost all users will find themselves at such locations every once in a while), we present only the results for ISP A, except where stated otherwise.

5.2.1 Reduction in RTT

To verify that RSFC can reduce the RTT for a TCP upload, we ran an experiment where we uploaded 1 MB of data from the phone using TCP Cubic and then repeated the process with RSFC. This experiment was repeated periodically every 15 minutes over several days and the cumulative distribution of our results is shown in Figure 5.4. We observe that in general, using RSFC results in much lower RTT compared to TCP Cubic. TCP Cubic's RTT is greater than 6 s 50% of the time while the RTT with RSFC is close to 1 s more than 90% of the time. It is also apparent that the upload throughput achieved by RSFC is almost identical to that for TCP Cubic. This simple experiment shows that RSFC can achieve a significant reduction in RTT without suffering any loss in upload throughput.

5.2.2 Improving Downstream Throughput

Next, we measured the improvement in downlink utilization achieved in the presence of a concurrent upload by running the following sets of experiments: (i) a

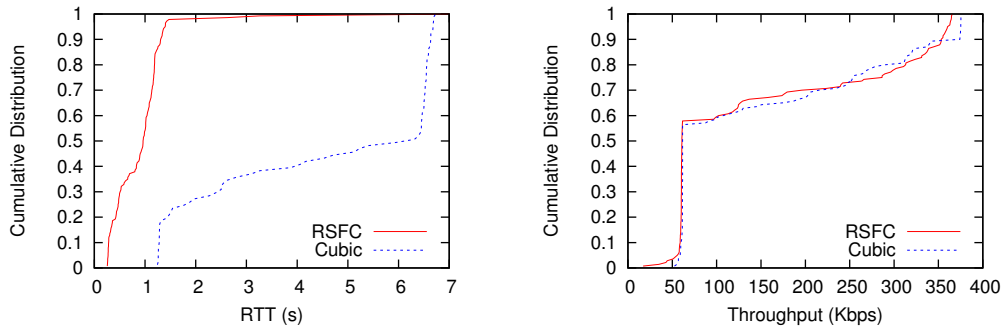


Figure 5.4: Cumulative distribution of RTT and throughput for TCP Cubic and RSFC uploads.

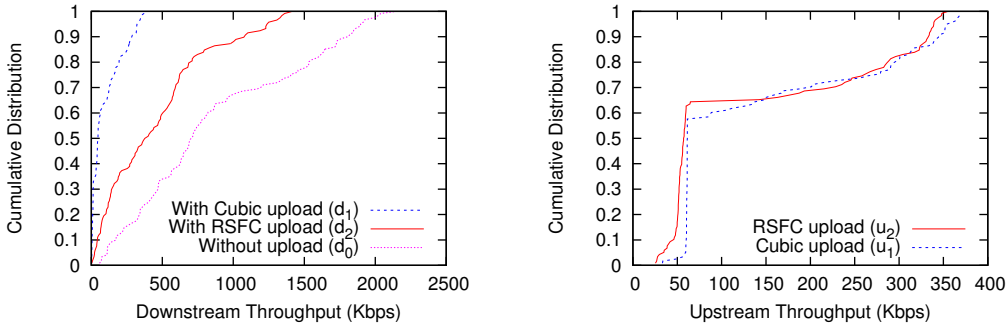


Figure 5.5: Cumulative distribution of the throughput achieved by the downstream and upstream flows under different conditions.

single TCP Cubic download flow (d_0), (ii) a single TCP Cubic upload flow (u_0), (iii) a TCP Cubic download flow with a concurrent TCP Cubic upload flow (d_1 and u_1), and (iv) a TCP Cubic download flow with a concurrent RSFC upload flow (d_2 and u_2). These experiments were run one after the other to minimize the effect of temporal variance. In Figure 5.5, we plot the cumulative distribution of the throughputs achieved.

As expected, the throughput of d_1 is poor. It is lower than 400 kbps all the time and it is close to 10 kbps 30% of the time. With RSFC, the downstream throughput improves significantly and 50% of the time, it is greater than 400 kbps. When we compare the throughputs of u_2 and u_1, we found that they are similar in spite of the increase in downlink utilization for RSFC. This suggests that our approach of

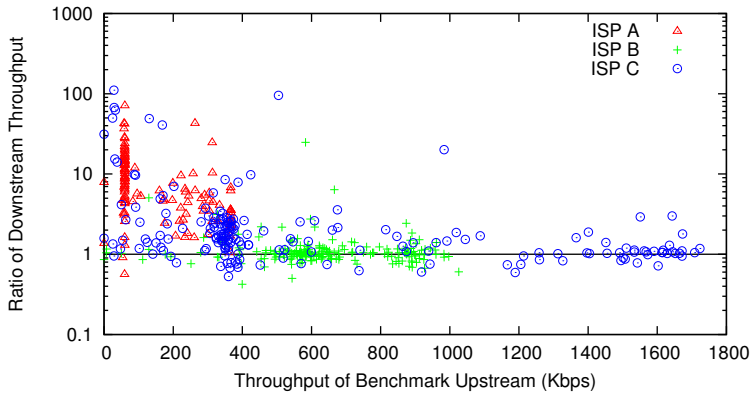


Figure 5.6: Plot of ratio between RSFC's downstream throughput to that of TCP Cubic against the throughput of the benchmark upstream flow.

regulating the uplink buffer size is effective; the buffer always has packets to send but the packets are not unnecessarily delayed. However, the throughput of d_2 is still not as good as the d_0 benchmark. As discussed in Section 4.4, this is likely due to the interactions between the concurrent flows arising from scheduling at the 3G layer, which is beyond our control.

We also investigated how the current uplink capacity will affect the improvement in the downlink utilization. In Figure 5.6, we compare RSFC to TCP Cubic by plotting the throughput ratio d_2/d_1 against u_0. We included the data points from all three ISPs for a better overview. It is clear that RSFC achieves the greatest improvement when the upload throughput is below 400 kbps. Figure 5.6 also shows that RSFC does not achieve much improvements for ISP B and ISP C. These two ISPs typically have higher uplink capacity, which makes it more unlikely for the uplink buffer to be saturated. In such cases, RSFC has a similar performance as TCP Cubic. Note that our experiments were conducted at a fixed location where ISP B and ISP C seem to have significantly better performance than ISP A. However, they could have lower upload speeds at other locations and

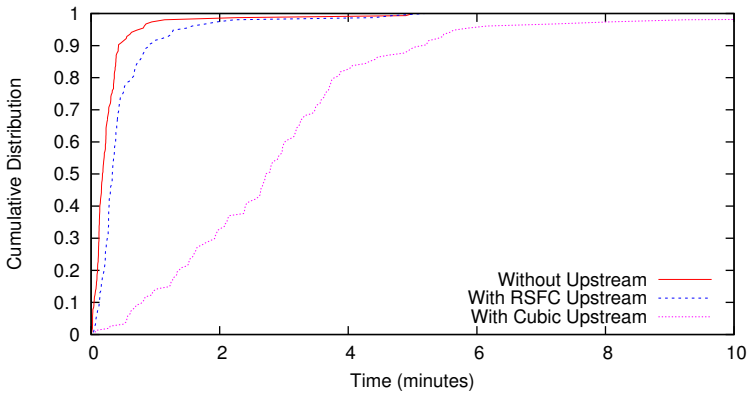


Figure 5.7: Cumulative distribution of the time taken to load the top 100 websites under different conditions.

in those instances, RSFC would be helpful for ISP B and ISP C as well.

5.2.3 Improving Web Surfing

Next, we investigate how RSFC performs in a common scenario: surfing the web while uploading data in the background. In this experiment, we visited the Alexa Top 100 websites [9] under three conditions: (i) with a concurrent RSFC upload, (ii) with a concurrent TCP Cubic upload, and (iii) without any concurrent uploads. In Figure 5.7, we plot the cumulative distribution of the time taken to receive the last byte of the last HTTP response. Without a concurrent upload, 90% of these websites take less than 30 s to load. However, with a TCP Cubic upload, 70% of them can take more than 2 minutes to load. Such performance degradation will severely impact user-perceived performance of the mobile web access. RSFC mitigates the impact of a concurrent upload as it reduce the load time to within 30 s for 70% of the websites and to within 1 minute for 90% of them.

5.2.4 Fairness of Competing RSFC Uploads

If there are multiple upload flows from a single 3G connection, RSFC will be applied to each flow independently. The rate estimation and `rwnd` advertisement for a flow would be done without considering the other flows. We attempt to understand how well this simple scheme will perform in practice by running the following experiment. We first upload 1 MB of data using a single TCP Cubic flow as a reference. After the upload is complete, we start two concurrent RSFC upload flows from the phone and upload 1 MB of data in total. This experiment was repeatedly executed over a 24-hour period.

We quantified the fairness of the throughput utilization between the two RSFC flows using the Jain fairness index [50], i.e. $(R_1+R_2)^2/(2\times({R_1}^2+{R_2}^2))$, where R_1 and R_2 are the throughput of the two flows. We also compared the efficiency of the throughput utilization for the RSFC flows to the TCP Cubic flow by computing the ratio $(R_1+R_2)/C$, where C is the throughput of the TCP Cubic flow. A cumulative distribution of our findings is shown in Figure 5.8.

We found that the two RSFC flows achieve very similar throughput. It turns out that the oscillating nature of our algorithm, between the fast and slow states, ensures that neither flow dominates the uplink. Hence, we conclude that RSFC flows are relatively fair when they contend with each other. In terms of utilization efficiency, the two concurrent RSFC flows seem to perform no worse than a single TCP Cubic flow about 80% of the time and can occasionally achieve better throughput since parallel TCP flows can generally better utilize the link capacity compared to a single flow.

For the remaining 20% of the time, the two RSFC flows seemed to perform

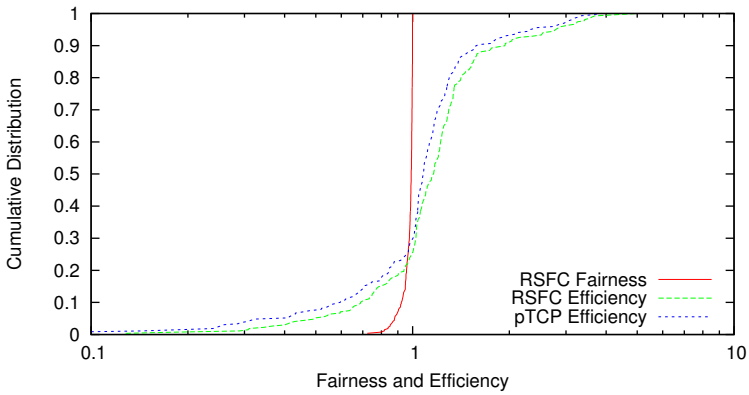


Figure 5.8: Cumulative distribution of the fairness between two RSFC uploads and the efficiency of two RSFC uploads compared to a single TCP Cubic upload.

worse than a single TCP Cubic flow. To further investigate why this might be the case, we ran a benchmark experiment where we compared the efficiency of two parallel TCP Cubic upload flows against a single TCP Cubic flow. The results of this experiment is labeled in Figure 5.8 as "pTCP Efficiency". Since this new curve was remarkably similar to the curve for the parallel RSFC flows, it suggests that it is likely that the two RSFC flows performed worse 20% of the time because of temporal variations in the 3G link.

5.2.5 Adapting to Changing Network Conditions

In Section 5.1.3, we described how RSFC updates RD_{min} and RTT_{min} to handle variations of the 3G link capacity to maintain good link utilization. To evaluate the necessity of updating RD_{min} and RTT_{min}, we ran an experiment where we uploaded data for 60 s using a variant of RSFC that keeps the values to the lowest measured from the start of the flow. We plot the resulting throughput and RTT achieved in Figure 5.9.

At the start of the flow, the estimated RD_{min} and RTT_{min} are accurate and the

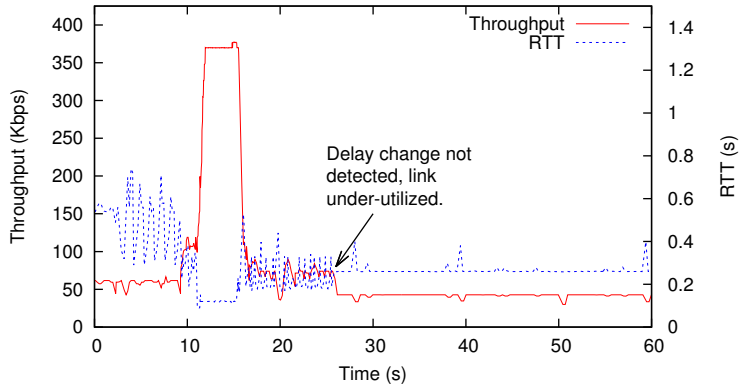


Figure 5.9: Plot of the average throughput achieved and RTT using RSFC variant without RD_{min} and RTT_{min} update mechanism.

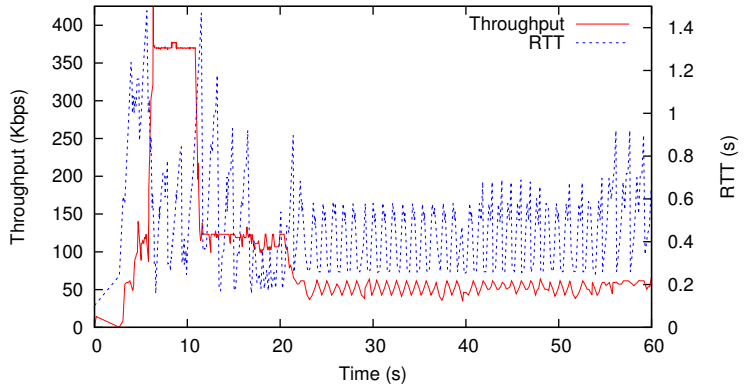


Figure 5.10: Plot of the average throughput achieved and RTT using full RSFC algorithm.

uplink is fully utilized. There are network fluctuations in the middle of the flow and after 25 s, the throughput is lower than that achieved during the first 10 s even though the network conditions for both periods are similar. This is because the underlying network delays decreased then subsequently increased, but their estimates RD_{min} and RTT_{min} were not updated. Hence, the `rwnd` advertised is smaller than the ideal value and the link is under-utilized.

We repeated this experiment with the default version of RSFC and our results are shown in Figure 5.10. We can infer from the oscillations in the RTT that the feedback mechanism of RSFC is operating correctly even though like before, there

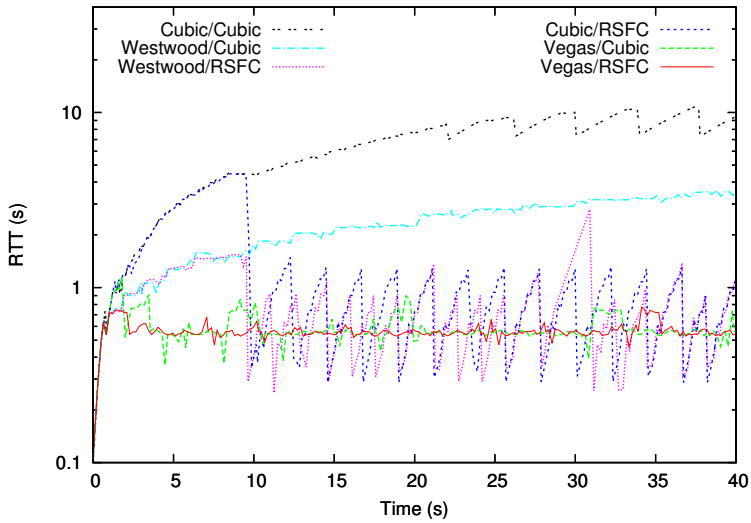


Figure 5.11: Plot of the RTT for the transfer of 1 MB file using different TCP variants at both sender and receiver side. In the legend, we indicate first the mobile sender followed by the receiver.

was a temporary decrease in the network delay at 5 s and a subsequent increase at 20 s. This demonstrates that the mechanism to update RD_{min} and RTT_{min} is effective and necessary to adapt to changing network conditions.

5.2.6 Compatibility with other TCP variants

We investigated RSFC's compatibility with other sender-side TCP congestion control algorithms by running a new set of experiments. In these experiments, we uploaded 1 MB of data from the phone to a receiver and alternated the TCP implementation on the phone between Westwood [68], Vegas [22], New Reno [43] and TCP Cubic [39]. We also varied the receiver-side algorithm on the server between TCP Cubic and RSFC.

In Figure 5.11, we plot the RTT achieved in the different experiments (the results of New Reno at the sender is not shown because it is similar to TCP Cubic). We see that if the receiver uses RSFC, then the RTT can be improved regardless of

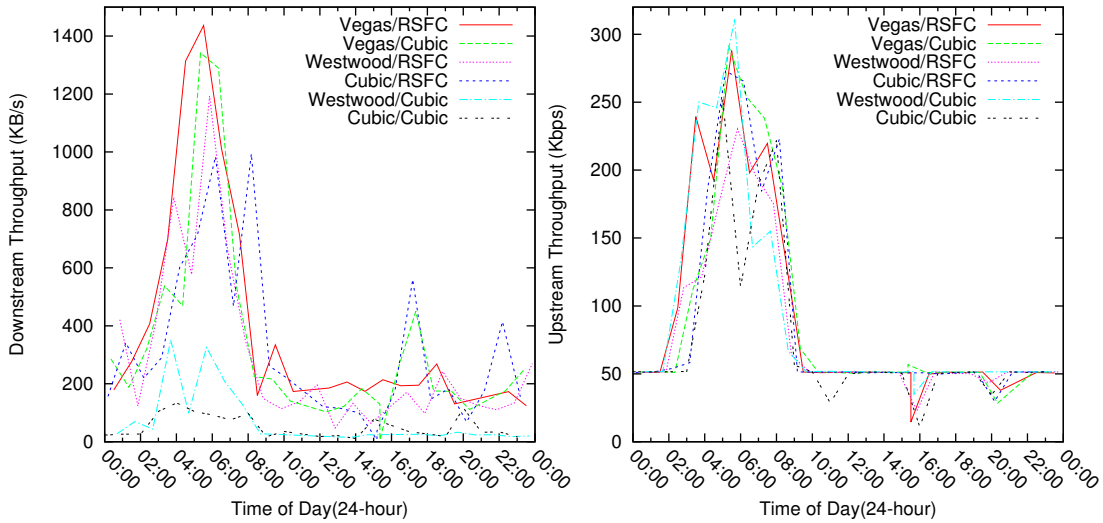


Figure 5.12: Plot of downstream throughput when the upstream is saturated with different algorithms over a 24-hour period. In the legend, we indicate first the mobile sender followed by the receiver.

the algorithm used at the sender. We also ran the simultaneous upload and download experiments described in Section 5.2.2 and plot our results in Figure 5.12. Once again, as long as RSFC is enabled at the receiver, the downlink utilization can be improved. This shows that RSFC is compatible with other TCP variants.

We noticed that the performance of Vegas/Cubic is slightly better than that for Cubic/RSFC. This is to be expected as Vegas is more aggressive in controlling the delays compared to RSFC. Moreover, TCP Vegas can immediately effect changes on the sender whereas RSFC's changes are delayed by at least one RTT_{min}. However, RSFC is fully compatible with TCP Vegas and is a compelling alternative to deploying TCP Vegas on mobile devices.

5.3 Summary

In this chapter, we proposed and evaluated a new algorithm called *Receiver-side Flow Control* (RSFC) and showed that it is effective at mitigating the "self-inflicted" congestion problem. RSFC is compatible with existing sender-side TCP congestion control algorithms, and is a practical technique that can be easily deployed at proxies of cellular data networks without requiring any modifications of the existing mobile devices. The key insight of our algorithm is a feedback loop that can dynamically adapt to the variations of the cellular link, by i) using the TCP timestamp to continuously estimate the one-way delay, queuing delay and RTT; ii) inferring whether the link is congested using the queuing delay instead of the packet loss; iii) continuously estimating the available uplink bandwidth and advertising an appropriate TCP receiver window (`rwnd`) to regulate the sending of the upstream.

Chapter 6

Conclusion and Future Work

In this chapter, we summarize our work and discuss some open questions and possible directions for future research.

In this thesis, we investigated the congestion problems in the wired Internet and in modern cellular data networks. For the wired Internet, we proposed and evaluated a new *massively-multipath (mPath) source routing* algorithm to improve end-to-end throughput for high-volume data transfers. For the cellular data networks, we conducted a thorough measurement study to understand the properties of the cellular link and in particular we identify a "self-inflicted" congestion problem that is caused by the saturated uplink buffer. Then, we proposed and evaluated a *Receiver-side Flow Control* (RSFC) algorithm to mitigate the uplink saturation problem.

Congestion in Wired Internet

Our studies corroborate the fact that the congestion of the wired Internet can often happen in the Internet core and show that detour paths can be used to route

around core-link bottlenecks. There are typically many possible detour paths and they can be easily constructed by simply employing a set of stateless and light-weight proxies. A number of practical mechanisms can be integrated to make the multipath solution more effective and efficient, including i) using the actual data to probe and evaluate the network passively during transmission; ii) dynamically adding or removing detour paths according to the path quality; iii) a coordinated congestion control algorithm that can achieve TCP friendliness; iv) using *loss intervals* to infer shared bottlenecks and v) using a *load aggregation* operation to re-distribute the data load when the shared link is detected.

Multipath routing is currently not widely used in practice due to the lack of infrastructure support and limited availability of multi-homing, which are required by existing solutions. Since mPath only requires stateless proxies to enable efficient multipath data transfers, we believe it is a more practical solution given the state of existing network infrastructure. Another factor that has traditionally hindered the adoption of multipath routing is the lack of use cases. However, recent work on using multipath solutions to transfer bulk data between datacenters [64, 77] shows that there are potential applications for high throughput multipath routing, and mPath can potentially be applied to these and other scenarios.

Congestion in Cellular Data Networks

We showed that the throughput of cellular data networks can vary by as much as two orders of magnitude within a 10-minute interval, and found that mobile ISPs often deploy large and separate downlink buffers for each user. ISPs also typically implement some form of fair queuing, but for different networks, the

buffer management policies may be quite different. We further measured the influence of the saturated uplink to the downlink performance, which we identified as "self-inflicted" congestion problem. We showed that the performance of a TCP download can be significantly degraded by a simultaneous TCP upload.

We then proposed a new algorithm called *Receiver-side Flow Control* (RSFC) and show that it is effective at mitigating the uplink saturation problem, especially in situations where uplink bandwidth is low. RSFC is compatible with existing sender-side TCP congestion control algorithms, and is a practical technique that can be easily deployed at proxies of cellular data networks without requiring any modifications of the existing mobile devices. Given that simultaneous uploads and downloads are likely going to become more common in cellular data networks, we believe that RSFC is an important mechanism for improving the user-perceived performance of the mobile devices, even though its benefits may seem somewhat indirect.

Generally, the congestion control mechanism of TCP does not work well with modern cellular links, as demonstrated by our experiments and also other work [46]. The TCP congestion control is required to maintain the fairness between flows in conventional networks where i) a single buffer is used for all competing flows; ii) FIFO and drop-tail policy are employed; iii) no fair scheduling is enforced. However, these conditions do not hold in cellular data networks because each subscriber typically has a dedicated buffer and the fair scheduling is enforced by the ISPs. Hence, there is no need to enforce fairness between users at transport layer and we can focus on optimizing performance instead.

RSFC is one attempt to optimize the performance for a single user and we show that it is effective in solving the "self-inflicted" congestion problem. We

investigate several practical and effective mechanisms, including: i) using TCP timestamp option to estimate many metrics like the one-way delay, queuing delay and RTT; ii) the one-way delay is extremely useful in the asymmetric scenario; iii) the queuing delay is a better indication of "self-inflicted" congestion than packet loss when the buffer is huge; iv) the `rwnd` can play a more important role in flow control, instead of just an indication of whether the receiver's buffer is full.

6.1 Open Issues and Future Work

Recently, multipath solutions using WiFi and cellular data networks simultaneously have attracted much attention [97, 20, 25]. The recently released IOS 7 was also found to be the first commercial system to incorporate MPTCP [21]. While the goal of the MPTCP deployment is likely to better exploit the properties of different networks, we believe that the general principles of multipath TCP routing discussed in this thesis is still applicable and more work remains to be done to better understanding multipath solutions for mobile data networks.

The second possible application of multipath routing is to potentially enhance the privacy and security for data transmission. In a single-path routing, all the intermediate nodes will have access to all the traffic for a flow that is routed through it. This might give rise to certain privacy and security problems as the node can potentially derive a unique profile and fingerprint for every flow that goes though it. Multipath routing could potentially allow us to avoid this problem by ensuring that no intermediate node has access to every packet in a data transmissions. It is interesting to investigate how can we find an optimal set of disjoint paths and distribute the data accordingly.

A third open question is motivated by our measurement studies. We have found that different ISPs and even different devices use different buffer configurations and queuing policies. It is unlikely that the chosen configurations and policies are all optimal. Whether these configurations are optimal and what makes a configuration optimal are candidates for further study.

Finally, it might be worth investigating how can we optimize the transmission protocols to be more cellular-friendly. In this thesis, we discuss one possible mechanism called Receiver-side Flow Control [99] and we further investigate a TCP rate control mechanism with the cooperation of other researchers [65]. Other recent works also suggest that more can be done in optimizing the transmission protocols for cellular data networks [46, 96]. If we can ignore the problem of fairness and assume that the mobile ISP has fair queuing mechanisms that will take care of that, we believe that there is scope to further optimize performance for individual subscribers.

Bibliography

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20122017 .

[2] ISPCheck. https://play.google.com/store/apps/details?id=com.ispcheck.

[3] SpeedTest. http://www.speedtest.net/mobile.php. [Online; accessed 16-September-2013].

[4] Atul Adya, Gregory Cooper, Daniel Myers, and Michael Piatek. Thialfi: A Client Notification Service for Internet-Scale Applications. In *Proceedings of SOSP '11*, October 2011.

[5] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the Performance of TCP Pacing. In *Proceedings of IEEE INFOCOM '00*, March 2000.

[6] Vaneet Aggarwal, Rittwik Jana, Kadangode Ramakrishnan, Jeffrey Pang, and N K Shankaranarayanan. Characterizing Fairness for 3G Wireless Networks. In *Proceedings of LANMAN '11*, October 2011.

[7] Aditya Akella, Jeff Pang, Bruce Maggs, Srinivasan Seshan, and Anees Shaikh. A Comparison of Overlay Routing and Multihoming Route Control. In *Proceedings of SIGCOMM '04*, September 2004.

[8] Aditya Akella, Srinivasan Seshan, and Anees Shaikh. An Empirical Evaluation of Wide-area Internet Bottlenecks. In *Proceedings of IMC '03*, October 2003.

[9] Alexa. Top Global Sites. `http://www.alexa.com/topsites`. [Online; accessed 15-February-2012].

[10] Manuel Álvarez Campana, Enrique Vázquez, Joan Vinyes, and Víctor Villagrá. Measuring Quality of Experience of Internet Access over HSDPA. In *Proceedings of WMNC '08*, October 2008.

[11] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proceedings of SOSP '01*, October 2001.

[12] Lachlan L.H. Andrew, Stephen V. Hanly, and Rami G. Mukhtar. Active Queue Management for Fair Resource Allocation in Wireless Networks. *IEEE Transactions on Mobile Computing*, 7(2):231–246, February 2008.

[13] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. In *Proceedings of SIGCOMM '04*, August 2004.

[14] Patrik Arlos and Markus Fiedler. Influence of the Packet Size on the One-Way Delay in 3G Networks. In *Proceedings of PAM '10*, April 2010.

[15] AT&T. A Different Take on the Big Game - Stats from the Stands. http://www.attinnovationspace.com/innovation/story/a7780988. [Online; accessed 15-April-2012].

[16] François Baccelli, Giovanna Carofiglio, and Serguei Foss. Proxy Caching in Split TCP: Dynamics, Stability and Tail Asymptotics. In *Proceedings of INFOCOM '08*, April 2008.

[17] Hari Balakrishnan, Venkata N. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. TCP Performance Implications of Network Path Asymmetry. RFC 3449, December 2002.

[18] Hari Balakrishnan, Venkata N. Padmanabhan, and Randy H. Katz. The Effects of Asymmetry on TCP Performance. In *Proceedings of MobiCom '97*, September 1997.

[19] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of SIGCOMM '99*, September 1999.

[20] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. Augmenting Mobile 3G Using WiFi. In *Proceedings of MobiSys '10*, June 2010.

[21] Olivier Bonaventure. Apple Seems to also Believe in Multipath TCP. http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html. [Online; accessed 30-September-2012].

[22] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[23] Mun Choon Chan and Ramachandran Ramjee. TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation. In *Proceedings of MobiCom '02*. ACM, September 2002.

[24] Mun Choon Chan and Ramachandran Ramjee. Improving TCP/IP Performance over Third-Generation Wireless Networks. *IEEE Transactions on Mobile Computing*, 7(4):430–443, 2008.

[25] Yung-Chih Chen, Yeon sup Lim, Richard J. Gibbens, Erich Nahum, Ramin Khalili, and Don Towsley. A Measurement-based Study of MultiPath TCP Performance over Wireless Networks. In *Proceedings of IMC '13*, October 2013.

[26] Dah Ming Chiu and Raj Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[27] Luca De Cicco and Saverio Mascolo. TCP Congestion Control over 3G Communication Systems: an Experimental Evaluation of New Reno, BIC and Westwood+. In *Proceedings of NEW2AN '07*, September 2007.

[28] Pralhad Deshpande, Xiaoxiao Hou, and Samir R. Das. Performance Comparison of 3G and Metro-Scale WiFi for Vehicular Network Access. In *Proceedings of IMC '10*, November 2010.

[29] Ahmed Elmokashfi, Amund Kvalbein, Jie Xiang, and Kristian R. Evensen. Characterizing Delays in Norwegian 3G Networks. In *Proceedings of PAM '12*, March 2012.

[30] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with Very Small Buffers. In *Proceedings of IEEE INFOCOM '06*, April 2006.

[31] Teng Fei, Shu Tao, Lixin Gao, and Roch Guerin. How to Select a Good Alternate Path in Large Peer-to-Peer Systems. In *Proceedings of IEEE INFOCOM '06*, April 2006.

[32] Michael J. Freedman, Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica. Non-Transitive Connectivity and DHTs. In *Proceedings of WORLDS '05*, December 2005.

[33] Gartner. Market Share Analysis: Mobile Phones, Worldwide, 2Q13. `http://www.gartner.com/id=2573119`. [Online; accessed 3-October-2013].

[34] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11):40–54, November 2011.

[35] Tom Goff, James Moronski, and D. S. Phatak. Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments. In *Proceedings of IEEE INFOCOM '00*, March 2000.

[36] Yunhong Gu and Robert L. Grossman. UDT: UDP-based Data Transfer for High-Speed Wide Area Networks. *Computer Networks*, 51(7):1777–1799, May 2007.

[37] Yunhong Gu, Xinwei Hong, and Robert Grossman. An Analysis of AIMD Algorithms with Decreasing Increases. In *Proceedings of GridNets '04*, October 2004.

[38] Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proceedings of OSDI '04*, December 2004.

[39] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, July 2008.

[40] Thomas J. Hacker, Brian D. Athey, and Brian Noble. The End-to-end Performance Effects of Parallel TCP Sockets on a Lossy Wide-area Network. In *Proceedings of IPDPS '02*, April 2002.

[41] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Transactions on Networking*, 14(6):1260–1271, December 2006.

[42] Martin Heusse, Sears A. Merritt, Timothy X. Brown, and Andrzej Duda. Two-way TCP Connections: Old Problem, New Insight. *SIGCOMM Computer Communications Review*, 41(2):5–15, April 2011.

[43] Janey C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of SIGCOMM '96*, August 1996.

[44] Ningning Hu, Li Erran Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. Locating Internet Bottlenecks: Algorithms, Measurements, and Implications. In *Proceedings of SIGCOMM '04*, September 2004.

[45] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proceedings of MobiSys '12*, June 2012.

[46] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proceedings of SIGCOMM '13*, August 2013.

[47] iN2015 Infocomm Infrastructure, Services and Technology Development Sub-Committee. Totally Connected, Wired and Wireless, June 2006.

[48] Manish Jain and Constantinos Dovrolis. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proceedings of SIGCOMM '02*, August 2002.

[49] Rahul Jain and Teunis J. Ott. *Design and Implementation of Split TCP in the Linux Kernel*. PhD thesis, Newark, NJ, USA, 2007.

[50] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System. DEC Research Report TR-301, September 1984.

[51] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of IMC '12*, November 2012.

[52] Marko Jurvansuu, Jarmo Prokkola, Mikko Hanski, and Pekka Perälä. HSDPA Performance in Live Networks. In *Proceedings of ICC '07*, June 2007.

[53] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan. Improving TCP Throughput over Two-way Asymmetric Links: Analysis and Solutions. In *Proceedings of SIGMETRICS '98*, June 1998.

[54] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan. Explicit Window Adoption: A Method to Enhance TCP Performance. *IEEE/ACM Transactions on Networking*, 10(3):338–350, June 2002.

[55] Dina Katabi, Issam Bazzi, and Xiaowei Yang. A Passive Approach for Detecting Shared Bottlenecks. In *Proceedings of ICCCN '01*, October 2001.

[56] Srinivasan Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of SIGCOMM '91*, September 1991.

[57] Peter Key, Laurent Massoulié, and Bing Wang. Emulating Low-priority Transport at the Application Layer: a Background Transfer Service. In *Proceedings of SIGMETRICS '04*, June 2004.

[58] Peter Key, Laurent Massouli, and Don Towsley. Path Selection and Multipath Congestion Control. In *Proceedings of IEEE INFOCOM '07*, May 2007.

[59] Wookyun Kho, Salman Abdul Baset, and Henning Schulzrinne. Skype Relay Calls: Measurements and Experiments. In *Proceedings of IEEE INFOCOM '08*, April 2008.

[60] George Kola and Miron Livny. DiskRouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers. Technical Report CS-TR-2004-1518, UW–Madison, 2003.

[61] M. Kühlewind and B. Briscoe. Chirping for Congestion Control - Implementation Feasibility. In *Proceedings of PFLDNeT '10*, November 2010.

[62] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet Routing Convergence. In *Proceedings of SIGCOMM '00*, August 2000.

[63] Markus Laner, Philipp Svoboda, Eduard Hasenleithner, and Markus Rupp. Dissecting 3G Uplink Delay by Measuring in an Operational HSPA Network. In *Proceedings of PAM '11*, March 2011.

[64] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-Datacenter Bulk Transfers with NetStitcher. In *Proceedings of SIGCOMM '11*, August 2011.

[65] Wai Kay Leong, Yin Xu, Ben Leong, and Zixiao Wang. Mitigating Egregious ACK Delays in Cellular Data Networks by Eliminating TCP ACK Clocking. In *Proceedings of ICNP '13*, October 2013.

[66] Xin Liu, Ashwin Sridharan, Sridhar Machiraju, Mukund Seshadri, and Hui Zang. Experiences in a 3G Network: Interplay Between the Wireless Channel and Applications. In *Proceedings of MobiCom '08*, September 2008.

[67] David A. Maltz and Pravin Bhagwat. TCP Splicing for Application Layer Proxy Performance. *Journal of High Speed Networks*, 9(3):225–240, January 1999.

[68] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of MobiCom '01*, July 2001.

[69] M. Meyer. TCP Performance over GPRS. In *Proceedings of WCNC '99*, September 1999.

[70] Ivan Tam Ming-Chit, Du Jinsong, and Weiguo Wang. Improving TCP Performance Over Asymmetric Networks. *SIGCOMM Computer Communication Review*, 30(3):45–54, July 2000.

[71] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *Queue*, 10(5):20–34, May 2012.

[72] Vern Paxson. End-to-end Internet Packet Dynamics. In *Proceedings of SIGCOMM '97*, September 1997.

[73] Vern Paxson. End-to-end Routing Behavior In the Internet. *SIGCOMM Computer Communication Review*, 36(5):41–56, October 2006.

[74] Maxim Podlesny and Carey Williamson. Improving TCP Performance in Residential Broadband Networks: a Simple and Deployable Approach. *SIGCOMM Computer Communication Review*, 42(1):61–68, January 2012.

[75] Jarmo Prokkola, Pekka Perälä, Mikko Hanski, and Esa Piri. 3G/HSPA Performance in Live Networks from the End User Perspective. In *Proceedings of ICC '09*, June 2009.

[76] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *Proceedings of IMC '10*, November 2010.

[77] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proceedings of SIGCOMM '11*, August 2011.

[78] Shansi Ren, Lei Guo, and Xiaodong Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. In *Proceedings of ICDCS '06*, July 2006.

[79] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Proceedings of PAM '03*, April 2003.

[80] Dan Rubenstein, Jim Kurose, and Don Towsley. Detecting Shared Congestion of Flows Via End-to-End Measurement. *IEEE/ACM Transactions on Networking*, 10(3), June 2002.

[81] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, January 1999.

[82] R. Scheffenegger and M. Kuehlewind. Additional Negotiation in the TCP Timestamp Option Field during the TCP Handshake. IETF Working Draft, October 2011.

[83] Daryl Seah, Wai Kay Leong, Qingwei Yang, Ben Leong, and Ali Razeen. Peer NAT Proxies for Peer-to-Peer Applications. In *Proceedings of NetGames '09*, November 2009.

[84] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low Extra Delay Background Transport (LEDBAT). IETF Working Draft, October 2011.

[85] H. Sivakumar, S. Bailey, and R.L. Grossman. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *Proceedings of SC '00*, November 2000.

[86] Joel Sommers and Paul Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *Proceedings of IMC '12*, November 2012.

[87] Neil T. Spring, Maureen Chesire, Mark Berryman, Vivek Sahasranaman, Thomas Anderson, and Brian Bershad. Receiver Based Management of Low Bandwidth Access Links. In *Proceedings of IEEE INFOCOM '00*, March 2000.

[88] Stanford Linear Accelerator Center. The PingER Project. http://www-iepm.slac.stanford.edu/pinger/site.html. [Online; accessed 19-September-2013].

[89] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proceedings of IEEE INFOCOM '06*, April 2006.

[90] Wee Lum Tan, Fung Lam, and Wing Cheong Lau. An Empirical Study on the Capacity and Performance of 3G Networks. In *Proceedings of INFOCOM '07*, May 2007.

[91] Pablo Tapia, Jun Liu, Yasmin Karimli, and Martin J. Feuerstein. *HSPA Performance and Evolution: A Practical Perspective*. WILEY, 2009.

[92] Fung Po Tso, Jin Teng, Weijia Jia, and Dong Xuan. Mobility: A Double-Edged Sword for HSPA Networks. In *Proceedings of MobiHoc '10*, September 2010.

[93] Curtis Villamizar and Cheng Song. High Performance TCP in ANSNET. *SIGCOMM Computer Communications Review*, 24(5):45–60, October 1994.

[94] Wei-Hua Wang, Marimuthu Palaniswami, and Steven H. Low. Optimal Flow Control and Routing in Multi-Path Networks. *Performance Evaluation*, 52(2-3):119–132, April 2003.

[95] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Morley Mao, and Ming Zhang. An Untold Story of Middleboxes in Cellular Networks. In *Proceedings of SIGCOMM '11*, August 2011.

[96] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of NSDI '13*, October 2013.

[97] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *Proceedings of NSDI '11*, March 2011.

[98] Yin Xu, Ben Leong, Daryl Seah, and Ali Razeen. mPath: High-Bandwidth Data Transfers with Massively Multipath Source Routing. *IEEE TPDS*, 24(10):2046–2059, October 2013.

[99] Yin Xu, Wai Kay Leong, Ben Leong, and Ali Razeen. Dynamic Regulation of Mobile 3G/HSPA Uplink Buffer with Receiver-Side Flow Control. In *Proceedings of ICNP '12*, October 2012.

[100] Yin Xu, Zixiao Wang, Wai Kay Leong, and Ben Leong. An End-to-End Measurement Study of Modern Cellular Data Networks. In *Proceedings of PAM '14*, March 2014.

[101] Yang Richard Yang, Min Sik Kim, Xincheng Zhang, and Simon S. Lam. Two Problems of TCP AIMD Congestion Control. Technical Report TR-00-13, Department of Computer Sciences, UT Austin, 2000.

[102] Ossama Younis and Sonia Fahmy. FlowMate: Scalable On-Line Flow Clustering. *IEEE/ACM Transactions on Networking*, 13(2), April 2005.

[103] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of SIGCOMM '91*, September 1991.

[104] Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry Peterson, and Randolph Wang. A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths. In *Proceedings of USENIX '04*, June 2004.

[105] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G. Griffin. Internet Routing Policies and Round-Trip-Times. In *Proceedings of PAM '05*, March 2005.