REDUCING SHORT FLOWS' LATENCY IN THE INTERNET

BY

QINGXI LI

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Associate Professor Philip Brighten Godfrey, Chair
Associate Professor Matthew Caesar
Professor Indranil Gupta
Doctor Nina Taft, Google
Abstract

Short flows are highly valuable in the modern Internet and are widely used by applications in the form of web requests or with user interactions. These kinds of applications are extremely sensitive to latency. A small additional delay, like one or two round trip times (RTTs), may easily cause user frustration and lose usability of services. In the most desirable scenario, we want to finish these kinds of flows in one network RTT. Furthermore, we would like the network’s RTT to be as close as possible to the speed of light. Unfortunately, in the current Internet, there are many unnecessary delays caused by different kinds of policies— in particular, transmission protocol and routing policies—driving us far away from this goal.

This thesis aims at answering the following two questions:

How can we optimize the transmission protocol to reduce short flows’ latency as close as possible to one RTT and why are network RTTs still significantly larger than the speed-of-light latency?

To reduce the transmission latency, we focused on the two main components of short flows, connection establishment and data transmission. ASAP, a new naming and transport protocol, is introduced to reduce the time spent on initial TCP connections. It merges functionality of DNS and TCP’s connection establishment functions by piggybacking the connection establishment procedure atop the DNS lookup process. With the help of ASAP, the host is able to save up to two-thirds of the time spent on initial connection without exposing significant DoS vulnerabilities.

For data transmission, we designed a new control rate mechanism, Halfback, which achieves low latency with limited bandwidth overhead and only requires sender-side changes. Halfback has an aggressive startup phase, finishing transmission for most short flows in one RTT, together with a Reverse-Ordering Proactively Retransmission phase which helps the host to recovery quickly from packet loss caused by the aggressive startup phase. Halfback is able to achieve 56% smaller flow completion time on average and three times smaller in the 99th percentile.

RTT between two hosts is able to be more than 6 times the speed-of-light latency for Directed Optical Fiber. To understand the composition of RTT inflation, we break down the path inflation on the end-to-end path into its contribution factors. Based on our result, 7.2% is caused by network topology, 18.8% is contributed by inter-domain routing policies, 54.9% is caused by peering policies, and 25.6% is caused by intra-domain routing policies. This result shows that the main component of the path infla-
tion is caused by peering policies which may require more attention for future research. Besides this, we also analyze the changes of the inflation caused by each contributing factor across five years. According to our analysis, the total inflation has been reduced by around 6% each year since 2010.
To My parents and Tao Zou.
Acknowledgments

First and foremost, I want to express my deepest gratitude to my advisor, Associate Professor Philip Brighten Godfrey, for initially giving me the opportunity to join this fascinating project and leading me into the world of network and system research, for his invaluable directions and active support during my graduate study at University of Illinois at Urbana-Champaign, for the willing advice whenever needed, and for the encouragement, belief and guidance. Brighten patiently taught me how to demonstrate my ideas, write papers and give presentations which are especially helpful for my future career. I have also learned a lot from his style of collaboration, communication, as well as the attitude to work-life balance. I will be forever grateful for having Brighten as my advisor.

I am especially fortunate to have the opportunity to work with Associate Professor Matthew Caesar for my first research project. He spent countless hours with me in this project, discussing all the details in protocol design with me, helping me to solve all the problems in implementation, and finally deeply understand experimental results. It has been an invaluable experience for me to learn how to implement a large system and do experiments in a worldwide testbed.

I am grateful to have my summer internship in Technicolor Palo Alto Research Center with Doctor Udi Weinsberg and Nina Taft in 2012. Their direction and advice added a new dimension in my research. They gave me many valuable suggestions in writing paper and presenting ideas to some people out of my area. Besides this, Nina also gives me a lot of insightful comments and suggestions for my thesis project. Sincere appreciation is also extended to Professor Indranil Gupta, my thesis committee, for his inspiring advice.

I am thankful to my Microsoft Research mentor Principal Researcher Ranveer Chandra, who provided me an opportunity to work in his group over summer 2011 for low-latency wireless display. He demonstrated how research projects are able to improve the performance of the products for the industry companies. In addition, he also encouraged me to demonstrate my idea to the industry group and cooperate with them.

Thanks also go to the members in the PlaNet group in Google, where I had my summer internship for 2013 and 2015. Specifically, I would like to thank Jiangbo Li, Junda Liu, Victor Lin, Min Tan, Fei Ye, Frank Pan, Shuang Yang, Ke Xiao, Yi Wang, Rui Wang, and Yunjia Dai for their technical help and informative discussion.
I am also grateful to the present and past members of my research group including Wenxuan Zhou, Mo Dong, Rachit Agarwal, Chi-Yao Hong, Ankit Singla, Ashish Vulimiri, Soudeh Ghorbani and Sangeetha Abdu Jyothi. We have been working together for paper submissions, system demos, and technical presentations. I would like to give very special thanks to my friends - Haiming Jin, Channon Chen, Zhenhuan Gao, Hongyang Li, Siting Chang, Tuo Yu, Fan Yang, Shen Li, Shu Shi, Lu Su, Shuo Tang and Guliz Seray Tuncay for their tips and advices throughout my graduate studies.

Last but not the least, I would like to thank my family for their greatest support, unconditional love, without which I could not have come this far. I am also extremely lucky to have my husband, Tao Zou, who always kept me motivated during the journey of my Ph.D. He filled my graduate student life with full happiness and countless memorable moments.
# Table of Contents

List of Tables .......................................................... ix

List of Figures .......................................................... x

Chapter 1  Introduction .................................................. 1
    1.1 Background and Research Problems .............................. 1
        1.1.1 Connection Establishment ............................... 1
        1.1.2 Data Transmission ..................................... 2
        1.1.3 Path Inflation ........................................ 2
    1.2 Research Statements and Approaches ............................ 3
        1.2.1 Reducing Latency for Connection Establishment .......... 3
        1.2.2 Reducing Latency for Data Transmission ................. 4
        1.2.3 Quantifying Path Inflation in Router-level .......... 6
    1.3 Thesis Contributions ......................................... 6
    1.4 Thesis Outline .............................................. 7

Chapter 2  Related Work ................................................. 8
    2.1 Reducing Connection Establishment Time ........................ 8
    2.2 Reducing Short Flows’ Transmission Time ..................... 9
        2.2.1 Startup Phase ...................................... 9
        2.2.2 Packet Loss Recovery ............................... 10
        2.2.3 Bufferbloat ....................................... 10
    2.3 Causes of Path Inflations ................................... 11

Chapter 3  ASAP: Accelerated Secure Association Protocol ............. 13
    3.1 Fast transport connection establishment ..................... 13
        3.1.1 The Role of the Three-way Handshake .................. 13
        3.1.2 Verifying Provenance without a Handshake ............. 14
        3.1.3 Eavesdropping Attacks and Defense ................... 17
        3.1.4 Dealing with Mobility ................................ 18
        3.1.5 Additional Security Properties ....................... 18
    3.2 Fast Name Resolution ....................................... 19
    3.3 Basic Protocol ............................................. 20
        3.3.1 Handling DNS Caching via Multiple Queries .......... 21
        3.3.2 Reducing Latency with Server-side DNS ............... 22
    3.4 Implementation ............................................ 22
    3.5 Evaluation ................................................ 23
        3.5.1 Methodology ....................................... 23
    3.6 Latency Reduction ......................................... 23
    3.7 Computational Overhead .................................... 26
        3.7.1 PV Eavesdropping Defense ............................. 27
    3.8 Deployment ................................................ 29
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Attackability of TCP and ASAP with two PVs. The columns are the topology, attackability of ASAP with optimally-placed PVs, attackability of TCP, and the chance that ASAP’s attackability is less than TCP’s if the PVs are placed randomly.</td>
</tr>
<tr>
<td>4.1</td>
<td>Different kinds of startup phase and lost packet recovery schemes.</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Overview of ASAP ........................................ 4
1.2 Tradeoff between common case latency (y axis) and feasible capacity in a pessimistic case of high workload from short flows (x axis). .... 5

3.1 Key messages in the basic ASAP transport protocol. Obtaining a Provenance Certificate when acquiring a certain IP address (messages 1 and 2); opening a transport connection (message 3). ............... 15
3.2 Timeline for TCP and three variants of ASAP. .................. 20
3.3 Overall download time, when DNS caching is (a) enabled, bypassing the ADNS; (b) disabled, with the ADNS collocated with the server; and (c) disabled, with the ADNS in a random location. ....................... 24
3.4 Latency improvement of ASAP’s transport layer, kernel implementa-
tion, excluding name resolution. ................................ 25
3.5 Cryptographic overhead. ...................................... 27
3.6 CDF (over PV locations) of the percent of IP addresses that are attack-
able in ASAP, based on Route Views routing data. ................... 28

4.1 CDF of fraction of traffic carried by different size of flows using measurements from [88] [35] [59] ........................................... 32
4.2 Halfback transmits a 10-packet flow .......................... 36
4.3 The configuration of the experiments in Emulab .................. 37
4.4 The number of normal TCP retransmissions of short flows in Planetlab experiments ......................................................... 38
4.5 The flow completion time of short flows in Planetlab experiments ... 39
4.6 The number of RTTs used in the transmission of short flows in Planet-
lab experiments ...................................................... 40
4.7 CDF of FCT under cases where packet loss happened ............ 41
4.8 CDF of FCT on home networks with different providers ........... 42
4.9 The performance of short flows for different router buffer sizes ... 43
4.10 Flow completion time for different flow size under 25% network uti-
лизation for different network traffic distributions ...................... 44
4.11 The performance of different mechanisms for different network utiliza-
tions while there are only short flows ............................ 46
4.12 Flow Completion Time normalized by the Flow Completion Time of TCP for different network utilizations with 10% of traffic created by short flows and 90% by long flows ...................... 47
4.13 TCP-friendliness of different non-TCP mechanisms .............. 48
4.14 Throughput of flows for (a) Optimal situation (b) Halfback (c) One TCP (d) Two TCP flows with half flow size .................. 49
4.15 Average response time for different mechanisms under different net-
work utilization ...................................................... 50
4.16 FCT and feasible capacity of mechanisms with different startup phases and lost packet recovery mechanisms. .......................... 52

5.1 The path inflation caused by different reasons in 2014 December based on the geolocation information provided by IPLG................ 55
5.2 The traceroute inflation calculated by different geolocation databases. 58
5.3 The path inflation caused by different reasons in 2014 December based on the geolocation information provided by MML............... 59
5.4 The path inflation caused by different characters for past five years. 60
5.5 The distance and number of hops reduced by sibling relationship for the routers in the destination city. ............................. 62
5.6 The first and last AS hop’s path inflation caused by different peering policies for routes with two As hops. ............................ 64
5.7 The path inflation caused by optimal peering policy. .................. 64
Chapter 1

Introduction

1.1 Background and Research Problems

Short flows are highly valuable in the modern Internet. They drive many important interactive networked applications, with web browsing the most prominent example. In the current Internet, around 99% of flows carry traffic less than 100 KB [88]. For this kind of flow, the user-perceived latency is a key challenge since even a relatively small delay is able to cause user frustration, loss of usability of services, and reduction of the number of customers and revenue [46, 83, 94, 78]. A recent study by Google [38] shows that an additional delay as small as 100 milliseconds measurably reduced users frequency of conducting searches; the effect increased over time, to a 0.74% drop after 4-6 weeks of a 400 ms delay, and persisted for weeks after the artificial delay was eliminated. Besides this, the revenue per person in Bing was reduced by 1.2% with a 500-millisecond delay and 4.3% with a two-second delay [100].

1.1.1 Connection Establishment

The main components of short flows are connection establishment and data transmission. Compared with long flows, connection establishment time is more important for short flows since the total number of Round Trip Times (RTTs) used by short flows are small. For example, downloading 11.18 KB (the median size of content located on a single host [12]) from a server at the United States average connection speed, 4.6 Mbps [23], ideally needs 30.1 millisecond for data transmission. However, before this, in current protocols, the client first needs to perform a Domain Name System (DNS) lookup, and execute TCP’s three-way handshake (3WH). This whole process may cause one to two additional RTTs. Since RTTs are commonly on the order of 50-100 millisecond [?] and the client may need to initiate multiple TCP sessions in serial to download the page, the overhead for current connection establishment is able to account for a significant fraction of the total delay. From our experiments, the median latency caused by 3WH was 72 millisecond, and by DNS was 101 millisecond for the 100 most popular U.S. websites [1]

There are many projects proposed to reducing latency spend on connection establishment. DNS latency is able to be reduced with caching and a lot of work tried to improve the performance. [87, 49] try to use cooperative caching and lookup to speed up DNS in isolation. [45] uses proactively caching to alleviate DNS resolution delay.
TCP Extensions for Transactions (T/TCP) [37, 73] first tries to reducing the connection establishment time by eliminating the TCP’s 3WH. But, due to its vulnerability of Denial-of-service (DoS) attacks, it is not widely deployed. Most work separate this problem as two research problems: reducing TCP’s connection establishment time and speeding up DNS performance. However, combining them as one problem is able to obtain additional gains.

1.1.2 Data Transmission

The second important component for short flows is data transmission. The Internet’s current transmission protocol, Transmission Control Protocol (TCP), tries to use bandwidth conservatively to achieve safety for all flows. However, at the same time, it sacrifices the latency of short flows due to its startup scheme and slow packets recovery mechanism. TCPs slow-start needs multiple RTTs to detect its fair sending rate. Many short flows even cannot leave this startup phase before finishing transmission. The situation becomes even worse because TCP also needs at least one RTT to detect and recover from packet loss. When there are not enough packets to generate duplicate ACKs or when the retransmitted packets are lost, the sender times out, waiting typically one or more RTTs. As a result, the completion time of a 32 KB flow from major web sites to well-connected (PlanetLab) clients on the Internet is around 8.7 RTT [99] which is far away from ideal. Finally, the above problems are magnified by bufferbloat in which large router buffers lengthen RTT.

Many mechanisms have been proposed to optimize short flow transmission time. Some require protocol changes and in-network router support, such as RCP [39], RC3 [79] and QuickStart [96], and therefore have not seen any significant deployment. Others, such as Proactive TCP [53], increasing the initial congestion window to 10 [48, 25] and JumpStart [75] focus on sender-only changes. It is challenging to design a sender-only mechanism for reducing latency because a sender has very limited information about the network at the beginning of a flow, and has to effectively guess the best way (i.e. starting rate and retransmission policy) to transmit data quickly. All the aforementioned sender-side optimizations choose to send more aggressively at the initial start-up phase to reduce flow completion time (FCT). However, sending aggressively will inevitably impact performance of other flows and indeed, all flows—even the aggressive ones may suffer if short flows dominate the utilization of the network. We refer to these as safety concerns. The key problem is to walk the delicate latency-safety tradeoff space and find the sweet spot.

1.1.3 Path Inflation

Previously we focused on the total number of RTTs used by short flows, but besides this, the latency of RTT is also important for short flows. The Direct Optical Fiber (DOF) latency between New York City (in the northeast of the U.S.) and San Francisco (in the southwest of the U.S.) should be around 20 millisecond; however, the RTTs in the U.S. are commonly on the order of 50-100 millisecond, which is far from the
DOF latency. Many factors are able to cause this kind of path inflation. For example, regarding a route from Gatineau, Canada, to a small city near Akola, India, since there is no DOF connecting Canada and India it needs to travel to New York City first, then go to Schweiz in Europe, and finally jump to India. This kind of detour will increase a small amount of latency compared with DOF latency. In addition, routing protocols, traffic engineering, and ISP policies are also able to increase the latency between two hosts.

This kind of latency/path inflation has been seen for more than ten years. In the past decade, a lot of researchers tried to figure out the contributors and the extent they contribute to this inflation. Most of these works can be separated into two categories: router-level and AS-level. Most router-level analyses [76, 106] were proposed more than ten years ago. For example, [101] published in 2003 quantifies the causes of path inflation into three levels and two categories and creates its own model for intra-domain routing protocol. Other router-level analyses only focus on a small network, such as [77] that only analyzes the inflation in the Spanish academic network. AS-level projects [43, 57, 81] only analyze path inflation based on the AS-level topology of the network. Some of the work [43, 107] directly uses the real routing information collected from the Internet. The others use the valley-free and prefer-customer policies as BGP routing policy. Since compared with router-level analyzing, they ignore a lot of topological information for the network, they miss some contributors of the path inflation.

1.2 Research Statements and Approaches

This paper studies the following questions:

- How can a transport protocol be designed to perform transactions of short flows with latency as close as possible to a single RTT?
- How are we far away from speed-of-light latency? What are the components of the path inflation in latency for one round trip time?

With the answers of these two questions, we are able to save a certain amount of RTTs for short flows’ establishment and transmission. In addition, we also have a better understand about the path inflation and points out the future direction to achieve speed-of-light Round Trip Time.

To achieve these, we separate these two questions into three parts, reducing connection establishment time, reducing data transmission time and analyzing the contributors of path inflation.

1.2.1 Reducing Latency for Connection Establishment

While DNS latency can be reduced with caching or employing a distributed set of DNS servers reachable via anycast, in many cases [71] the lookup will have to visit an authoritative DNS server that is relatively distant. In our project, we cut the latency
by as much as one RTT by piggybacking the first transport packet within the DNS query. The DNS server directly sends this packet to the server instead of sending DNS response back to client and let client send the packet to server (Fig. 1.1) Although related DNS shortcutting techniques have been proposed before [72], we show that it can be done with modifications to only the client and the local DNS server, without requiring changes in the global DNS infrastructure.

Besides this, we introduce a protocol which eliminates the need for TCP’s 3WH, saving another RTT per connection. Since 3WH is an important part of TCP, it allows the server to verify the provenance (source address) of a client’s request, to guard against denial of service (DoS) attacks. To avoid having the same vulnerability as T/TCP, our design ensures verifiable provenance while avoiding the need for a handshaking step on every connection. A client obtains a certificate of provenance for its current location, by handshaking with multiple provenance verifiers (PVs) to limit eavesdropping attacks. The client includes this certificate with future transport connection requests. The server can then locally verify the certificate and send a response without waiting for a traditional handshake.

1.2.2 Reducing Latency for Data Transmission

To reduce the data transmission time used by short flows, we evaluate existing solutions in terms of their latency performance and more importantly their latency-safety trade-off. We evaluate both flow-level and application-level benchmarks. At the flow level, we benchmark the latency with flow completion time (FCT) and benchmark safety with two metrics: feasible network utilization, which is defined as the maximum network utilization achievable before the throughput collapses, and TCP friendliness. At the application level, we benchmark latency with web request response time and benchmark safety with feasible network utilization with flow arrival process adhering to the real application requests patterns.

We evaluated normal TCP and five existing solutions head-to-head: Proactive TCP,
Figure 1.2: Tradeoff between common case latency (y axis) and feasible capacity in a pessimistic case of high workload from short flows (x axis).

Reactive TCP [53], increasing initial congestion window to 10 [48, 25] (referred to as TCP-10 in this paper), PCP [28] and JumpStart [75] (Fig. 1.2). JumpStart and TCP-10 are closer to the trade-off frontier than the other two, but are still not good enough. TCP-10 is still too conservative in many cases and renders both long FCT and slow web response time even when the network load is low. JumpStart achieves better flow-level latency performance by pacing out all the data at first RTT. However, after the first batch of data gets paced out, it falls back to normal TCP with bursty and reactive-only transmission. As we show in §4.3, the bursty retransmission makes JumpStart too aggressive and thus renders unsatisfying safety benchmarks: it has low flow-level feasible network utilization, which makes its application-level web response time unacceptable even under median network load, and impacts TCP friendliness. Moreover, JumpStart also has suboptimal latency performance by only relying on reactive packet loss detection.

Based on the evaluation of existing solutions, we propose a new short-flow transmission optimization mechanism, Halfback, that improves both latency and safety at both the flow level and the application level. Halfback borrows the initial starting phase from JumpStart: pacing out packets within one RTT for short flow. However, Halfback rises where JumpStart fails with a novel Reverse-Ordered Proactive Retransmission (ROPR) mechanism to improve reactivity to packet loss and improve safety by limiting aggressiveness at retransmission. ROPR proactively retransmits packets in reverse order (starting at the end of the short flow) at the rate of receiving ACKs.
1.2.3 Quantifying Path Inflation in Router-level

In this work we want to analyze the path inflation in two directions. The first direction is the same as in previous works: we want to figure out the extent to which each factor inflates the latency. To achieve this, we separate the contributors into two categories, network topology and routing policies. For the routing policies, we further separate them into three parts: the policies used to select the path within an ISP (intra-domain routing policies), those used to select the peering links to reach neighboring ISPs (peering policies), and those used to select the path of sequence of ISPs to reach the destination (inter-domain routing policies). This kind of result is able to help the future researchers to gain insight when they design new networks, routing policies, or ISP policies.

The other direction is how the path inflation changed across multiple years, especially for the inflation caused by different factors. Since path inflation was discovered, a lot of researchers have paid attention to it and dozens of projects proposed to analyze [101, 76, 43, 57] or reduce [44, 29] this inflation. However, up to now, no one has tried to measure how much improvement has been made and how much inflation has been reduced in the past a few years. In this work, we quantify the inflation caused by the same possible factors as noted above for five years. To make the results comparable, we use the same data resource for network topology, AS relationship, router geolocation information, and route information for different years. Based on these results, we are able to figure out how the inflation has changed during the past five years and how much improvement we have made.

1.3 Thesis Contributions

Our major contribution is designing two protocols, ASAP and Halfback, to achieve flow completion time as close as possible to one RTT. In addition to this, by quantifying the contributors of path inflation in router-level, we point out the future research direction to reducing short flows’ latency further. Other contributions are as below:

- ASAP is the first transport protocol that can complete requests in a single RTT without exposing significant DoS vulnerabilities. With the help of ASAP, the connection establishment time is cut by up to two-thirds. Based on our evaluation for DoS vulnerabilities, compared with TCP, when there are two provenance verifiers used to prove the clients’ location, ASAP has 91.41% chance being less vulnerable than TCP if the verifiers are randomly located. If we are able to choose the location of the verifiers, in the worst case, there are less than 5% of the client-server pairs may be attacked by the attacker.

- We evaluate Halfback with both flow-level and application-level benchmarks. Based on the result of our evaluation, Halfback reduces latency by 13% (21% in the 25% of cases where there is packet loss) compared to JumpStart and is 29%, 61%, 51% and 52% better than TCP-10, Proactive TCP, Reactive TCP
and vanilla TCP. Further, we also evaluate Halfbacks FCT-vs.-safety tradeoff compared with existing approaches. The results show that compared with Jump-Start (the FCT winner among past proposals), Halfback achieves 19.25% lower FCT, significantly improves the feasible network utilization by $1.4 \times$, and also improves TCP friendliness for both long and short TCP flows. Finally, we compared the web response time by using Halfback and other proposals with realistic website data and request patterns. Because Halfback operates on a better flow-level trade-off point, as shown in Fig. 1, it achieves significantly better application-level latency-vs.- safety tradeoff with 592 millisecond (22%) page load time reduction at 30% network utilization and significantly improves feasible network utilization by $1.57 \times$ comparing to JumpStart.

- We evaluate the contribution of four possible causes of path inflation—network topology, intra-domain routing policies, peering policies, and inter-domain routing policies—in router-level based on the data set of December 2014. According to our results, the main contributor of path inflation is peering policy, which increases the inflation by $1.62 \times$, in the median. For the other three causes, in median, network topology increase the latency by 0.82%, intra-domain routing policies increases it by $1.29 \times$, and inter-domain increases it by $1.21 \times$.

- Ours is the first work which evaluates the changes of path inflation over a period of around five years. In the past few years, the inflation caused by network topology has been reduced by 0.16% in the median from July 2010 to December 2014. This also helps to reduce the inflation caused by the other factors since one more link may change the shortest path with different policies. To summarize, in median, for the past five years, the inflation caused by inter-domain routing policies, peering policies, and intra-domain routing policies has been reduced by 8.82%, 20.45%, and 5.88%, correspondingly.

### 1.4 Thesis Outline

This remainder of this dissertation is organized as follows. In Chapter 2 we discuss related works which tries to reduce short flows’ latency or quantify the path inflation in the Internet. Chapter 3 introduces the new protocol ASAP used to save the time spend on connection establishment. Chapter 4 shows how we reduce the transmission time for short flows by Halfback. In Chapter 5, we analyze the causes of path inflation and how it is changed across recent five years.
Chapter 2

Related Work

2.1 Reducing Connection Establishment Time

Several previous work aim to bypass TCP’s 3WH to improve its performance for connection establishment. However, without carefully designed, these protocols cannot gain widespread popularity due to their vulnerability of malicious attacks, such as SYN flood, amplified reflection attacks and so on. In this section, we introduce some previous work which try to reducing the connection establishment time by eliminating TCP’s 3WH and their security mechanisms which used to avoid previous attacks.

TCP Extensions for Transactions (T/TCP) [37, 73] proposed in 1994 is the first work which tries to bypass TCP’s connection handshake. Instead of mitigating the security vulnerabilities caused by eliminating TCP’s 3WH, T/TCP mainly focuses on combating old and duplicate SYNs. In T/TCP, each client-server pair has a monotonically increasing variable, Connection Count (CC). This variable helps the server to figure out the old or duplicate SYN requests. However, since CC value is monotonically increasing, it is easy for an attacker to fake a CC value which is able to pass the test of the server. This leaves T/TCP open for all kinds of attacks we mentioned before, as discussed in [62].

TCP Cookie Transactions (TCPCT) [98] is an extension of TCP proposed to secure TCP connections. Unlike T/TCP which is designed to reduce latency, TCPCT is primarily designed to eliminate server state during 3WH and avoid DoS attacks. The TCPCT-enabled hosts use cookie exchange to negotiate elimination of server state. Since the cookie pair is much too large to fit with the option space in TCP header, TCPCT allows adding space in SYN packets for additional options while let hosts exchange limited amounts of data during the 3WH. In this case, the server sider client needs to send the response in the SYN-ACK packet back to the client immediately after receiving data in the SYN packet. Otherwise, it can only send the response after 3WH finished. Besides this, TCPCT also constrains the size of server’s response to one packet.

TCP Fast Open (TFO) [89] is a more recent alternative of T/TCP. The same as TCPCT, TFO also uses cookies to eliminate server state and allows SYN and SYN-ACK packets carry data. Unlike TCPCT which limits the size of the server’s response, TFO allows the server to send data back during the handshake even after SYN-ACK sent. TFO uses the same duplicated SYN handling mechanism as unmodified TCP
which means that it allows the duplicate and old SYN with data. This makes TFO simpler and more appealing to implement. However, it also leaves TFO open to SYN flood attacks and the damage caused by this kind of attacks is much worse due to the data carried in.

In addition to reducing connection establishing time from 3WH, there are also some previous work try to reduce time from DNS resolution [87, 72]. Most of them are designed to improve the caching performance, for example, [45] proactively cache DNS records and [87, 90] use cooperative caching and lookup to speed up DNS in isolation. Besides this, DEW [72] also explores ways by which a variety of Web requests and responses could be piggybacked on DNS messages, similar as ASAP. However, compared with ASAP which only requires modification for the LDNS and leaving the global DNS infrastructure untouched, DEW needs to modify both LDNS and ADNS which increases its difficulty of widely deployment.

2.2 Reducing Short Flows’ Transmission Time

In this section, we discuss work related to the challenges of reaching a high sending rate quickly, dealing with loss, and bufferbloat.

2.2.1 Startup Phase

Many projects have been proposed to accelerate the startup phase of TCP. These can be separated into five categories.

Aggressive startup: RC3 [79] uses a high initial sending rate and requires routers to support priority queues to avoid negative effect. [48, 25] propose higher initial congestion window size, and [75] proposes JumpStart, all of which we have evaluated in

Sharing information between connections or hosts to make flows start at a more appropriate rate: These mechanisms need complex cooperation [34] or even additional bandwidth [36]. They also increase the difficulty of deployment by introducing a new protocol.

Bandwidth estimation [41, 50] may not be accurate as end hosts lack real-time visibility into the network unless the routers explicitly mark available bandwidth in the header [96] which increases the difficulty of deployment. We experimentally evaluated one bandwidth estimation scheme [28].

Caching schemes [84] will draw back to Slow-Start when the variables are aged and need careful tuning of their variables. We tested a caching scheme and even under optimistic conditions, Halfback outperformed it in terms of latency, albeit with more bandwidth overhead. While caching performs better than TCP, it still may not pick the optimal window size, and does not improve packet loss response.

Faster connection setup mechanisms like TCP Fast Open [89] and ASAP [111] focus
on reducing the time used in the three-way handshake of TCP connection establishment. While connection setup time is a fairly large portion of short flows’ lifetime, the handshake is orthogonal to Halfback’s optimization mechanism. Halfback focuses on reducing the number of RTTs used for the actual data transfer of short flows and therefore, any of the connection establishment optimizations can be a drop-in replacement for Halfback’s connection establishment process. All experiments in §4.3 include the connection setup time without optimization.

2.2.2 Packet Loss Recovery

Short flows are sensitive to packet loss. The time spent by TCP for detecting packet loss and retransmitting the lost packets is undesirable for short flows. [109, 66] reduce packet loss at the last RTT of TCP’s slow-start by choosing an appropriate slow-start threshold, ssthresh. As they are based on bandwidth estimation, the inaccuracy of the bandwidth estimation causes inaccurate ssthresh estimation. Besides this, these schemes offer no help to short flows that are too short to leave the slow-start phase, which is a very common case.

Proactive retransmission [53] retransmits some packets before receiving the signal of packet loss. These mechanisms try to save time used to detect packet loss and avoid a timeout when the retransmitted packet is lost or the packet loss happens at the end of the flow. However, the proactively retransmitted packets require extra bandwidth from the network. This additional bandwidth reduces the scheme’s feasible capacity and may cause problem to the co-existing flows or the whole network. We have quantitatively evaluated the proactive scheme of [53] in §4.3.

2.2.3 Bufferbloat

Bufferbloat [58, 42] happens when routers have large buffers that cause long queuing delay, increasing the reaction time of packet loss and causing large latency for short flows.

Many AQM algorithms [54, 52, 64] are used to solve this problem. These protocols are more focused on the queue length than on the queuing delay and needed careful variable tuning for different networks which avoids them being widely deployed. Later AQM algorithms, most recently PIE [86] and CoDel [82], are directly focused on queuing delay. However, the efficiency of the routers may be reduced since they requires timestamps on every packets.

In additional to AQM algorithms, some protocols, such as DRS [51] and DRWA [70], reduce the effect of Bufferbloat on the end host. In DRS [51], the receiver automatically adjusts its buffer size to twice the estimated congestion window size of the sender. Many operating systems use similar auto-tuning algorithms as DRS, like Linux since kernel 2.4.27, Windows since Vista. As this kind of auto-tuning is unidirectional, it can only mitigate the bufferbloat problem in some situations. DRWA [70], used in cellular networks, adjusts the receiver’s buffer size bidirectionally to keep the
queue size within a proper range. It is able to reduce the delay by 25% to 49% in general cases and increase TCP throughput by up to 51%.

2.3 Causes of Path Inflations

Path inflation was first found around 15 years ago [101]. Since then, this phenomenon has received a lot of attentions from researchers. Many projects analyze path inflation and explain the causes of it. The works can be separated into two categories: router-level [76, 106, 101, 77] and AS-level [81, 107, 43, 57].

[101] tries to quantify path inflation based on router-level map got from traceroute. In their paper, they identify six possible causes of path inflation which are separated into two categories: network topology and routing policy, at three layers: intra-domain, inter-domain and peering. Besides this, the authors also create a new intra-domain routing model according to the routing information collected. Based on their result, intra-domain traffic engineering has minimal impact on path inflation and peering policies and inter-domain routing lead to significant inflation. Especially for inter-domain routing, as BGP uses minimum AS-hop count to break the tie, around 50% of the paths are longer than the shortest available path.

[106] also uses router-level network topology and tries to analyze the changing of path inflation caused by the BGP protocol from April 2000 to May 2001. In their work, the authors use early-exit peering policy together with valley-free and prefer-customer policies to generate router-level policy path and compared the distance rate of this path over the shortest router level path in 2000 with that in 2001. According to their result, the inflation caused by this policy is nearly the same from 2000 to 2001.

Instead of considering all the nodes equally, [77] assigns a weight to each node based the volume of traffic amount. The weighted path inflation is able to be used by the network operators to decide where to reduce the path inflation, either the paths with large path inflation or the paths which are important due to traffic volume. However, since this weighted path inflation needs to collect traffic information for all the host, it is very difficult to analyze it for large networks. In this paper, the authors only focus on the Spanish academic network.

In addition to router-level path inflation, some research projects analyze AS-level inflation. According to the routing information used, they are separated into two categories. The works[107, 43] in the first category directly use the routing information collected from the Internet. For example, [107] uses Scriptroute to collect data from 363 PlanetLab nodes across 30 countries. Instead of using a snapshot of routes in the Internet, this work collects the routing information from March 2007 to April 2007 and use it to figure out how the AS-level path inflation changed in a short time period. Based on their results, the largest difference in AS path inflation on a particular connection is able to be as high as 6, and the average is 2.5. Besides this, some nodes experienced different paths over 70% of the experimental time.

For the second kind of projects [81, 57], instead of using the real routing information, they create their own AS-level routing policy model. The most commonly
used policies are valley-free and prefer-customer policies. In [81], the authors have two scales analyzing. Internet-scale for AS-level and small-scale whose topology and routes information are generated based on different variables, such as the routing policy used, the AS size, the number of peering links between ASs. For Internet-scale, they just use valley-free and prefer-customer policies to simulate BGP routing. For small-scale, this paper analyzed how the variables is able to affect the path inflation in the network. This work demonstrates that when we design the future Internet, how we are able to reduce the Path Inflation.

The router-level works, like [76, 106, 101], which analyze the large-scale networks are conducted around ten years ago and may not accurately represent the current path inflation situation in the current Internet due to the quick development of the Internet in recent decades. Recent measurements of path inflation [77, 81, 107, 43, 57] either only consider a small network [77] or mainly focus on AS-level networks. They cannot help us to have an idea about the path inflation in the whole Internet including all the routing policies and physical layer situations.
Chapter 3

ASAP: Accelerated Secure Association Protocol

3.1 Fast transport connection establishment

In this section, we present ASAP’s core transport connection establishment protocol, which eliminates the 3WH while guarding against DoS attacks. We begin by revisiting the motivation for the 3WH.

3.1.1 The Role of the Three-way Handshake

In TCP’s 3WH, the client sends the server a SYN packet containing an initial sequence number (ISN); the server acknowledges this with a SYN-ACK including its own ISN; and the client ACKs the server’s ISN. The client can then begin sending data (such as an HTTP request). The 3WH thus adds one RTT of delay.1

There are two primary benefits of the 3WH. We discuss each, and how they fit into ASAP’s design.

A. Idempotence

The 3WH was originally [105] designed to ensure a form of idempotence: if a packet is retransmitted or duplicated in the network, it should not cause a connection to be opened more than once. By challenging the client to echo back a pseudorandom number (the ISN), the 3WH verifies that the client’s request is still current.

We argue that this transport-layer idempotence is neither sufficient nor necessary for applications’ needs. First, it is not sufficient by an end-to-end argument: transport-layer idempotence does not ensure end-to-end idempotence. If a higher-layer entity retries the request, such as when a human clicks on a link twice after the server appears to respond slowly, the transaction may be executed twice. As a result, some web sites resort to imploring the human user, “Do not click Submit twice!”

Second, transport-layer idempotence is not necessary: an application that desires this property can simply perform a handshake at the higher layer. Moreover, many applications do not need it. If a server delivers a web page twice a very small fraction of the time, this is only a slight inefficiency, rather than a correctness problem.

1The client can include data with the first SYN packet. But if the server acts upon this data before receiving the client’s ACK, then the functionality of the 3WH is nullified.
Therefore, we argue that the benefit of ensuring idempotence in a general-purpose transport protocol does not justify its cost in added delay. ASAP will however make idempotence violations unlikely (§3.1.2.D).

B. Denial of Service Protection

The 3WH also lets the server test the provenance of a client request from some source IP \( c \). The fact that the client is able to echo the server’s pseudorandom ISN, is a reliable indicator that the client is located at \( c \).

If the server completes requests without waiting for the 3WH, two DoS vulnerabilities emerge. First, the client could fabricate a large number of requests from many spoofed IP addresses, making it difficult for the server to filter requests from a single attacking client. Second and more critically, the attacker could perform reflection/amplification attacks: it sends relatively small requests with the source address set to a victim’s address. The server then sends a larger amount of data to the victim, thus amplifying the attacker’s power and hiding the origin of the attack.\(^2\)

One could hope that ISP networks perform egress filtering to block source spoofing. However, security is preserved only if all networks across the Internet choose to perform filtering and do so without bugs. This idealistic assumption is false in practice [30].

We conclude that the DoS protection afforded by the 3WH is highly valuable. The main goal of the rest of this section is to develop a protocol to verify source provenance without introducing an RTT delay.

3.1.2 Verifying Provenance without a Handshake

ASAP leverages cryptographic proof to verify the provenance of client requests without requiring an RTT delay on every connection. First, the client handshakes with a provenance verifier (PV) to obtain a provenance certificate (PC). The PC corresponds to cryptographic proof that the PV recently verified that the client was reachable at a certain IP address. After obtaining this certificate once, the client can use it for multiple requests to place cryptographic proof of provenance in the request packet sent to servers, in a way that avoids replay attacks.

This subsection presents our basic provenance verification protocol. Subsequently, we will deal with two subtle problems: eavesdropping near the PV (§3.1.3) and mobility (§3.1.4). Fortunately, those two refinements only require changes to the process of obtaining a PC.

A. Choosing a Provenance Verifier

The PV may be any party trusted by the server. We envision two common use cases.

First, the PV may simply be the web server itself, or a PV run by its domain at a known location (pv.xyz.com). The first time a client contacts a domain, it obtains a

\(^2\) Even with a 3WH, an attacker can reflect a SYN-ACK packet off a server, but no significant amplification occurs.
Figure 3.1: Key messages in the basic ASAP transport protocol. Obtaining a Provenance Certificate when acquiring a certain IP address (messages 1 and 2); opening a transport connection (message 3).

PC from the PV prior to initiating the application-level request to the server; thereafter, it can contact the server directly. Thus, the first connection takes two RTTs (as in TCP), and subsequent connections require a single RTT. This technique will be highly effective for domains that attract the same client frequently (even if the specific server varies each time), such as popular web sites or content distribution networks.

Second, one or more trusted third parties could run PV services. The advantage is that a client can avoid an RTT delay for each new server or domain. The disadvantage is that servers need to trust a third party. But this is not unprecedented: certificate authorities and root DNS servers are examples in today’s Internet.

The above two solutions can exist in parallel. If the client uses a PV the server does not trust, ASAP falls back to a 3WH and can use an appropriate PV for future requests.

B. Obtaining a Provenance Certificate

The protocol by which a client obtains a PC is shown in Fig. 3.1. Before beginning, the client and PV have each generated a public/private key pair ($K_{pub}^c/K_{priv}^c$ and $K_{pub}^pv/K_{priv}^pv$ respectively) using a cryptosystem such as RSA. The client then sends a request to the PV:

$$\{K_{pub}^c, d_c\}$$
where $d_c$ is the duration for which the client requests that the PC be valid. The PV replies with the PC:

$$PC = \{K_{pub}^c, a_c, t, d\} K_{priv}^c.$$  

Here $a_c$ is the source address of the client, $t$ is the time $PC$ becomes valid, and $d$ is the length of time $PC$ will remain valid. The PV sets $t$ to be the current time, and sets $d$ to the minimum of $d_c$ and the PV’s internal maximum time, perhaps 1 day (§3.1.4).

To verify provenance (§3.1.1.B), it is sufficient to use a single UDP message: while it doesn’t prove to the PV that the client can receive messages at $a_c$, the client can only use the PC if it is able to receive it at $a_c$. However, the PV itself is somewhat better protected from DoS by using TCP, especially with SYN cookies, since this ensures that the PV checks for address spoofing before it performs cryptographic functions (which are expensive relative to sending a SYN-ACK).

C. Sending a Request

Once the client $c$ has a current PC for its present location, it can contact a server and include the PC in its request in order to bypass the 3WH.

However, a naïve implementation including only the PC would allow anyone who obtains the PC (an eavesdropper or a malicious server that $c$ contacts) to use it to induce any server to send data to $c$. To guard against this attack, $c$ also constructs a request certificate (RC) encrypted with its private key:

$$RC = \{\text{hash(meta, data)}, t_{req}\} K_{priv}^c.$$  

Here $\text{hash}$ is a secure hash function, $meta$ is the message metadata (source and destination IP address and port, protocol number, initial sequence number), $data$ is the application-level data (such as an HTTP request), and $t_{req}$ is the time the client sends the request.

The client can now open a transport connection to the server with a message of the form:

$$meta, PC, RC, data.$$  

Upon receipt, the server verifies validity of the request. To do this, the server must already know the public key of each PV that it trusts. It determines whether the PC is valid for one of these by checking that it decrypts correctly, the current time lies within $[t, t + d]$, and $a_c$ matches the source address. If so, it uses the client’s public key $K_{pub}^c$ from $PC$ to check that $RC$ decrypts correctly, the hash value in $RC$ matches $\text{hash(meta, data)}$, and the time $t_{req}$ is recent, e.g., within the last 5 minutes. (This timeout only needs to be long enough to cover most clock inaccuracy, which is on the order of hundreds of milliseconds on NTP-enabled hosts, and packet transit time.)

If all these tests pass, then the request is accepted and the connection proceeds as in TCP after the 3WH: data is passed to the application, and data (e.g., a web page)

---

\[\text{We expect the number of trusted PVs to be small, but to avoid iterating through each, the message could simply include a short identifier for the relevant } K_{pub}^p.\]
may be sent immediately back to the client. Thus, the client can receive results within a single RTT.

D. Errors and Idempotence

In the protocol above, a number of errors can occur. Clocks at the PV, client, or server could be out of sync so that certificates are rejected, or the PV chosen by the client may not be trusted by the server. In these cases, we can simply fall back to a regular 3WH.

Another error case is sending of duplicate requests, most commonly when the client retransmits after a timeout. There are several cases. If the server has not received the request yet (e.g., the first was dropped), then it proceeds as in the protocol above. If the server has already received the request and the connection is still active, then it can realize this and simply ignore the duplicate. If the server has already received the request but the connection is closed, it may believe the request is new, and pass the data to the application.

The last case violates idempotence (§3.1.1.A). It is unlikely to occur in benign cases: closing the connection requires receipt of a FIN packet from the client, which it would only send after a retransmission of the original request, and even after that the TCP stack enters the TIME_WAIT state before finally clearing the connection after a timeout. However, the server may receive the connection request after that due to a replay attack. The server may choose to guard against such cases by remembering hash values of recent requests. The server needs to maintain this state only for the RC timeout period (5 minutes as specified above).

3.1.3 Eavesdropping Attacks and Defense

Thus far, determining the validity of a client’s address hinges on the client being able to receive messages at a given address \( a \). But in fact, this depends not only on the client’s location, but also on from where the message is sent. If an attacker can eavesdrop on any part of the path \( PV \to a \), then it can obtain a PC for \( a \). In the same way, a client can induce a TCP server \( s \) to send data to \( a \) if it can eavesdrop on any part of the path \( s \to a \). Therefore, if the PV is colocated with the server, ASAP’s security (in this sense) is equivalent to TCP’s.

But if a single PV is used by servers in many locations, the attack could be more damaging. Consider, for example, a PV run by a globally-trusted third party. An attacker who can eavesdrop on the PV’s network providers can obtain PCs for any address, and these PCs are valid at every server in the Internet!

To defend against this attack, we use the following technique: the organization running the PV service places PV servers in several diverse locations; the client must successfully handshake with all of them to obtain a PC. In this case, the attacker would need to eavesdrop on all paths \( PV_1 \to a, PV_2 \to a, PV_3 \to a \), traversing diverse geographical locations. Intuitively, the attacker has either compromised many networks or is in fact physically close to \( a \). Note that this change only requires modification of the
protocol between the client and the PV, with no changes to the PC format or interaction between client and server.

But how many PVs are necessary, where should they be placed, and how much (quantitatively) can the eavesdropping attack be limited? We give two answers to this question.

First, in the Appendix, we prove that $O(k \log n)$ PVs are sufficient (though perhaps not necessary) to defend against an attacker that can eavesdrop on $k$ nodes in an $n$-node network, even if the attacker can choose those nodes after knowing the PV locations. Specifically, we show that for nearly all possible PV placements, for every client $c$, either: (1) the attacker cannot impersonate $c$ in ASAP, or (2) the attacker might be able to impersonate $c$ but even in TCP, the attacker could fool at least half of the Internet into believing that it is $c$.

Second, in §3.5, we show that the situation is even better in practice. In real-world networks, even with 2 PVs, for the large majority of PV placements ASAP provides better protection against a worst-case eavesdropping attacker than TCP.

### 3.1.4 Dealing with Mobility

In our basic protocol, PCs remain valid for a fixed timeout, such as 1 day. The fixed timeout may be suitable for many applications. In some cases, however, a client may be authorized to use a source IP for only a short duration, perhaps because it is mobile, and we may wish to bound the duration of invalid use of a PC.

Suppose a client is authorized to use an IP address only for time period $[t, t + T]$. The difficulty is that the PV does not know $T$. A simple approach would be to pick a fixed PC timeout $d$. This results in a tradeoff: $d$ is short and the client has to contact the PV frequently ($T/d$ times); or $d$ is long and invalid use of the PC could last arbitrarily longer than $T$.

However, we can do much better with an adaptive expiration time. The following protocol guarantees that the duration of invalid use is $\leq T + O(1)$ with only $\log_2 T - O(1)$ requests sent to the PV. The protocol extends our basic client-to-PV protocol (again without any changes to the PC format or client-to-server protocol). The first time the client contacts the PV, the PV issues a certificate for a short duration $d_0$, e.g., 30 minutes. Just before this PC expires, the client requests another, with a refresh option where it includes its old PC in the message to the PV. The PV verifies that the old PC is valid and current, and if so, issues a PC with duration twice that of the old PC. This is then repeated, and guarantees that the client will be certified to use the address only during $[t, t + \max(2T, d_0)]$ with $\lceil \log_2(T/d_0) \rceil$ requests sent to the PV. In practice, there would likely be a maximum duration as well (e.g., 1 month). Note that the PV remains stateless.

### 3.1.5 Additional Security Properties

**Replay attacks:** Assuming the private keys and the secure hash function are not compromised, an eavesdropper that has heard every message has few options for replay
attacks. Specifically, even knowing a PC for a client, the attacker cannot create a novel valid RC. It can only replay existing RCs for a limited amount of time (e.g. 5 minutes, using the timeout above), which can be filtered by the server (§3.1.2.D).

**DoS attacks on servers:** ASAP introduces cryptographic overhead on servers, which could be exploited to perform DoS attacks. The worst case would be that a large number of hosts present valid PCs but invalid RCs to the server. In our implementation (§3.4), the server would be forced to perform one RSA 1024 verification, and one RSA 512 verification before declaring the request to be invalid.

Although ASAP increases the amount of work the attacker can force the server to do with a single packet, this attack has an easy defense: if a server detects that it is under attack and cannot handle the rate of requests, it can simply fall back to standard TCP handshaking. The attack thus only causes a single-RTT increase in latency rather than a service outage; thus, the attack may have limited value to attackers.

We also note ASAP’s extra computation overhead is partially compensated for by slightly reduced resources: (1) the server sends and receives one fewer packet than in the 3WH, and (2) as in TCP with SYN cookies, it avoids storing state for half-open connections. Finally, ASAP’s computational overhead could be decreased in future implementations with a faster cryptographic algorithm, such as elliptic curve cryptography (ECC) [61].

**Key compromise:** If the client’s key $K_{c_{priv}}$ is compromised (e.g., if the client is infected with a bot) this will expose only the client to DoS attacks. An attacker can then impersonate the client and mount reflection attacks, but only directed to the compromised client.

The more serious problem is if the PV’s keys are compromised. If the PV is run on a per-domain basis, it can simply discard its old keys and create new ones. If the PV is a trusted third party, servers that trusted it will have to be made aware of the compromise and remove the PV from their list of trusted PVs. Similar problems are also encountered in web Certificate Authorities [92] and similar solutions apply here.

**Privacy:** ASAP clients might be easily tracked across requests and across locations, since each request includes the client’s public key. However, the client can simply change its (arbitrary) public key when it changes its IP address and obtains a new PC, thus providing privacy that is essentially equivalent to today.

### 3.2 Fast Name Resolution

To reduce delay, ASAP piggybacks transport connection establishment atop the DNS lookup process. The intuition is that once the client’s request reaches a DNS server that knows the web server’s IP address, forwarding the message directly will be faster than going via the “triangle route” from the DNS to the client to the server, which occurs today.

In general, this shortcutting will save up to 1 RTTs, depending on the location of
the DNS server that knows the server’s IP address.\footnote{4The Internet occasionally violates the triangle inequality \cite{95}. This could either lessen or heighten ASAP’s benefit. Our evaluation will show shortcutting offers significant improvement in practice.}

In realizing this idea, our key goal is deployability. The procedure described here requires changes only at resources under control of the client and the server interested in using the protocol: the client, the server, and the authoritative DNS server (ADNS).

3.3 Basic Protocol

In ASAP (Figure 3.2b), if the client does not have the server’s IP address, it first constructs a DNS query. In the query, it inserts connection establishment information \( CI \). Specifically, \( CI \) is a sequence of bytes encoding \( meta, PC, RC, data \) as described in §3.1.2.C. ASAP does not modify the format of or add fields to the DNS query. Instead, we encode the connection establishment information into the hostname field of the DNS request, concatenated with the hostname being looked up. For example, if the client is looking up \( www.xyz.com \), it would generate a DNS request for \( a.CI. www.xyz.com \), where \( a \) is an arbitrary character which will not appear in the normal name (e.g., ASCII code 13), used by ASAP to determine if the request is from an ASAP-enabled client or a legacy client.

Since \( CI \) is unique, the local DNS (LDNS) will not have the name cached, and will route the client’s query towards the ADNS for \( xyz.com \). The ADNS (which

\[\text{Figure 3.2: Timeline for TCP and three variants of ASAP.}\]
supports ASAP) strips off the CI field and sends CI, which carries the address of the client, spoofed by the DNS server, to the server. The server then responds directly back to the client with the requested object using the ASAP transport protocol (§3.1). Since the LDNS is still waiting for a response, the ADNS also returns an A record mapping a.CI.www.xyz.com to the web server’s IP address. This response is given a small TTL since caching it will be useless for future requests (as CI is unique for each request).

While Extension Mechanisms for DNS (EDNS0) [108] enable the use of long names, to be compatible with older DNS servers, the total size of the hostname field in the query should remain below 253 bytes. In our implementation meta, PC, RC occupies a total of 212 bytes, leaving 41 bytes for the actual hostname and application data. This would be enough space for retrieving HTTP objects that do not require long parameters from the client. Specifically, the server can choose short hostnames and compressed pathnames for web objects (i.e., a URL like www.xyz.com/bX4r could be mapped by the web server to a longer pathname). However, some HTTP requests may require long client-specific parameters. At a minimum, the server will receive the first line of the HTTP request, which includes the URL [?]; conceivably, servers can decide individually whether they have enough information to act on the request (e.g. setting parameters to default values), or must wait for more data. Moreover, cryptographic algorithms using smaller key size for an equivalent amount of security, such as ECC [61], could be leveraged to provide more room for the application data.

Making this scheme practical requires solving two more problems below.

### 3.3.1 Handling DNS Caching via Multiple Queries

Unfortunately, embedding CI into the requested hostname presents problems for DNS caching, which DNS uses to improve latency and scalability. In the basic protocol described above, ASAP prevents caching because CI, and thus the hostname, varies across every connection. If the client’s LDNS (where we are particularly interested in caching) supports ASAP, then it can simply strip off CI from the name. However, in practice, ASAP may not be supported at all LDNS servers.

To address this, ASAP clients also generate a second DNS request (Figure 3.2(c)), for the original hostname (www.xyz.com), to cause that hostname to be cached at the LDNS. The cached entry can then be used by ASAP clients (to avoid intermediate DNS lookups), as well as non-ASAP clients (to directly lookup the remote server). As an optimization, upon receiving a response to the second DNS request, if the ASAP client has not yet received a response from the server, the ASAP client immediately sends its CI request directly to the server’s IP address. This can speed ASAP connections for cases where the remote server’s IP address is locally cached. Note that this optimization can cause multiple copies of the request to reach the server; but this case is functionally equivalent to the client retransmitting after a timeout, which the ASAP server (like a TCP server) must handle anyway, by ignoring the duplicate.
3.3.2 Reducing Latency with Server-side DNS

One remaining shortcoming of ASAP is that all queries must traverse a single ADNS. This increases load on the ADNS, and increases latency for sites that replicate content across the wide area to place it near users.

To address this, ASAP embeds some DNS functionality into the web servers. In particular, each ASAP-enabled server can run a Server DNS (SDNS), which performs similar functionality to the ADNS but is co-located with the server. When the ADNS is first queried, it responds to the LDNS with an NS record mapping www.xyz.com to the SDNS, and an A record mapping the SDNS to the server’s IP address. Thereafter, the LDNS will map requests of the form a.CI.www.xyz.com directly to the SDNS, and thus to the server, avoiding the ADNS.

The SDNS can be implemented as an extension to the web server that decapsulates the ASAP query, to avoid running an additional process. Also note that the service provider can perform standard load balancing and redirection by choosing which SDNS IP address to return (e.g., one physically near the requesting client).

3.4 Implementation

Fast transport connection establishment: We implemented two versions of ASAP’s transport layer. First, we built an application-layer implementation using UDT (version 4.8) [21], a reliable and congestion aware UDP-based transport protocol. We modified UDT to implement our fast transport connection establishment protocol (§3.1) by (a) adding a new connect interface on the client side that, in addition to taking the socket descriptor as input, also takes the domain name, certificates, and application data; (b) adding a new pre-fetch interface on the server side, which lets the server begin immediately transmitting data upon establishment of the connection. For ASAP’s cryptographic operations, we used SHA-256 for hash and 1024-bit RSA, as implemented in OpenSSL [16] version 1.0.0c.

To compare more precisely to TCP, we built a second implementation of ASAP within Linux kernel 2.6.38.4. The client sends a special SYN message carrying connection information and data; the server’s kernel validates provenance and passes the data to the application immediately. Note that this special SYN is compliant with TCP. By adding an ASAP option in the kernel and setting it to true, the special SYN message is allowed to have a data section, instead of only a header as in typical SYN messages. Since we could not directly make use of OpenSSL in kernel space, and since no asymmetric cryptographic algorithm is included in the standard release of the Linux kernel, we ported and modified an RSA patch [17] for Linux kernel 2.6.21, and incorporated the cryptographic part of the design as a kernel module.

The fast transport connection implementation may be run alone, or to further reduce delay, may be run jointly with our fast name resolution implementation.

Fast name resolution: We implemented ASAP’s fast name resolution as a set of extensions to the Unbound DNS server version 1.4.8. This code forms the foundation.
for the ADNS and the SDNS operations. We run unmodified Unbound as the LDNS and intermediate DNS servers in our experiments.

3.5 Evaluation

We evaluated our implementation of ASAP in a PlanetLab deployment, a local deployment with emulated latency, and microbenchmarks (§3.5.1). Overall, we find that ASAP can reduce transmission time by up to two round trips, significantly reducing latency of short web traffic (§3.6). However, ASAP also has computational overhead for cryptographic processing; we show this is manageable (§3.7). Finally, we show that using just two PVs effectively limits eavesdropping attacks (§3.7.1).

3.5.1 Methodology

We chose 24 representative PlanetLab nodes to act as clients and servers: 18 domestic nodes (UIUC, UCLA, UPenn and so on) and 6 international nodes (Brazil, New Zealand, Japan, Singapore, Taiwan, Zurich). The RTTs among these 24 nodes range from 3 ms to 440 ms. Each client was assigned a LDNS, in the same site as the client. Similarly, the server’s ADNS was placed at another PlanetLab node located in the same site as the server. We also tried placing the ADNS server on more distant nodes to see how much influence this variation causes to the performance of our design. Unless otherwise mentioned, the client downloads 11.18KB of data, the median size of data downloaded from per single host per connection according to [12], during the connection.

We implemented the transport component of ASAP as extensions to UDT. We compare against an unmodified implementation of UDT, and unmodified TCP. We implemented the name resolution component of ASAP as extensions to the Unbound DNS server, and use an unmodified copy of Unbound as a baseline. We targeted a design with low complexity, resulting in an implementation with relatively few lines of code (less than 3000 for name resolution, transport, client/server, and provenance verification functions).

3.6 Latency Reduction

In this section, we study the latency of ASAP. We define the latency savings ratio (LSR) as the time it takes ASAP to download an object divided by the time it takes today’s Internet to download the object (using UDT or TCP).

Overall latency: We first evaluate the complete ASAP protocol. We consider two separate cases: (a) the server’s IP is already cached on the LDNS, and (b) the server’s IP is not cached, requiring a lookup to traverse to the server’s ADNS. We achieve the first case by sending an initial request to “warm up” the cache before collecting results. For the first case (Figure 3.3a), the latency savings ratio is similar to the transport-
Figure 3.3: Overall download time, when DNS caching is (a) enabled, bypassing the ADNS; (b) disabled, with the ADNS collocated with the server; and (c) disabled, with the ADNS in a random location.
only experiments. For the second case (Figure 3.3b), ASAP can save up to two RTTs, resulting in more significant latency reduction.

Finally, since the ADNS may not always be near the server, we perform an experiment where we place the ADNS server at a random site (Figure 3.3c). We find that ASAP can still reduce latency in this case: even if the ADNS and server are not collocated, it is generally faster to go directly from ADNS to server, rather than from ADNS to client to server.

**Transport latency:** To understand the benefits of ASAP’s connection setup procedure, we microbenchmark only its transport operations (including connection setup, requesting the web page, and data transmission delays). Figure 3.4 shows transport latency as compared to TCP in the kernel.

Here, we use two virtual machines on one physical machine acting as the client and the server respectively. We use [15] to add latency between the two virtual machines,
making the RTT between them roughly 100ms. The client downloads 11.18KB data, as in the previous evaluation. Figure 3.4a shows the CDFs of ASAP’s and TCP’s transmission time over 1000 trials. The results confirm a significant reduction in latency. ASAP nearly always achieves one RTT reduction in latency (100ms) as compared to standard TCP. We then vary the file size from 1KB to 500KB, and run each case over 200 trials. Figure 3.4b shows the median values of total delay. ASAP achieves lower latency in all cases.

### 3.7 Computational Overhead

ASAP adds some additional computational overhead to several parts of today’s Internet, including clients, web servers, DNS servers; and the new infrastructure we deploy, the Provenance Verifier. To characterize overhead of each of these components, we instrumented our implementation with code to measure the (a) pass-through time, i.e., the time from when a packet was received to when it was processed and forwarded, and (b) microbenchmarks, to characterize what fraction of overheads were due to cryptographic processing. We collected our experiments on a single core of a 2.83GHz Intel Core 2 Quad Q9550 processor with 4GB RAM.

**Provenance Verifier:** Each request to the PV requires a private encryption to generate a PC. We would like the overhead at the PV to be low, to reduce the number of PVs, by enabling each PV to service a larger number of clients. Figure 3.5 shows a CDF of the PV’s per-request processing time with a 1024-bit RSA key, over 1000 trials. We find that more than 95% of certification requests can be processed in less than 0.8ms. Assuming each client needs to renew a PV once per day, a single PV server with a single core could handle several tens of millions of clients.

**Client:** There are two key sources of overhead at clients: the time spent communicating with the PV, and the time spent communicating with the server. The former happens rarely — it is done when the client obtains a new IP address (e.g. via DHCP), and therefore does not affect the latency associated with connection setup (except for connections initiated immediately after obtaining a new address). To evaluate the amount of overhead ASAP introduces when communicating with the server, we perform microbenchmarks (Figure 3.5). ASAP’s overhead at the client was dominated by cryptographic operations, namely, the encryption operations involved in constructing the request certificate (RC). However, this overhead was typically on the order of 1-2ms, substantially less than typical round-trip times.

**DNS servers:** ASAP increases DNS overhead in two ways. First, the client sends two queries to DNS, which could double its workload in the worst case (if entries are cached, this overhead could be substantially reduced). Second, the ADNS does additional processing on packet contents: it removes the ASAP component of the request before processing and caching the appropriate information, then replaces the ASAP component before forwarding the response back to the LDNS. To evaluate this overhead, we performed an experiment where the client sends 1000 requests to the server.
We found that overhead increased by on the order of 100 microseconds at the LDNS and the ADNS. However, this overhead is small compared to the total packet processing delay in Unbound (0.7 ms on average).

**Web server:** An ASAP-enabled web server performs an RSA verification of the message before processing it. We measure this overhead in Figure 3.5. Here, we made a client running in the UCLA PlanetLab site send 1000 requests to the ASAP-enabled web server. The median time required to perform the verification operations on a single request is 0.326 ms, the mean is 0.359 ms, and the 95th percentile is 0.462 ms. This is significantly less than typical RTTs, so although it does add some computational overhead, ASAP would provide a large benefit in end-to-end delay.

### 3.7.1 PV Eavesdropping Defense

Recall (§3.1.3) that TCP and ASAP have differing security with respect to eavesdropping. Call a client-server pair \((c, s)\) **attackable** if an attacker can induce \(s\) to accept an incoming transport connection and send data to \(c\). In TCP, \((c, s)\) is attackable if the attacker can observe messages sent from \(s\) to \(c\) (either because it is truly located at \(c\), or because it is eavesdropping). In ASAP, in the worst case of a globally trusted PV, \((c, s)\) is attackable for any \(s\) if the attacker can observe messages sent from the PV to \(c\).

Here we evaluate empirically the efficacy of using multiple PVs, such that \((c, s)\) is attackable only if the attacker can observe messages from *all* PVs to \(c\). We begin with a network map and a set of routes. We pick locations for one or more PVs, and then find the worst location for a single-site eavesdropper — i.e., the site that maximizes the number of attackable pairs \((c, s)\) given the PV locations. We then iterate this for many PV locations. Finally, we perform a similar worst-case attackability calculation for TCP, which is equivalent to maximizing betweenness centrality in the given network.
We first study the attackability for ASAP in the Internet based on actual routes observed in Route Views [93]. This data set includes routes from a limited set (about 26) of vantage points, to all destination IP addresses in the Internet; the PVs are chosen from those locations, as are servers for TCP. We allow the attacker to eavesdrop on any path that traverses one single (adversarially-chosen) autonomous system.

Fig. 3.6 shows attackability (i.e., fraction of attacked client-server pairs) on the $x$ axis; the $y$ axis is a CDF over possible PV locations. One PV is vulnerable to the attacker, who can simply eavesdrop on the PV’s AS. However, the attacker’s effectiveness is reduced dramatically for two PVs. Further PVs offer diminishing returns, asymptoting to 4.9% attackability. In contrast, TCP’s attackability is 24%, and even when the attacker is prevented from eavesdropping on a tier-1 AS, this only falls to 9.84%.

Table 3.1 shows results for two PVs in additional topologies: six ISP networks, as measured by Rocketfuel [102], in which we assume routes follow shortest paths; CAIDA’s AS-level map of the Internet [40], in which we assume routes follow common customer/provider/peer policies; and the Route Views data with and without the tier 1 ASes being attackable.

These results show that for an attacker who can eavesdrop on one worst-case site, two PVs are sufficient to limit the attackability. In particular, even if two PVs are placed randomly, the chance that ASAP has fewer attackable client-server pairs than TCP is at least 75% in all topologies we tested. Moreover, the PVs can be intentionally placed in two topologically-diverse, and therefore better-than-random, locations.
### Table 3.1: Attackability of TCP and ASAP with two PVs.
The columns are the topology, attackability of ASAP with optimally-placed PVs, attackability of TCP, and the chance that ASAP’s attackability is less than TCP’s if the PVs are placed randomly.

<table>
<thead>
<tr>
<th>Topology</th>
<th>ASAP (best PVs)</th>
<th>TCP</th>
<th>Chance ASAP better</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS 1221</td>
<td>15.38%</td>
<td>39.43%</td>
<td>83.37%</td>
</tr>
<tr>
<td>AS 1239</td>
<td>2.20%</td>
<td>17.80%</td>
<td>84.98%</td>
</tr>
<tr>
<td>AS 1755</td>
<td>5.75%</td>
<td>28.55%</td>
<td>77.65%</td>
</tr>
<tr>
<td>AS 3257</td>
<td>6.21%</td>
<td>28.81%</td>
<td>75.30%</td>
</tr>
<tr>
<td>AS 3967</td>
<td>3.80%</td>
<td>37.76%</td>
<td>91.50%</td>
</tr>
<tr>
<td>AS 6461</td>
<td>4.35%</td>
<td>44.65%</td>
<td>88.04%</td>
</tr>
<tr>
<td>AS-level Internet</td>
<td>1.64%</td>
<td>17.72%</td>
<td>87.56%</td>
</tr>
<tr>
<td>Route Views</td>
<td>4.89%</td>
<td>23.61%</td>
<td>91.41%</td>
</tr>
<tr>
<td>Route Views w/o T1</td>
<td>4.89%</td>
<td>9.84%</td>
<td>93.88%</td>
</tr>
</tbody>
</table>

3.8 Deployment

Like most new transport and naming protocols, our design requires changes to certain clients and servers. However, ASAP can be deployed in an incremental and end-to-end manner, with even a single client-server pair realizing benefits. We next describe the requisite changes at participating clients, servers, and the server’s ADNS.

**ADNS:** Since the ADNS is typically owned and operated by the service provider, many deployed systems (e.g., Akamai) leverage the ADNS as an easy-to-modify location to place new functionality. Our implementation of ASAP consists of some simple extensions to software running at the ADNS.

**End host clients:** We require modifications to the client’s TCP implementation. A key question is whether this is deployable in a backwards-compatible manner, so clients interacting with legacy servers will simply fall back to TCP. One option is the strategy of our implementation: include the certificates in the data portion of the SYN packet. This is likely to work well in practice since most current TCP implementations discard this data; but technically it could be unsafe because legacy servers that do use SYN data would misinterpret our certificates as application data. Another option is to put the certificates in a TCP option, as in TCP Fast Open [89]. However, for this we would need more space for options, as proposed in [49].

To avoid changing end host network stacks, applications could use ASAP on top of UDP, as in our modification of UDT [21]. To avoid modifying end hosts entirely, ASAP could be deployed at web proxies and caches.

**Server:** Like the client host, the server should be modified with extensions to support the ASAP protocol. If desired, ASAP could also be deployed as a reverse proxy, to avoid the need to modify servers; however, we note that service providers already commonly customize their operating system and web server implementations.

**PVs:** Note that ASAP does not require deployment of a shared PV infrastructure. Such a deployment scenario would be useful and could arise, but in an initial deployment each organization could host its own PVs, thus requiring little coordination. This
deployment is likely to bring large benefits for content distribution networks and other large content providers.

**DNSSEC instead of PVs:** An alternative to verifying source addresses with PVs is to use DNSSEC’s [31, 33, 32] designated signer (DS), public key (DNSKEY), and signature (RRSIG) records to certify ownership of IP addresses via reverse DNS lookup, as in [74]. The server would verify the chain of certificates from the root of the DNS hierarchy to the client. Servers could cache some of these records near the top of the hierarchy, while the client would provide others in its request. This leads to a tradeoff: placing more records in the query increases its size, which is already constrained (particularly when piggybacked within DNS queries); but if we require the server to cache more levels of the hierarchy of certificates, it will lead to cache misses and the server will need to query the DNSKEY records from corresponding DNS servers, increasing delay. In addition, compared with our PV design, using DNSSEC may be more difficult to deploy, as it requires each client’s local ISP to issue certificates of IP ownership to the client.

**Leveraging accountable Internet architectures:** While ASAP does not require extensive changes to the Internet, it can benefit from deployment of previously proposed clean-slate designs. For example, systems that cryptographically certify location or ownership of IP addresses [27, 74] may obviate the need to run PVs.
Chapter 4

Halfback: Running Short Flows Quickly and Safely

4.1 Background

4.1.1 Design Goals and Rationale

Rate control for short flows should be deployable, low latency, and safe. We discuss each goal and its corresponding implication on design rationale of latency optimization for short flows.

(1) **Deployability**: Mechanisms that require changes in routers and the TCP protocol have proven hard to deploy. Software changes within senders and receivers require less coordination among parties. We focus in particular on *sender-side changes only*, since significant senders (major service providers like Google or Amazon) have centralized control over their deployments, and have an incentive to change because even hundreds of milliseconds affect user behavior and revenue. For example, [7] observed major content providers enlarging their initial congestion window, and we observed an 8-segment ICW in use at google.com.

(2) **Low Latency**: The first requirement to achieve low latency is *aggressive startup*. Even with some historical hints, a sender will not be able to perfectly predict the appropriate rate for a new flow since it lacks real-time visibility into the end-to-end path. Nor will it have time to gradually learn the rate, as in TCP. Therefore, low-latency flows have to be *aggressive* in the sense of starting with a sending rate that will occasionally turn out to be higher than the steady-state.

Second, we need **fast recovery from packet loss**. Even one RTT spent detecting packet loss is undesirable. TCP uses fast retransmission and SACK to respond to loss. However, senders still need to wait at least one RTT after a loss, or even more if packets are lost at the end of the flow. Some algorithms [53] use erasure coding but this requires a new protocol and increases the difficulty of deployment. Thus, to respond to loss more quickly than one RTT, we need to implement some form of *proactive retransmission* which retransmits packets even before receiving a signal of loss.

Finally, a scheme which achieves low latency should be **minimally affected by bufferbloat**. Bufferbloat caused by large buffers in routers increases RTT by increasing queuing delay. One can mitigate the effect of bufferbloat by finishing transmission in fewer RTTs.

(3) **Safety**: An aggressive mechanism can easily cause a series of problems. First, as
TCP is a conservative protocol, the aggressive mechanism can damage the competing (short and long) TCP flows. Second, as the aggressive startup phase has a high initial sending rate, this can even cause problems for other aggressive flows. Finally, proactive retransmission needs extra bandwidth which can increase network utilization and cause the onset of performance collapse at lower utilization than TCP. In summary, a safe aggressive mechanism should avoid congestion collapse in the range of realistic network utilization, be TCP friendly and incur limited bandwidth overhead.

There is hope, however, that with a well-designed mechanism, more aggressive start-up can be safe and avoid Internet-wide catastrophic effects. TCP is very conservative for short flows when it faces this tradeoff. At the same time, the average network utilization is typically around 20% to 30% in the Internet (based on 2003 measurements of backbone links [55]). Fig. 4.1 shows a CDF of the fraction of traffic carried by a range of flow sizes in several networks. In the measurements labeled “Internet” from a Tier-1 ISP, only 34.7% of bytes were carried by flows smaller than 141KB [88] even though more than 95% [25] of web transfers are smaller than this size. Furthermore, as noted in a recent forecast report [4], by 2019, video streaming traffic will comprise more than 80% of global Internet traffic. Therefore, start-up phase optimization mechanisms with carefully tuned aggressiveness for the small portion of very short flows probably will not severely overload the Internet as a whole. This kind of optimization is also likely to be applicable to data center networks. In measurements at a private [35] and a public [59] data center, less than 1% of transmitted bytes were in flows smaller than 141KB. Therefore even Proactive TCP [53], which doubles the workload created by short flows, only increases network utilization by 0.2% to 10.4% (i.e., 20% · 1% to 30% · 34.7%) in these environments when applied to flows smaller than 141KB.

### 4.1.2 Overview of Existing Solutions

Several works have developed approaches deployable at end-hosts which aim to reduce latency for short flows. PCP [28] uses packet-trains to measure available bandwidth and sets its sending rate at measured rate. However, the probes take time and can
yield inaccurate (often too conservative) results on very small samples, resulting in unacceptably long FCT. Our experiments with PCP showed that it can have higher flow completion time than TCP.

Reactive TCP [53] uses a probe timeout (PTO) to retransmit the last packet as a probe, thus avoiding the longer retransmission timeout (RTO). However, this does not solve the problem that the starting phase is too conservative in TCP and can only mitigate the effect of packet loss in the case of tail loss. TCP-10 simply increases initial congestion window to 10. It does achieve better latency performance, but as we will see in § 4.3, it is still too conservative for just transmitting short flows. Proactive TCP transmits two copies of every packet in a short flow and unsurprisingly incurs severe safety problems as its latency performance collapses even with relatively low network utilization.

Our experiments showed that the existing proposal which achieves lowest flow completion time, at least in low-utilization scenarios, is JumpStart [75]. JumpStart accelerates short flows by transmitting the entire flow in one RTT. This is done with packet pacing, so that packet transmissions are evenly spaced across this single RTT. However, after the first batch of data is paced out, JumpStart falls back to normal TCP with bursty and reactive-only retransmission. JumpStart is effective if all packets get successfully pushed through the network. However, very commonly, some packets in the first batch are dropped. When that happens, JumpStart has two problems. First, it relies on TCP’s reactive packet loss detection and has to wait at least one RTT to recover. Second, and more significantly, JumpStart uses TCP’s retransmission mechanism and will aggressively burst out all lost packets and will often incur even more loss. This bursty retransmission mechanism gives it significantly worse flow-level safety compared to TCP-family solutions. Our experiments show JumpStart [75] has performance collapse when short flows drive the network utilization to around 50%. This unsatisfying flow-level safety property actually translates to even worse application-level latency performance because web page requests usually involve multiple concurrent short flows, magnifying the packet loss problem by creating a brief transient high utilization scenario. Indeed, when using real webpage request patterns, JumpStart’s application-level performance begins to collapse (i.e., it becomes worse than TCP’s) at network utilization of 30%.

As mentioned in 4.1.1, inevitably, aggressive startup phases will sometimes choose too high of a rate. The above discussion of JumpStart illustrates that it’s easy to send data quickly but the trickiest part of the problem is how to best handle the inevitable over-shoots. Furthermore, the application-level results illustrate that an individual sender should want to do this not only to help other sender’s flows, but also to reduce interference among its own flows.

## 4.2 Halfback Design

Measurement and analysis suggest that we should design a protocol with an aggressive initial packet sending phase and intelligent proactive packet retransmission phase that
reduces latency and also limits aggressiveness to improve safety. We propose Halfback to realize this design rationale with two mechanisms: Pacing and Reverse-Ordered Proactive Retransmission. The Pacing phase follows past work in that it delivers data quickly, but may incur higher loss rate; the ROPR phase recovers from that potential loss effectively with limited aggressiveness in retransmission.

4.2.1 Pacing Phase

After the three-way handshake, the sender has acquired the flow control window size advertised by the receiver, and a sample RTT. Halfback’s first data transmission phase then begins. The fastest way to transmit the data is simply in one immediate burst at line rate. However, this arbitrarily large sending rate may harm the existing flows and increase packet loss. Instead, we borrow a technique from JumpStart [75]: we pace out all the data in one RTT. Compared to sending in an immediate burst, this method adds at most one RTT (for a total of two) but bounds the transmission rate so there is significantly lower chance of a burst of packet losses, which is good for all flows on the network.

In addition, we also give an upper bound of the data transmitted which can be used to bound the transmission rate. This upper bound equals the minimum of the flow control window size, flow size, and a Pacing Threshold. If the flow size exceeds this bound, Halfback falls back to TCP (§4.2.3). The application designers could simply set Halfback’s Pacing Threshold to a constant value that would be sufficient to transmit most small web objects. In our experiments, we use a threshold of 141KB which can cover more than 95% [25] of web transfers. Another option, not evaluated here, is to set the threshold to the largest throughput observed on recent connections, times the RTT derived from the three-way handshake. This setting efficiently avoids a too-aggressive startup phase.

4.2.2 Reverse-ordered Proactive Retransmission (ROPR) Phase

After completing the Pacing Phase, Halfback proactively begins protecting itself from packets that may have been lost. The basic idea is that in addition to normal packet retransmission, Halfback proactively retransmits the flow’s packets, but does so in reverse order and at the same rate that it receives ACKs from the previous phase. This proactive retransmission helps Halfback quickly recover from loss while limiting impact on other flows. To better understand this scheme, we explain the design rationale for each aspect of this ROPR phase: starting time, retransmission rate, and the order in which to retransmit packets.

We choose to start this phase when the sender receives the first ACK after the Pacing phase. Due to inaccurate estimation of RTT, ACKs can be received before the pacing phase finishes. In that case, ACKs will not trigger proactive retransmission until all new packets are paced out. This avoids competition between the paced and retransmitted packets. It also allows the sender to do some useful work — whereas in
standard TCP, having transmitted all the data, the sender would simply be idle waiting for ACKs.

For the retransmission rate, we use the rate at which the sender receives ACKs. In contrast to TCP and JumpStart, which can send a burst of (reactive) retransmitted packets, this (proactive) ACK-based retransmission better approximates the current available bandwidth of the bottleneck link. That is, roughly speaking, for each one of the paced packets that leaves the bottleneck queue, we send one proactively retransmitted packet. As a result, we avoid affecting other flows, and significantly reduce the probability that the retransmitted packets are lost again which helps the sender avoid timeout and reduces bandwidth overhead.

The design decisions above specify when and how we can proactively retransmit a packet; but which packet do we send each time we have the chance? The goal here is to quickly recover from any packet loss caused by the aggressive startup phase. As we are retransmitting proactively, we don’t know which packets are lost; so Halfback tries to proactively retransmit packets in decreasing order of the probability they were lost. When the Pacing phase sends a large amount of data in a short period, the packets at the end of the flow have a higher probability of overflowing a bottleneck queue and being lost than the packets at the beginning. Thus, in ROPR, the sender proactively retransmits packets in reverse order. When combined with the fact that ROPR matches the rate of ACKs from the Pacing Phase, this means that in the typical case, the ACKs (moving forward) will meet the retransmissions (moving backward) in the middle of the flow. Thus, ROPR typically retransmits only 50% of the short flow—hence the name Halfback—which means that it will only increase network utilization by 0.1% to 5.2% in the typical network environments mentioned in §4.1.1.

RC3 [79] uses a seemingly similar mechanism that transmits packets in reverse order in its Recursive Low Priority (RLP) control loop. However, we want to highlight that RC3’s reverse-ordered transmission is totally different from Halfback in terms of when, how and why. RLP transmits reversed-ordered packets at line rate, and it does so concurrently with TCP’s normal forward-ordered packet transmission. More importantly, RC3 requires in-network changes to transmit the reverse ordered packets to a lower priority queue in the network. RC3 uses reverse ordering to avoid transmitting the same packet for both primary control loop and RLP control loop, whereas Halfback use it for proactive recovery from packet loss.

### 4.2.3 Falling Back to TCP

Aggressive transmission is not useful for long flows, where overhead would have greater impact and flow completion time is less critical. Without information about exact flow sizes, Halfback needs a mechanism to fall back to normal TCP for long flows. A practical solution is to transmit aggressively for the first $k$ bytes, effectively the Pacing Threshold discussed in §4.2.1, and then fall back to TCP. Halfback will successfully deliver the first $k$ bytes of the flow using its Pacing and ROPR phases, and then will fall back to TCP with a congestion window of $s \cdot RTT$, where $s$ is estimated
from arriving ACKs during the ROPR phase. Other bandwidth estimation mechanisms can also be used [65, 104].

### 4.2.4 Example

In this section, we walk through an example 10-segment flow transmitted by Halfback. Fig. 4.2 shows the whole process. In the first RTT, Halfback’s Pacing phase, the sender paces out all the ten segments in one RTT. When it receives the first ACK, the sender enters Halfback’s ROPR phase to proactively recover from potential packet loss. In this phase, for each ACK received, the sender will proactively retransmit one unACKed packet in reverse order: it receives ACK 1, and retransmits packet 10; it receives ACK 2 and retransmits packet 9; and so on, until it receives ACK 5 and retransmits packet 6. Next, the sender receives ACK 6. At this point, all the unACKed packets have already been proactively retransmitted, and the sender leaves ROPR phase.

As shown in Fig 4.2, the first transmission of packet 9 was dropped because the aggressive startup phase overflowed the router buffer. But Halfback proactively retransmitted the packet during the ROPR phase and thus recovered from the loss before being notified of it. In contrast, a normal TCP sender needs to wait until timeout since there are not enough duplicate ACKs (three are needed) to generate a lost-packet signal. Even if we retransmit the last packet multiple times to generate enough duplicate ACKs to avoid timeout, as in Reactive TCP, the receiver will receive packet 9 \(0.9 \cdot RTT\) later than Halfback and thus add \(0.9 \cdot RTT\) to the FCT.

ROPR masks the latency penalty from packet loss but it also carries the cost of additional packet retransmission with additional bandwidth consumption. However, as we demonstrate in Fig. 1.2, TCP is too conservative for short flows and in §4.3.3 we show that this additional bandwidth will not cause problems for the whole network and co-existing flows.

### 4.3 Experiments

In this section, we conduct a performance evaluation of eight schemes to optimize short flow latency: TCP, TCP-10 (set ICW to 10) [48, 25], TCP-Cache (caching older values of the cwnd and ssthresh), JumpStart [75], PCP [28], Reactive TCP [53], Proactive
TCP [53], and finally Halfback. Our goal is to determine where these schemes lie in the tradeoff space between latency and safety.

The experiments consist both flow-level benchmarks and application-level benchmarks as listed below. For flow-level benchmarks, we first compare different protocols’ latency performance with flow completion time (FCT) under different network scenarios. And we evaluate the latency-safety trade-off with two metrics: TCP friendliness and feasible network utilization, which we define as the maximum achievable network utilization before the throughput collapses. To understand how the flow-level benchmarks translate to application performance, we also evaluated application-level benchmarks that measure the latency-safety trade-off with traffic patterns based on real web sites.

### 4.3.1 Experiment Settings

**Protocol Parameters:** We use code from the PCP project directly [22]. For each of the other mechanisms, we implemented the scheme within UDP-based Data Transfer (UDT) [21] with Selective ACK. The segment size is 1500 bytes including the header. The flow control window size advertised by the receiver is 141KB, the same as that of Windows XP [48]. In our evaluation, we still use 2 segments as the default initial window size for TCP protocols (except TCP-10). Note that although [48, 25] suggested to set the ICW to 10 segments, it is not universally deployed. Halfback sets the Pacing Threshold to the flow control window size.

**Evaluation Environment:** We test Halfback in both the wild Internet and controlled emulation environments. In PlanetLab (§4.3.2.A), we randomly chose approximately 2.6K pairs among 100 hosts to act as senders and receivers. The locations of the nodes include Asia, North America, Australia, Europe, and South America. The RTTs range from 0.2ms to 400 ms. The flow size is 100KB. We also test Halfback with four home networks (§4.3.2.B) from different providers (AT&T DSL with about 6Mbps downlink connected to a home wireless router, Comcast with a wired 25Mbps downlink, ConnectivityU with shared WiFi in a whole building and ConnectivityU with a wired

1 We used the method of [7] to measure the ICW of the 10 most popular websites and found only four of them increased their ICW, including one that set it as four segments.
Figure 4.4: The number of normal TCP retransmissions of short flows in Planetlab experiments

connection) in Champaign, Illinois. The clients are deployed behind home networks and servers are on 170 PlanetLab nodes. The clients request short flows of 100KB size from the servers. All other experiments are performed in Emulab, with the topology in Fig. 4.3 emulating a single-bottleneck access network. We evaluated the performance of Halfback with a wide range of realistic workloads and varying network parameters. Unless otherwise stated, the router’s buffer size is the BDP between sender and receivers, 115 KB. For flow-level benchmarks in Emulab, unless otherwise stated, short flows have size 100 KB and have exponential interarrival-time distribution.

### 4.3.2 Flow-level Benchmarks: Latency

**A. Global Internet Evaluation on Planetlab**

Fig. 4.5 is the CDF of the FCT of short flows in our PlanetLab evaluation across
2.6K node pairs. The FCT includes both the data transmission time and connection setup time. TCP has mean FCT of 1883 ms, with JumpStart significantly better at 905 ms and Halfback at 791 ms (13% reduction). Among the 2.6K experiments, 75% of them have no packet loss during transmission and therefore, Halfback and JumpStart will have same FCT for those pairs. If we normalize the FCT by RTT (Fig. 4.6), 60% (not 75% due to RTT estimation inaccuracy) of the flows can be transmitted in 2 RTTs which is one third of TCP’s time. Halfback’s lower mean FCT than JumpStart is because ROPR can handle packet loss better with proactive recovery and limited aggressiveness to avoid timeout. Halfback’s 99th percentile FCT is 27.8% of TCP’s, 29.9% of TCP-10’s and 87.8% of JumpStart’s. To further understand Halfback’s performance gain in the face of packet loss, Fig. 4.7 shows the CDF of FCT for the 25% of cases where packet loss does happen. Halfback achieves a significant 193ms (21%) reduction in median FCT compared to JumpStart.

We also measured the distribution of number of packet retransmissions in a flow
Figure 4.6: The number of RTTs used in the transmission of short flows in Planetlab experiments

(Fig. 4.4). In general, the network utilization is low in PlanetLab and therefore Jump-Start and Halfback both achieve low packet loss in 90% of trials. At the same time, due to their aggressive startup phase, they have relatively large 99th percentile packet loss. This happens when the bandwidth of the bottleneck link is noticeably smaller than the pacing rate in the aggressive startup phase and/or the bottleneck router buffer is small. Note that Halfback runs normal TCP retransmission in parallel with ROPR; so ROPR masks the latency penalty from loss but does not reduce the number of normal TCP retransmissions.

B. Home Access Networks

PlanetLab nodes are mostly in research institutes and therefore, generally have more access bandwidth than normal end-user access networks. To get some insight into how Halfback performs under actual home access networks, we re-run the evaluation of § 4.3.2.A with four clients behind four different home networks and 170 servers.
Fig. 4.7: CDF of FCT under cases where packet loss happened

on PlanetLab nodes on Oct. 11th, 2015. We only compare the FCT of Halfback and TCP in this experiment. Note that while we believe these experiments are representative of home connections, the several measurement locations should not be interpreted as representative of the individual providers’ general performance.

Fig. 4.8 shows that in these real home networks, Halfback achieves significantly improved FCT compared to TCP. Specifically, Halfback’s median FCT is 50%, 68%, 50% and 18% less than TCP’s in access networks provided by Comcast wired connection, ConnectivityU wireless, ConnectivityU wired connection and AT&T wireless respectively. We believe Halfback achieves less improvement in the AT&T network because the evaluated network is of low bandwidth, but further detailed investigation is needed. This experiment also does not include the effect of CDN caching, but it demonstrates that Halfback improves short flows’ FCT in actual end-user networks. Larger scale and more comprehensive evaluation will be left to future work.

D. Effect of Bufferbloat

Overly-large router buffers can be filled by TCP, producing bufferbloat, which increases queuing delay and FCT of short flows. In this section, we evaluate the average FCT for different router buffer sizes. From the results, we demonstrate that Halfback consistently works well across small and large buffers. In this experiment, there is one background TCP flow and multiple short flows sharing the bottleneck link. The average interval between the short flows is 10 s. The whole experiment runs for 600 s.

Fig. 4.9(a) shows the resulting average FCT. Compared with the other schemes, Halfback, JumpStart, TCP-cache and TCP-10 are less affected by bufferbloat as they finish transmissions in fewer RTTs. Their average FCTs only increase ∼500 ms, while TCP’s increases 1048 ms.

TCP-10, TCP-Cache, and JumpStart all begin sending quickly. As a result, when router buffers are small (< 50KB) they experience significantly higher FCT than for
their optimal buffer size. Halfback also begins sending quickly but ROPR helps it recover from the resulting loss, achieving up to 45% lower FCT than JumpStart and 60% lower FCT than TCP-10 when the buffer is small.

PCP does not perform well when it co-exists with TCP. A PCP sender uses probing to estimate the queue length on the end-to-end path. It will not send data, except probing, when the queuing delay is increasing during the probing. But the competing TCP senders keep building up the queue, so that PCP is actually more conservative than the competing flows.

Fig. 4.9(b) shows the measured number of normal retransmissions, which equals the total number of packet losses noticed by the receiver. Halfback only has 6 retransmitted packets on average which is 10.6% of JumpStart when the router buffers are small. We focus on normal retransmissions here to demonstrate that Halfback can effectively use proactive retransmission to protect it from using TCP’s normal retransmission mechanism, which causes prolonged FCT. Halfback and JumpStart both have packet loss due to their aggressive startup phase, but in JumpStart, the retransmitted packets are sent at line rate which causes a large fraction of them to be lost again and each lost packet may require multiple retransmissions. Since the sender needs to wait until timeout when the retransmitted packets are lost, the loss of retransmission significantly increases the short flows’ FCT. Halfback’s ROPR sends proactively retransmitted packets at the rate of ACKs received which approximates the available bandwidth at the bottleneck link. Therefore, retransmitted packets are rarely lost again and Halfback can, compared with JumpStart, have less retransmission overhead. PCP has the smallest number of retransmission due to its conservative probing scheme.

D. Effect of Flow Size Distribution

The previous experiments all used fixed-size 100 KB flows. In this section, we evaluate FCT with flow size distribution drawn from measured distributions: a 10 Gbps
backbone link of a Tier-1 ISP [88], a 1500-node cluster in a Microsoft data center network [59], and a private enterprise data center network [35]. We truncate the distributions and set the maximum flow size to be 1 MB (as longer flows would use TCP). The time interval between two flows is varied to achieve 25% network utilization.

We investigate FCT as a function of flow size, shown in Fig. 4.10. For flows of a few tens of KB, TCP-Cache (and in a narrow region, TCP-10) achieves better performance than Halfback, but after about 75 KB Halfback and JumpStart have the best performance, achieving up to 313 ms lower latency than TCP and up to 233 ms lower than TCP-10.

TCP-Cache has an unrealistic advantage here: the experiments use a sequence of flows on an unchanging network topology (Fig. 4.3) with constant utilization and flow size distribution. Real-world use of TCP-Cache, where senders encounter a diverse

---

**Figure 4.9: The performance of short flows for different router buffer sizes**

---

2 Original data sets were not available; the distributions here were approximated from figures in the publications.
Figure 4.10: Flow completion time for different flow size under 25% network utilization for different network traffic distributions
range of receivers across the public Internet and network conditions change, would have poorer estimates of the correct rate. A large-scale characterization of TCP-Cache performance would require logs of server/client interactions across time and is outside the scope of this study (and is an interesting area for future work).

But why does TCP-Cache outperform Halfback? Halfback sends at a high rate; however it paces its data over one RTT, which can delay FCT for very small flows. Indeed, Fig. 4.10 shows that TCP-Cache outperforms Halfback in a very small range of small flow size. An easy refinement of Halfback would be to send a first batch of data as a burst (either 10 segments as in TCP-10 or a historically-sized window as in TCP-Cache) before Halfback’s Pacing Phase.

4.3.3 Flow-level Benchmarks: Latency-Safety Trade-off

Mechanisms with aggressive initial startup phase come with overhead that may be problematic and in extreme cases could cause performance collapse. We need to ensure that a mechanism chosen for short flows is safe, in the sense that potential performance degradation is limited. In this section we measure the effects when aggressive short flows compete with (1) each other, (2) long TCP flows, (3) short TCP flows, and (4) the transient disruption effect on ongoing flows. The results indicate that the aggressive startup phase used by both JumpStart and Halfback can increase aggressiveness, but Halfback’s ROPR phase significantly mitigates this problem.

A. Short Aggressive vs. Short Aggressive

We begin with the most demanding environment: aggressive short flows competing with each other under high utilization. All flows run the same protocol, so there is no issue of TCP-friendliness, but all flows are short and hence all incur overhead to achieve low latency. This is a pessimistic scenario, because as noted in [4], most Internet flows are long video streaming flows and therefore, even in a highly utilized edge network, the extra load incurred by more aggressive short flows will be much smaller than the scenario evaluated here. We evaluate Halfback in this challenging network condition because we believe rate control protocols should be reasonably robust to unusual scenarios in addition to performing well in the common case.

The flows are all 100 KB, and we vary average network utilization (transient utilization can be higher) from 5% to 90% in 5% increments. The key question is to what feasible capacity we can push utilization before performance collapse, with a spike in packet loss and FCT.

The results (Fig. 4.11) show that TCP, TCP-10, TCP-Cache, and Reactive have feasible capacity of 85% to 90% utilization. Due to its proactive retransmission that doubles bytes transmitted, Proactive TCP has performance collapse at 45% network utilization. JumpStart performs slightly better, with feasible capacity of 50%. Halfback improves this significantly to 70%, similar to PCP but with dramatically better FCT than PCP.

B. Short Aggressive vs. Long TCP

45
Figure 4.11: The performance of different mechanisms for different network utilizations while there are only short flows

In this experiment, 10% of the traffic is generated by short flows and 90% is generated by 100 MB long flows. We vary the short flows’ rate control mechanism, but the long flows always run TCP. We vary the average interval between flows to achieve different network utilizations, from 30% to 85%. For lower-variance comparisons, all the experiments for different schemes use the same schedule of flow arrivals for each network utilization.

Fig. 4.12 shows the average FCT of short flows and long flows, normalized by their FCTs under a baseline scenario where the short flows run TCP.

For short flows, compared with TCP, Halfback achieves around 56% lower FCT, JumpStart is 51% lower and TCP-10 is 29% lower. Proactive TCP experiences a small increase in FCT as its proactive retransmission increases the queuing delay and causes additional packet loss. For long flows, Proactive TCP increases their FCT up to 25% due to its whole-flow proactive retransmission and JumpStart increases it about 10% because of the aggressive startup phase and its propensity to retransmit the same packets multiple times. Halfback only slows long flows by 3% as in its ROPR phase, the retransmission rate approximates the available bandwidth and avoids affecting other flows.

C. Short Aggressive vs. Short TCP

In this experiment, half of the flows employ a non-TCP mechanism and the others use TCP. In each scenario, we pick one non-TCP protocol and one network utilization (ranging from 5% to 30% in steps of 5%). Figure 4.13 shows the results as a scatter plot, where each point is a particular protocol at a particular utilization. The x-axis is the average FCT of TCP flows in that scenario, divided by the average FCT if all flows run TCP; the y-axis is the average FCT of the non-TCP flows in that scenario, divided by the average FCT if all flows run the non-TCP protocol. In other words, we measure
Figure 4.12: Flow Completion Time normalized by the Flow Completion Time of TCP for different network utilizations with 10% of traffic created by short flows and 90% by long flows

The results show that Halfback, TCP-10, TCP-Cache and Reactive TCP are TCP-friendly as their results are located near (1, 1) where the FCTs of TCP flows and non-TCP flows are not affected due to multi-protocol deployment. Halfback has an aggressive startup phase, but as its FCT is small, it leaves more space for TCP flows after its transmission.

JumpStart and Proactive TCP are somewhat non-TCP-friendly. JumpStart, due to its aggressive startup and propensity to retransmit the same packets multiple times, increases the co-existing TCP flows’ FCT. Proactive has high overhead from retransmitting every packet. Because PCP’s probing can only succeed when the TCP flows stop sending new data, as we explained in §4.3.2.C, its FCT is increased while co-existing
D. Effect on Throughput of Ongoing Flows

For an important class of real-time applications, like video conferencing and online gaming, throughput is very important. Aggressive mechanisms for low latency may, unfortunately, affect the background flows’ throughput. In this evaluation, we run a background TCP flow and after achieving full bandwidth, start a short flow with Halfback or TCP. We count the number of successfully transmitted packets in every 60 ms and calculate each flow’s throughput. The results are shown in Fig. 4.14.

Ideally, we would like the throughput of the background flow and short flow to be like Fig. 4.14(a) in the sense that TCP recovers quickly and short flows finish transmission fast. When we employ Halfback for the short flows (Fig. 4.14(b)), as the background flow employs TCP whose AIMD congestion control needs a long time to recover from sending rate reduction after packet loss, the sender needs 180 ms to achieve half bandwidth and around 2 s, 1 s longer than that when we employ TCP for short flows (Fig. 4.14(c)), to achieve full bandwidth.

However, this effect of throughput is mitigated by several facts. First, the background flow can quickly achieve half its former bandwidth, and could recover even more quickly with a protocol like PCC [47]. Second, and most importantly, the same effect can be caused by short TCP flows. In the current Internet, to achieve lower latency, many applications separate their data into multiple parts and start several TCP connections simultaneously. We evaluate how two TCP flows each with half the flow size may affect the background flow (Fig. 4.14(d)). In this case, it needs about 2.7 s to recover full bandwidth and the short flows still have longer FCT than Halfback.
Figure 4.14: Throughput of flows for (a) Optimal situation (b) Halfback (c) One TCP (d) Two TCP flows with half flow size

4.3.4 Application-level Benchmark: Webpage Response Time

In the previous sections, we compared different approaches in terms of flow-level benchmarks including latency performance and latency-safety trade-off. However, the flow-level benchmarks do not directly translate to application-level performance. To better understand the connection of flow-level benchmarks to application performance, we evaluate a realistic scenario where a client randomly requests the front page of one of the 100 most popular web sites [20] including all objects. The server will send all the objects of this website in the same order as when the client uses the Chrome web browser. We vary the inter-arrival time between two web requests to control the network utilization. In this experiment, we measure the average web request response time (delivering all objects) at different network utilizations for different protocols.

As shown in Fig 4.15, JumpStart’s response time becomes larger than TCP and is 592ms (27%) larger than Halfback at only 30% utilization. Even for lower utilizations, the ordering between protocols changes compared to flow-level results: JumpStart is now worse than TCP-10. Overall, Halfback achieves much better latency-safety trade-off at the application level for web browsing. This unexpected result is because of the concurrent connections that web browsers usually start simultaneously which can cause transient high network utilization. JumpStart will incur high packet loss and cannot recover quickly.

Halfback is also affected by the concurrent connection effect. The response time becomes slower than TCP at 55% of network utilization, which is smaller than the flow-
Figure 4.15: Average response time for different mechanisms under different network utilization

<table>
<thead>
<tr>
<th>Startup phase</th>
<th>Lost packet recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Additional bandwidth</td>
</tr>
<tr>
<td>Slow Start</td>
<td></td>
</tr>
<tr>
<td>2-segment initial cwnd</td>
<td>0%</td>
</tr>
<tr>
<td>10-segment initial cwnd</td>
<td>50%</td>
</tr>
<tr>
<td>Pacing</td>
<td>Pacing whole flow in one RTT</td>
</tr>
</tbody>
</table>

Table 4.1: Different kinds of startup phase and lost packet recovery schemes.

level benchmark result in §4.3.3.A. However, since the average network utilization is only 20% to 30% [55], and this experiment is a pessimistic case in which all utilization is from relatively short web flows (rather than movie downloads), it is safe to deploy Halfback into the Internet. On the other hand, JumpStart’s application-level results are unsatisfying. In sum, compared to JumpStart, Halfback improves significantly in both latency and safety at the application level.

4.4 Discussion

In this section, we discuss and evaluate in detail why Halfback’s ROPR phase contributes to its better performance over other schemes.

In Table ??, we list several different schemes that can be used for initial startup phase and packet loss recovery phase. For short flows, choosing initial start-up mechanism is important but relatively simple: it has to be more aggressive than TCP’s conservative approach. As shown in Fig. 4.16, when the network utilization is small, pacing all data out at one RTT (as in JumpStart) achieves the smallest FCT which is
80.1% of TCP-10’s and 50.8% of TCP’s. However, using only this aggressive startup phase will cause performance to collapse at 50% network utilization which is much smaller than TCP-10’s (85%) and TCP’s (90%) feasible capacity. This is because the normal TCP retransmission scheme cannot quickly recover the lost packets and is still too aggressive itself by bursting out all lost packets with high possibility to incur more packet loss.

Therefore, a good design of proactive packet retransmission mechanism is needed. The key questions then are what design decisions to make in terms of additional bandwidth used, order of retransmission and retransmission rate. Halfback proactively uses 50% additional capacity, reverse-ordered transmission and retransmission rate that is clocked by the receiving ACKs. In the following, we will experimentally show that these are good design decisions for Halfback.

**Additional bandwidth:** We choose Proactive TCP (100% additional bandwidth used) and TCP (0% additional bandwidth used) to see how additional bandwidth used may affect the feasible capacity. Both mechanisms have the same startup phase and retransmission rate and direction. As shown in Fig. 4.16, Proactive TCP’s feasible capacity is only 45% and that of TCP is 90%. Therefore, excessive bandwidth overhead can cause severe safety problems. But without additional bandwidth, just like JumpStart and TCP, the mechanisms cannot achieve good enough latency performance. In our design, we try to efficiently use limited (50%) additional bandwidth to achieve small FCT with relatively large feasible capacity. It is also possible to dynamically tune the additional bandwidth used for proactive retransmission according to the history of network conditions (e.g. instead of sending one retransmission for each ACK, we could send two retransmissions for every three ACKs). The tradeoff of that scheme is an interesting open question for future research.

**Retransmission direction:** Here we test a new scheme, Halfback-forward. Halfback-forward and Halfback both use pacing startup, 50% additional bandwidth in proactive retransmission at same rate. The only difference is that Halfback-forward proactively retransmits packets in forward order instead of reverse order. The feasible capacity of Halfback-forward falls to 35% comparing to Halfback’s 70%. This is because first half of the flow is much less likely to have packet loss than second half of the flow. Therefore, the additional proactive transmission is effectively wasted and simply adds unnecessary utilization on top of normal retransmission.

**Retransmission rate:** To choose the proactive retransmission rate, we test Halfback and another new scheme, Halfback-burst. The only difference between these two mechanisms is Halfback proactively retransmits at a rate matching received ACKs, while Halfback-burst uses line rate for retransmission. Halfback-burst’s feasible capacity is significantly smaller than that of Halfback since line rate is much larger than the available bandwidth of the bottleneck link. This causes many retransmitted packets to be lost and wastes the bandwidth used in proactive retransmission.

In sum, Halfback’s ROPR does make good design decisions with proactive, reverse-ordered and pacing-based retransmission. Without any one of them, ROPR, and thus
Figure 4.16: FCT and feasible capacity of mechanisms with different startup phases and lost packet recovery mechanisms.

Halfback, will not work effectively. Finding an even better trade-off is conceivably possible and would be an interesting area of future work.
Chapter 5

Quantifying Router-level Path Inflation in the Internet across Years

5.1 Causes of Path Inflation

The distance between the University of Illinois at Urbana-Champaign and San Jose is around $2.92 \times 10^3$ km which means that Direct Optical Fiber (DOF) latency, the time spent by light on the geographical direct fiber between these two cities, is around 14.6 ms ($2.92 \times 10^3 km / (2 \times 10^8) m/s$). However, based on our measurement, the one-way latency between these two positions is able to be as high as 37.95 ms, 2.6 times the ideal one-way latency.

This kind of latency inflation can be caused by many factor. For example, the design of the network topology is able to increase the inflation due to its lack of directed links. Routing protocols may also cause the path inflation since the shortest path may not be chosen to carry the traffic. In this section, we try to quantify the path inflation in router-level to better understand how much improvement we are able to achieve when we design new protocols or future networks.

In the following sections in this chapter, we define the **path inflation** as the rate for the sum of the distance between two adjacent routers on the end-to-end path over the length of the DOF between these two hosts. It is different from the **latency inflation**, which is equal to the one-way latency between the two hosts measured by Ping over the time spent by light on the DOF.

5.1.1 Dataset

To analyze the causes of inflation in router-level, we need the following data:

**Internet topology:** This data is used to identify whether there exists a direct link between two routers or not. In our analysis, we use the data from CAIDA in December 2014 [19]. This dataset is collected by traceroute from 94 Ark monitors located in 36 countries, which includes 64.6 million nodes. To scale down the analysis, we combine all the nodes of one ISP in the same city as one Point of Presence (POP) [101]. After this combination, there are 250,590 POPs in our dataset.

**Real route information:** The latency inflation between two hosts can be affected by many factors, e.g. the co-existing traffic, which makes this inflation varied across time [107]. To make the problem simple, instead of analyzing the fluxible latency

\[ \text{The speed of light in fiber is around 1.5 times of that in vacuum} \]
inflation, in this work we try to figure out the causes of path inflation which equals the
distance of the end-to-end path over the distance between two hosts. In this case, we
need the real route information between the two hosts which is able to be collected by
traceroute.

**IP geolocation:** Since we need to calculate the sum of the distance between each of
two adjacent routers, we need the geolocation for all the devices included. However,
this kind of information is not open to all people for various reasons, such as business
competition and security problems. For example, when the attacker has the geolocation
information for some important routers, they are able to cause havoc to the whole
network by breaking down this important infrastructure facilities.

However, since the geolocation information is very useful for many applications,
e.g. CDN, there are many projects try to get a relatively accurate geolocation for the
routers [110, 85, 60]. In section 5.2 we will demonstrate how different IP geolocation
databases may affect the result of the path inflation. In this section, we use the data
collected in December 2014 by IPligence [11], which started to provide geolocation
service in 2008. The IP geolocation in this database is around 93.9% [97] accurate
when compared with CAIDA’s ground truth database [67], which includes the real
geographic location of a small number of routers.

**AS relationship:** In this section, we also want to figure out how valley-free and prefer-
customer policies increase the path inflation and to compare these results with the to-
tal inflation caused by inter-domain routing protocols. To achieve this, we need the
customer-provider relationship between each of two ASes to calculate the shortest path
with the corresponding policies. In this section, we use the AS relationship collected
by CAIDA in 2014 [18].

### 5.1.2 Inflation Contributors

In the following analysis, we quantify the inflation contributors into two categories:
network topology and routing protocols. For network topology, we consider the infla-
tion caused by lack of direct links. For routing protocols, we separate them into three
parts: intra-domain routing protocols (used to select the path within one ISP), inter-
domain routing protocols (used to select the sequence of ISPs for the end-to-end path)
and peering policies (used to select the path between two ISPs).

**Network topology:** It is impossible to have direct links between any two hosts since
we cannot have so many routers and fibers. Therefore, for the source and destination
hosts, they need to jump multi-hops to reach each other, especially when both of them
are located in small cities. We define the inflation caused by this reason as topology
inflation.

**Inter-domain routing protocols:** Currently, the Internet uses Border Gateway Pro-
tocol (BGP) [91, 103] as the inter-domain routing protocol, which allows the Au-
tonomous Systems (ASes) to propagate routing information and choose AS routes. In
this paper, we analyze the path inflation caused by two typical BGP routing policies,
valley-free and prefer-customer [56, 26, 69]. In addition, since the Internet Service Providers (ISPs) may also evolve some other policies to choose the inter-domain route, we also use the AS path of the real route collected by traceroute to analyze the total path inflation caused by inter-domain routing protocols.

**Peering policies:** In one AS, the routers that know how to forward packets from current AS to the next AS hop are gateway routers. Peering policies are used to choose the in-/out-gateway routers when the packet is forwarded from one AS to the other AS. There are many peering policies, such as early-exit, late-exit, and best-exit. In this analysis, we use the same gateway routers found by the real route to figure out the inflation caused by peering policies.

**Intra-domain routing protocols:** There are many intra-domain routing protocols, such OSPF [80], RIP [68], static routing [63], and so on. Different ASes may have their own intra-domain routing policy, which increases the difficulty of analyzing the inflation caused by intra-domain routing protocols. Some previous works [101, 106] try to create their own model to simulate the routing process in one ISP. However, this kind of simulation causes inaccuracy for the analysis of intra-domain routing policy inflation. In our work, instead of directly figuring out the intra-domain inflation, we use the indirect way (reducing all the other inflations, device-lacking, peering and inter-domain routing inflations from the total inflation) for our calculation.

### 5.1.3 Inflation for Different Contributors

Figure 5.1 demonstrates the inflation contributed in December 2014 by the four factors in Section 5.1.2. We use the DOF distance as the baseline. The inflation caused by different contributors equals the distance of the shortest path with corresponding policies over the distance of DOF. Below is how we calculate the inflations in the...
A. Topology (Top.) inflation is the distance of the shortest path in the current Internet topology over DOF distance.

B. Topology and valley-free (VF) inflation is the distance of the shortest path in which there is no AS providing transit services between any two of its providers or peers over the length of DOF.

C. Topology, valley-free, and prefer-customer (PC) inflation is the distance of the shortest path with valley-free and prefer-customer policies over DOF distance. In addition to valley-free, prefer-customer policy requires the ASes to choose the route from customer AS instead of from peering or provider AS, even if the latter is shorter.

D. Topology and observed AS path (ASP) inflation is the distance of the shortest path which visits the same sequence of ISPs of the end-to-end path collected from traceroute over the length of DOF. This inflation is slightly different from C since some ISPs may not conform to Gao-Rexford [56] policies.

E. Topology, AS path, and peering inflation is the distance of the shortest path with the same AS route and peering links (the links connecting two nodes in different ASes) of the end-to-end path over the DOF distance.

F. Observed traceroute inflation is the distance of the route collected by traceroute over the length of DOF.

According to the results, physical contributors, e.g. lack of routers or direct fibers, increase the path inflation by 0.82% in the median and by 18.73% at the 90th percentile. This result shows that for the current Internet topology we are able to achieve the end-to-end latency very close to the time spent by light on the DOF.

The total inflation caused by inter-domain (the difference between A and D) routing protocols is nearly the same as the total inflation caused by prefer-customer (the difference between A and B) and valley-free policies (the difference between A and C). In the results, the median of valley-free inflation is 1.077 and the 90th percentile is 2.251. Compared with prefer-customer policy whose median inflation is 1.105, the inflation caused by valley-free policy is slightly small. This is because there are more routes satisfied with valley-free than with prefer-customer, so it is easier to find a relatively shorter path in these routes for valley-free than for prefer-customer.

Peering inflation (the difference between E and D) is the main component of the total path inflation. Based on our results, the inflation increased by peering policies is around 62.20% at the median and 289.70% at the 90th percentile, which is much larger than AS path inflation and the inflation caused by intra-domain routing protocols. It is more difficult to reduce the inflation caused by peering policies than it is to reduce that caused by intra-domain routing policies or even inter-domain routing protocols since achieving small peering inflation sometimes may require that the current ISPs have some global information for the end-to-end path.
Intra-domain routing protocols (the difference between F and E) only increase the path inflation by 29.04% at the median and 58.48% at the 90th percentile, which is small compared with the inflation caused by inter-domain routing protocols and peering policies. This is because it is easier to achieve the shortest path while there is only one ISP involved. Due to using the smallest number of hops path instead of the shortest distance path as the main component of intra-domain routing protocols and traffic engineering there are still small amounts of inflation caused by intra-domain routing protocols.

Based on this result, 0.72% of the total path inflation is caused by network topology; 18.82% of the total path inflation is contributed by inter-domain routing protocols; 54.85% of that is caused by peering policies; and the rest, 25.6%, is contributed by intra-domain routing protocols. We are able to find that the main component of the inflation is caused by peering policies.

5.2 Geolocation Validation

In this section, we demonstrate how different databases of routers’ geolocation information affect the result of total path inflation. To achieve this goal, we use the same database as in Section 5.1 for the Internet topology, the AS relationship and real Internet routes between some hosts. Except these, we use different databases for devices geolocation to compute the total path inflation between hosts based on the real route in the Internet.

5.2.1 Databases

In this section, we use following geolocation databases:

MaxMind [13] was founded in 2002 to provide the geolocation service for a range of databases, from country level to city level. In this section, we use three databases from MaxMind: MaxMind GeoLite City (MML), MaxMind GeoLit2 City (MML2) and MaxMind Geo City (MM). GeoLite City and GeoLite2 City are free and updated every month. According to their own data, the accuracy of GeoLit2 is 67% in the United States, determined by checking with some known IP address and location pairs. Geo City is a commercial database which has higher accuracy and larger coverage than GeoLite City and GeoLite2 City. Based on their own data, Geo City covers more than 99.99% of the IP addresses in use and is 81% accurate for cities in the United States within a 50 km radius.

DBIP [5] is one of the most comprehensive and accurate IP address databases. It owns more than 8 million IPv4 and IPv6 blocks. However, most of their blocks are in the United States, around 47.7%, which may reduce its accuracy for global analysis. DBIP updates its database every month and hundreds of thousands of blocks are added; for example, 755K blocks were added in November 2015.

IP2Location (IP2L) [10] is developed and maintained by Hexasoft. In addition to
the device’s geolocation, it also provides some other information, such as the ISP and network bandwidth, based on the IP address. The accuracy of IP2Location is 76.31% for cities in the United States, which is a little bit lower than that of MaxMind Geo City. They argue that this inaccuracy is due to the dynamic IP allocation of some large ISPs, such as AOL and MSN TV. This is especially true for AOL, which uses a network to route all its traffic through Reston, Virginia, which makes the geolocation services unable to determine the location of people who dial into the AOL network.

IPligence (IPLG) \cite{irving2006ipligence} has been providing geolocation service since 2006. It has 93.9% accuracy when compared with the ground truth database from CAIDA \cite{caida2004groundtruth}. This database \cite{ipligence2006database} contains the geolocation information for a small number of routers from a private database for a tier-1 ISP and a tier-2 ISP and the public database for CANET \cite{canet2004geolocation}, GEANT \cite{geant2004geolocation}, Internet2 \cite{internet22004geolocation}, I-Light \cite{i-light2004geolocation}, and National LambdaRail \cite{nationallambda2004geolocation}.

EdgeScape (ES) \cite{akamai2001edgescape} is provided by Akamai, which primarily provides a content delivery network service. Since CDN service needs to identify customers’ geolocation based on their IP addresses which are naturally fit for geolocation service, they added the geolocation service EdgeScape in 2001. Considering its worldwide scope and integration with networks, its accuracy of IP geolocation information may be higher than the other databases.

\subsection{5.2.2 Inflation for Different Geolocation Databases}

Fig. 5.2 demonstrates the traceroute inflation for around 1.9 million pairs of hosts for each database. In this result, we are able to find that the traceroute inflations are similar to each other, except that of DBIP and MML, which are slightly different. This is because for the hosts with inaccurate geolocations, their geolocations will not be far away from the correct ones. However, this difference may be smaller than the differ-
inference when we compare the databases with the ground truth database which has the real
geloadion of the routers. This is because it is easy for the geo-location databases to
make the same mistake; for example, both IP2Location and IPligence identify some
hosts in Canada to the United States. Currently, we do not have enough ground truth
data for IP mapping to generate the accurate traceroute inflation.

Among all the databases we are able to access, we choose two representative ones,
MML and IPLG, and compare path inflation caused by the factors mentioned in §5.1.2
based on these two geolocation databases. We find out that even through the data of
the path inflations are different, we are still able to get the same conclusions as §5.1.
The AS path inflation is still equal to the sum of the path inflation caused by valley-
free and prefer-customer. Besides this, the main component of the total path inflation
is still peering inflation. Moreover, valley-free inflation is still slightly smaller than
prefer-customer inflation.

5.3 Path Inflation across the Years

Path inflation was first noticed around a decade ago. Since then, a lot of researchers
have been paying attention to it. They try to quantify the causes of this phenomenon in
different directions and hope it can be reduced in future work. In this section, we want
to figure out how the inflation changed since people started paying so much attention to
it. The result demonstrates that the inflation caused by different contributors is reduced
by around 24.22% in total for the past five years.

In this section, we use the same data sets as in Section 5.1 for each year, except for
the geolocation information. Since most databases only provide routers’ geolocation
information for the current Internet, in this section we use MaxMind LiteGeo City as
the resource of geolocation, which is stored by CAIDA, from July 2010 to December

---

Figure 5.3: The path inflation caused by different reasons in 2014 December based on the
geloadion information provided by MML.
2014. Even through the result of total path inflation calculated from MML is slightly different from the other geolocation databases, since we only care about the trend of path inflation changed and we use the same database for geolocation information, using MML as resource of geolocation does not affect our results.

Figure 5.4: The path inflation caused by different characters for past five years.

Fig.5.4 demonstrates the median and 90th percentile results of the path inflation caused by the following: network topology, valley-free routing policy, prefer-customer routing policy, inter-domain routing protocols, peering policies, and intra-domain routing protocols. According to our results, we are able to find that the inflation caused by each factor is reduced every year, especially from 2011 to 2012. For the total inflation, it is reduced by 2.2% from 2010 to 2011, 10% from 2011 to 2012, 6% from 2012 to 2013, and 8% from 2013 to 2014. Besides this, the rate between the inflation caused by intra-domain routing protocols and that caused by inter-domain routing protocols have been relatively stable for these five years. However, the rate between that of peering policies and inter-domain routing protocols has been reduced from 1.25 to 1.08, which
means that people notice the relatively high inflation caused by peering policies, and, compared with inter-domain routing protocols, they pay more attention to reduce the inflation caused by peering policies.

5.4 Suggestions to Reduce Path Inflation

We do not suggest directly using the shortest-path routing in AS-level or as intra-domain routing protocol due to the scalability and safety problems. It has been demonstrated [24] that when shortest-path routing is used in the network, the maximum congestion will scale poorly. In addition, shortest-path routing will cause hot-spot links, and the failure of any of these links will cause a catastrophe for the whole network.

In this section, we introduce some approaches to reduce the end-to-end inflation. These approaches are able to be used in the Internet with no dependency on the network topology. Most of them can always be used no matter what kinds of AS level routing protocols are used by the Internet and what kinds of weighted shortest path intra-domain routing protocols are used by ISPs. This gives the flexibility to the networks when they choose their own routing protocols to avoid hot-spot links and increase their scalability.

5.4.1 Last AS Hop Relationship

In the real routes collected from traceroute, we find this phenomenon: \( A \leadsto B \leadsto C \leadsto D \), where \( A \) and \( D \) are in the same city and there exists direct link between \( A \) and \( D \). This kind of route increases the number of hops and sometimes also increases the total inflation when \( B \) and \( C \) are not in the city of \( A \) and \( D \). Most of the time, this phenomenon happens at the end of the route and \( D \) usually is the destination host. After finding out the ISP to which these nodes belong, we find these hops existed just because \( A \) and \( D \) are not in the same ISP, and the route needs to jump to gateway \( B \) to swap to the ISP of the destination node.

Due to this phenomenon, we suggest that the routers in the same city of the destination host be treated as the routers in the sibling ISP of the destination host. In this case, when the route reaches the city of the final destination, it is able to directly jump to the destination host if the router owns the link to the host. Fig.5.5 shows that the number of hops and inflation are reduced by this mechanism when this phenomenon occurs. According to the results, we are able to find that this mechanism may not reduce much inflation (Fig. 5.5.a) since usually this kind of detour will choose the gateway nodes nearly the destination. However, if we consider the number of hops (Fig. 5.5.b) reduced, this mechanism helps a lot: 17.8% in the median. As we know, transmission latency, which monotonically increase with the number of hops is usually larger than propagation latency which is monotonically increasing with the total length of fibers, so even this mechanism cannot reduce much path inflation, but it is able to reduce much latency inflation caused by the last few jumps.
5.4.2 Intra-domain Routing Policy

Most ISPs use weighted shortest path as their intra-domain routing protocols. The weight of each link is assigned based on multiple characters. For example, when providers decide the weight, they may consider the bandwidth and the amount of traffic to go through to avoid congestion. The primary factor of the weight usually is the number of hops. There are many reasons make the providers use the number of hops instead of the length of the link as the primary factor. First, the transmission delay is usually larger than the propagation delay, so the path with the smaller number of hops and longer total length of fibers is able to have relatively small latency inflation (the rate for the real end-to-end latency over the time for light spent on the DOF between the source and destination hosts). In addition, the route with the smaller number of hops causes the smaller total amount of workload for the routers on the end-to-end
path. Finally, using number of hops as the primary character is easier to be deployed compared with length of links. Due to the reasons, even though using the number of hops as the primary factor of weight may increase the path inflation, most providers still chose it as their intra-domain routing protocol.

However, we argue that the length of links is also an important factor. According to the analysis in Section 5.1, we are able to find that using link length is able to reduce the total path inflation up to 29.04% in the median which is around 25.6% of the total inflation. Besides this, the configuration of a newly added router is not complex. The only additional work is calculating the length of the link while setting up the connections between this newly added router and the others. Compared with the number of hops, the length of links needs to reconfiguration when the location of the router is changed. However, if the router is moved in a small area, e.g. in one building or in one city, it is not necessary to reconfigure the weight of the link due to the small change of the length and the long distance moving of the routers usually is not common. Considering these reasons, we suggest the providers use the length of links as a factor of the weight for links—for example, using the total length of the route to break the tie for the routes with the same number of hops—to reduce the path inflation caused by intra-domain routing protocols.

5.4.3 Peering Policy

The inflation caused by peering policies is around 54.85%, which is the largest one among all the factors. Reducing this kind of inflation is very difficult due to the ISPs’ lower degree of cooperation. The peering policies are able to be separated into three categories. The first one is early-exit or hot potato peering policies in which the current ISP will forward the packet to the next AS hop as soon as possible. The second one is late-exit peering policies, for this peering policy, the current ISP will choose the come-out gateway router which is closest to the destination host as the peering node. Late-exit peering policy requests the gateway routers with large memory since the come-out gateway router depends on the prefix of the destination host. The last category is peering policies, which is in the middle of these two.

Fig.5.6 shows the distance of the first and second AS hop, normalized by the DOF distance, for the route with two AS hops. Based on this result, we are able to find that there exists a primary AS hop which contributes nearly all the distance between the source and destination hosts. For the optimal route, i.e., the shortest path with the same AS path for the real route, most have the primary AS hop in the first AS hop. The result of traceroute is closer to the optimal situation. For the early-exit peering policy, the primary AS hop slightly prefers to be the second one. This result demonstrates that in the current Internet, most ISPs do not always use early-exit peering policies; instead, their choice is closer to the optimal ones.

Fig.5.7 demonstrates the location of the primary AS hop for the optimal route (the shortest path for this AS route) with the number of AS hops from 2 to 6. With this result, we are able to figure out that the primary AS hop is usually in the middle of the
AS routes and the size of these ISPs are relatively large. Since large ISPs usually have more direct links, especially for the links which cross countries or continents, they are a good candidate for the primary AS hops. Besides this, the ISPs with many prefixes are usually small, compared with the tier-1 or tier-2 ISP, so the first and last few ASes on the end-to-end path are relatively small, which means the ASes with primary AS hop are always in the middle of the AS routes. In this case, the source node needs to jump a few AS hops to the large ISP, which is the AS of the primary AS hop, and then jump back to the AS of the destination nodes.

According to previous results, we suggest the AS choose the peering policy based
on its own size. If the size of the ISP is large, the ISP should choose its peering policy to be late-exit or the peering policies closer to late-exit to make itself the primary AS hop. If the size of the ISP is small, then the ISP should choose its peering policy as early-exit or the peering policies closer to early-exit and forward the packet to the ISP of the primary AS hop as soon as possible. This mechanism is able to save some inflation caused by the small ISP; it also moves some inflation caused by peering policies to that caused by the large ISP’s intra-domain routing protocols, which is easier to be reduced. For deployment, this mechanism does not need any cooperation between the ISPs. In addition, the gateway routers in the large ISP usually are able to handle the workload of the late-exit due to their large memory and fast computing speed.
Chapter 6

Conclusion and Future Work

Short flows are widely used and highly valuable in the modern Internet as lots of applications use them, such as in the form of web requests, to make interaction with the users. This kind of flow is sensitive to user-perceived latency and a small additional delay is able to cause the loss of customer attention and revenue.

In the past a few decades, a lot of works have tried to analyze the causes of latency and to reduce it for short flows in different directions [79, 53, 72], such as reducing connection establishment time, fast recovering packet loss, saving time spent on TCP’s slow start and so on. In this work, we try to reduce the flow completion time of the short flows as close as possible to one RTT. Besides this, we also intend to break down the latency inflation on the end-to-end path into its contribution factors and hope to inform future research on latency reduction for short flows.

6.1 Thesis Achievements

Below are the achievements of this work:

• We design ASAP, a new low-latency transport protocol for wide-area networks. ASAP revisits classic Internet design decisions by modifying and merging functionality of DNS and TCP to substantially reduce connection establishment delay, benefiting interactive communications such as web browsing.

• We evaluate transport-layer performance of ASAP which includes downloading time of individual files, computational overhead, and vulnerability of DoS attacks. According to our evaluation, ASAP is able to save around 100 to 200 ms for downloading a file with size from 1KB to 500KB. The computational time for PVs, servers, and DNS servers is smaller than 1 ms and that for client is around 1 to 2 ms. Besides this, we also demonstrate that two PVs with random locations are sufficient to make the ASAP less vulnerable than TCP in 75% cases.

• We experimentally compare existing solutions for reducing data transmission latency and, more importantly, the trade-off between latency and safety at both the flow level and the application level. We argue that existing solutions are still operating away from the sweet spot on this trade-off plane.

• We designed Halfback, a new aggressive transport scheme for short flows, based on the diagnosis of existing solutions. Halfback substantially reduces transmis-
sion delay and achieves high sending rate quickly. In addition, with the help of Reverse-Ordered Predictive Retransmission, Halfback works well for challenging situations, like high utilization networks, with limited effect on competing TCP flows. Finally, as Halfback only requires changes in the sender and is TCP-friendly, it is feasible to deploy into the current Internet.

- We also quantify the router-level path inflation on a large scale across five years. According to our result, the main component of the path inflation is caused by peering policy even though the proportion of it has been reduced in the past a few years. Besides this, the total path inflation has been reduced around 22% since 2010.

- In addition to providing analysis, we also give some suggestions: 1) consider routers in the destination city as with sibling relationship of destination host; 2) choose peering policies based on ISP size; and 3) consider total fiber length in intra-domain routing policies to reduce the path inflation or latency inflation caused by peering, inter-domain, and intra-domain routing policies. These suggestions only require small changes for the corresponding policies and are easy to be deployed into the current Internet.

6.2 Future Work

The necessity to reduce the flow completion time of short flows will never end. Here we discuss several of the most interesting and challenging research directions that we still have to solve for this topic.

Evaluating application-level performance of ASAP for downloading a multi-object web page: Downloading a multi-object web page may compound ASAP’s benefit (because the browser often needs to open TCP connections to multiple servers in serial) and also may reduce ASAPs relative benefit (when connection establishment latency is dwarfed by other delays). Evaluating such application-level performance is an important consideration for future work.

Building-in Halfback into a web browser and evaluating the performance: To add Halfback into a web browser, we need to implement it in kernel layer, which may reduce its performance due to some system problems. For example, we use pacing strategy in Halfback, which may result in a high CPU utilization. Besides this, concurrent connections may also reduce the performance of Halfback.

Combining Halfback and ASAP: ASAP tries to reduce the time spent on connection establishment and Halfback tries to reduce the time for data transmission. Combining them helps us to achieve flow completion time as two RTTs and all data received in one RTT. However, there is some additional work needed, such as RTT estimation during connection establishment.

Evaluating the suggestions for path inflation deduction: We give some suggestions to reduce path inflation in §5.4. However, we didn’t evaluate how these suggestions
may work based on the real traffic records in the Internet. This kind of evaluation will be very useful for us to design mechanisms to reduce path inflation or latency inflation.
References


Appendix A: Bounding the number of PVs

We model a network as an arbitrary graph $G$ with $n$ vertices $V$, and assume that it employs fixed but arbitrary single-path routing. That is, when $s$ sends a message to $d$, it follows a specific arbitrary path $P(s,d)$; these paths may or may not be related to shortest paths in the network. The attacker can eavesdrop on some set of locations $A \subseteq V$, and therefore can eavesdrop on $s \rightarrow d$ traffic when $P(s,d) \cap A \neq \emptyset$.

**Definition 1** A source-destination pair $(s, d)$ is **attackable** in a protocol (TCP or Halfback) for a given set $E$ if the attacker can cause $s$ to send a flow of data to $d$. A destination $d$ is **attackable** if there exists a source $s$ for which $(s, d)$ is attackable. If there are $\geq n/2$ such sources, then $d$ is **highly attackable**.

We assume that Halfback uses a set $P$ of PVs which are trusted by all servers. Therefore, if any $d$ is attackable in Halfback, then $(s, d)$ is attackable for all sources $s$.

**Theorem 1** Suppose $(k + 2)\log_2 n$ PVs are placed in uniform-random locations, and the attacker eavesdrops on an arbitrary set of $k$ locations after knowing where the PVs are placed. With probability $\geq 1 - \frac{1}{n}$ (over the choice of PV locations), any destination that is attackable in Halfback is highly attackable in TCP.

**Proof.** Fix any destination $d$ and attacker locations $A$. Let $S_A$ be the set of sources $s$ for which the attacker can eavesdrop on the path $s \rightarrow d$, and let $f = |S_A|/n$. If $f \geq \frac{1}{2}$, then $d$ is highly attackable in TCP and the theorem holds for this $d$.

Otherwise, if $f < \frac{1}{2}$, for Halfback, $\Pr[d \text{ is attackable}] = \Pr[P \subseteq S_A] = f^{|P|} < 2^{-|P|}$. Now, we want to bound the probability that any of the $n$ possible destinations is attackable for any of the $\binom{n}{k}$ possible sets $A$. By a union bound over these $n\binom{n}{k}$ events, the probability that any bad event happens is $< n\binom{n}{k}2^{-|P|} \leq nk^{k+1}2^{-|P|} \leq \frac{1}{n}$ since $|P| \geq (k + 2)\log_2 n$. 

75