Halfback: Running Short Flows Quickly and Safely

Qingxi Li, Mo Dong, and P. Brighten Godfrey
University of Illinois at Urbana-Champaign

ABSTRACT

Interactive applications like web browsing are sensitive to latency. Unfortunately, TCP consumes significant time in its start-up phase and loss recovery. Existing sender-side optimizations use more aggressive start-up strategies to reduce latency, but at the same time they harm *safety* in the sense that they can damage co-existing flows' performance and potentially the network's overall ability to deliver data. In this paper, we experimentally compare existing solutions' latency performance 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. Based on the diagnosis of existing solutions, we introduce Halfback, a new short-flow transmission mechanism that operates on a better latency-safety trade-off point: Halfback achieves lower latency than the lowest latency previous solution and at the same time significantly better safety. As Halfback is TCP-friendly and requires only sender-side changes, it is feasible to deploy.

CCS Concepts

•**Networks** → **Network protocol design; Transport protocols;** *Network experimentation;* **Network protocol design; Transport protocols;**

Keywords

TCP; short flows; flow completion time.

1. INTRODUCTION

Short flows 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 [30]. For this kind of flow, user-perceived latency is the key challenge as even relatively small delays can cause the loss of customer attention and revenue.[1]

The Internet's current transmission protocol, TCP, tries to use bandwidth conservatively, but sacrifices the latency of short flows. As TCP's slow-start needs multiple RTTs to detect its fair sending rate, many short flows do not 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 [33] 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 [12], RC3 [26] and QuickStart [32], and therefore have not seen any significant deployment. Others, such as Proactive TCP [18], increasing the initial congestion window to 10 [6, 15] and JumpStart [25] 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 trade-off* space and find the sweet spot.

In this paper, we evaluate existing solutions in terms

CoNEXT'15, December 01-04, 2015, Heidelberg, Germany

ACM ISBN 978-1-4503-3412-9/15/12.

DOI: http://dx.doi.org/10.1145/2716281.2836107

[1] A 400 ms additional delay in Google web search caused a 0.74% drop in search frequency after 4 to 6 weeks [11]. The revenue per person in Bing was reduced by 1.2% with a 500 ms delay or 4.3% with a two second delay [34].

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, Reactive TCP [18], increasing initial congestion window to 10 [6, 15] (referred to as TCP-10 in this paper), PCP [7] and JumpStart [25]. 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, 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 reactiveness 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.

We evaluated Halfback with both flow-level and application-level benchmarks. We give highlights of a subset of results here. For flow completion time, we evaluated Halfback against the aforementioned existing mechanisms on PlanetLab across the global Internet with 2,600 node pairs. We found that 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 evaluate Halfback's FCT-vs.-safety tradeoff compared with existing approaches. The results show that compared with JumpStart (the FCT winner among past

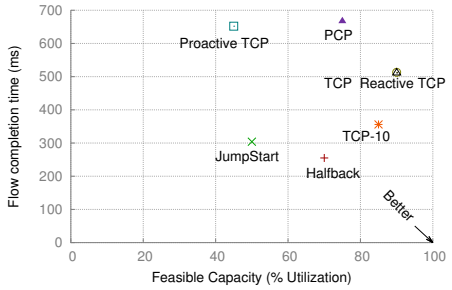


Figure 1: Tradeoff between common case latency (y axis) and feasible capacity in a pessimistic case of high workload from short flows (x axis).

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 592ms (22%) page load time reduction at 30% network utilization and significantly improves feasible network utilization by $1.57\times$ comparing to JumpStart.

Summary of key contributions. **(1)** We implement several existing latency mechanisms that had not been previously compared head-to-head, including JumpStart, Reactive TCP, PCP, Reactive TCP and Proactive TCP, and experimentally evaluate their latency performance at large scale (2.6K pairs of hosts across five continents) in an Internet environment and investigated their flow-level and application-level latency-safety trade-off using Emulab. **(2)** We design a novel short-flow transmission optimization mechanism, Halfback, including the ROPR mechanism that achieves fast packet loss recovery and limits retransmission aggressiveness. **(3)** We implement Halfback and evaluate its performance in the same environment as the other mechanisms and find that Halfback achieves better latency-safety trade-off at both the flow level and the application level and better TCP friendliness.

2. BACKGROUND

2.1 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

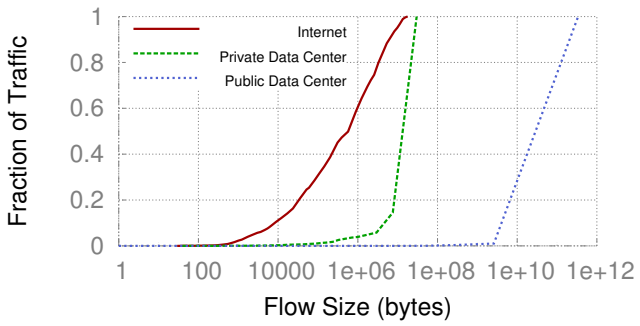


Figure 2: ***CDF of fraction of traffic carried by different size of flows using measurements from [30] [9] [21]***

change because even hundreds of milliseconds affect user behavior and revenue. For example, [2] 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 [18] 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 [19]). Fig. 2 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 [30] even though more than 95% [6] of web transfers are smaller than this size. Furthermore, as noted in a recent forecast report [1], 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 [9] and a public [21] data center, less than 1% of transmitted bytes were in flows smaller than 141KB. Therefore even Proactive TCP [18], which doubles the workload created by short flows, only increases network utilization by 0.2% to 10.4% (i.e., $20\% \cdot 1\%$ to $30\% \cdot 34.7\%$) in these environments when applied to flows smaller than 141KB.

2.2 Overview of Existing Solutions

Several works have developed approaches deployable at end-hosts which aim to reduce latency for short flows. PCP [7] 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 [18] 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, it is still to 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 [25]. 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, Jump-

Start 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 [25] 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 2.1, 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.

3. 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.

3.1 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 [25]: 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 (§3.3). 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% [6] 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.

3.2 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

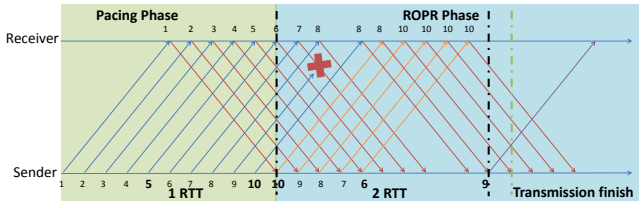


Figure 3: ***Halfback transmits a 10-packet flow***

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 § 2.1.

RC3 [26] 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.

3.3 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 §3.1, 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 [22,35].

3.4 Example

In this section, we walk through an example 10-segment flow transmitted by Halfback. Fig. 3 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 3, 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, TCP is too conservative for short flows and in §4.3 we show that this additional bandwidth will not cause problems for the whole network and co-existing flows.

4. EXPERIMENTS

In this section, we conduct a perfomance evaluation of eight schemes to optimize short flow latency: TCP, TCP-10 (set ICW to 10) [6, 15], TCP-Cache (caching older values of the cwnd and ssthresh), JumpStart [25], PCP [7], Reactive TCP [18], Proactive TCP [18], 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.

4.1 Experiment Settings

Protocol Parameters: We use code from the PCP project directly [5]. For each of the other mechanisms, we implemented the scheme within UDP-based Data Transfer (UDT) [4] 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 [15]. In our evaluation, we still use 2 segments as the default initial window size for TCP protocols (except TCP-10). Note that although [6,15] suggested to set the ICW to 10 segments, it is not universally deployed.[2] Halfback sets

[2]We used the method of [2] 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.

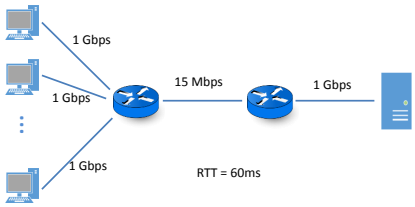
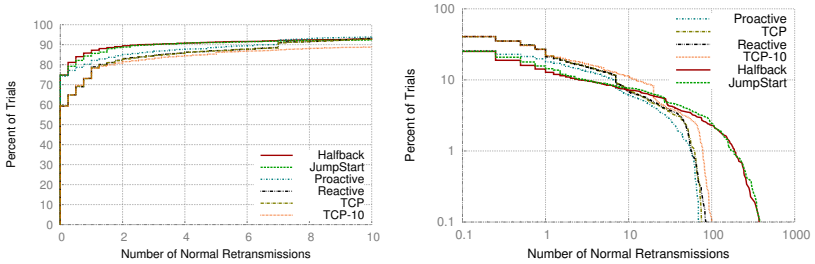


Figure 4: The configuration of the experiments in Emulab



(a) CDF (b) Complementary CDF

Figure 5: ***The number of normal TCP retransmissions of short flows in Planetlab experiments***

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 (§ 4.2.1), 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 (§ 4.2.2) 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 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 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.

4.2 Flow-level Benchmarks: Latency

4.2.1 Global Internet Evaluation on Planetlab

Fig. 6 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. 7), 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

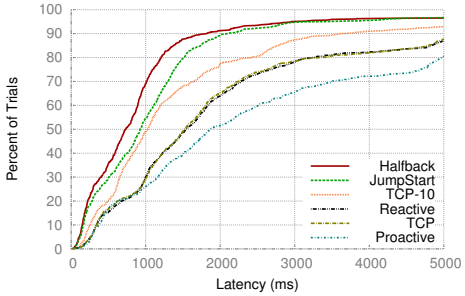


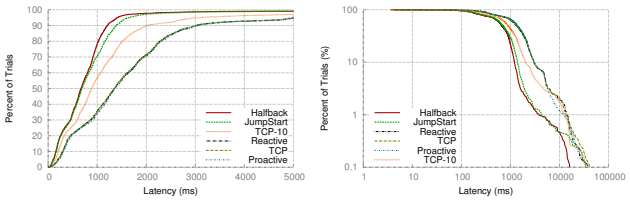
Figure 8: CDF of FCT under cases where packet loss happened

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. 8 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 (Fig. 5). In general, the network utilization is low in PlanetLab and therefore JumpStart 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.

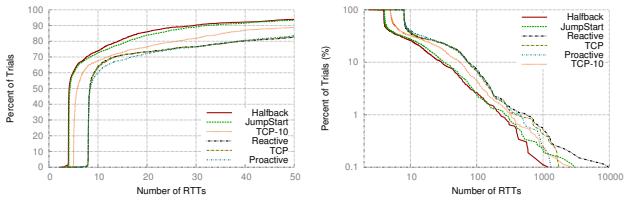
4.2.2 Home Access Networks

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 § 4.2.1 with four clients behind four different home networks and 170 servers 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 rep-



(a) CDF (b) Complementary CDF

Figure 6: ***The flow completion time of short flows in Planetlab experiments***



(a) CDF (b) Complementary CDF

Figure 7: ***The number of RTTs used in the transmission of short flows in Planetlab experiments***

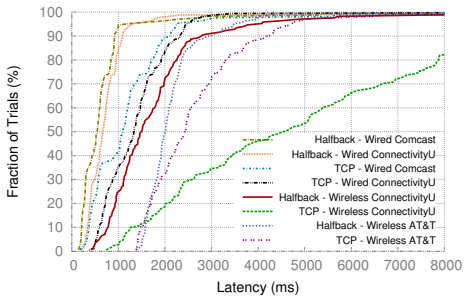


Figure 9: CDF of FCT on home networks with different providers

resentative of the individual providers' general performance.

Fig. 9 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.

4.2.3 *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. 10(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.

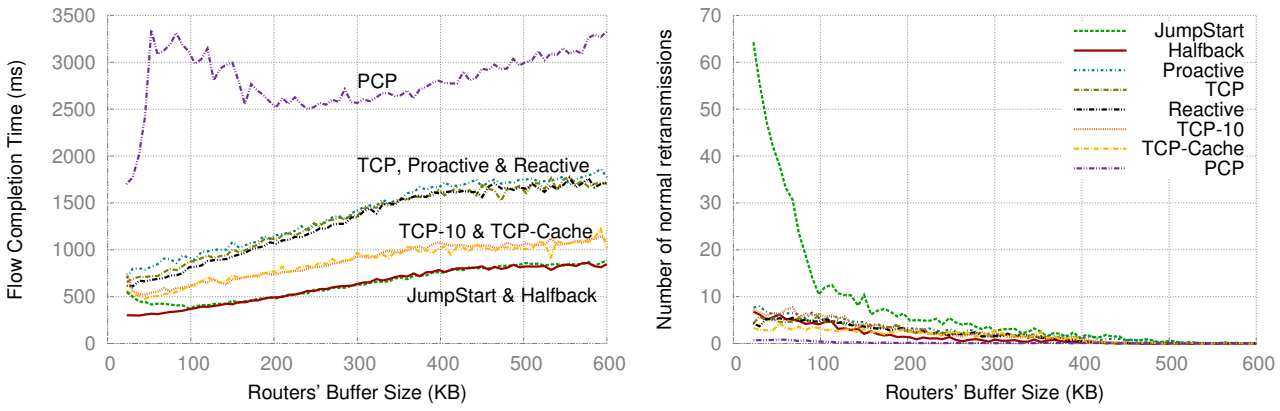
PCP does not perform well when it co-exists with TCP. A PCP sender uses probing to estimate the queue length on the end-to-end path. It will not send data, except probing, when the queuing delay is increasing during the probing. But the competing TCP senders keep building up the queue, so that PCP is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

Fig. 10(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.

4.2.4 *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 [30], a 1500-node cluster in a Microsoft data center network [21], and a private enterprise data center network [9][3]. We truncate the distributions and set the maximum flow size to be 1 MB (as longer flows

[3]Original data sets were not available; the distributions here were approximated from figures in the publications.



(a) Mean flow completion time

(b) Number of normal retransmissions

Figure 10: ***The performance of short flows for different router buffer sizes***

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. 11. 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) with constant utilization and flow size distribution. Real-world use of TCP-Cache, where senders encounter a diverse 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. 11 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 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.

4.3.1 Short Aggressive vs. Short Aggressive

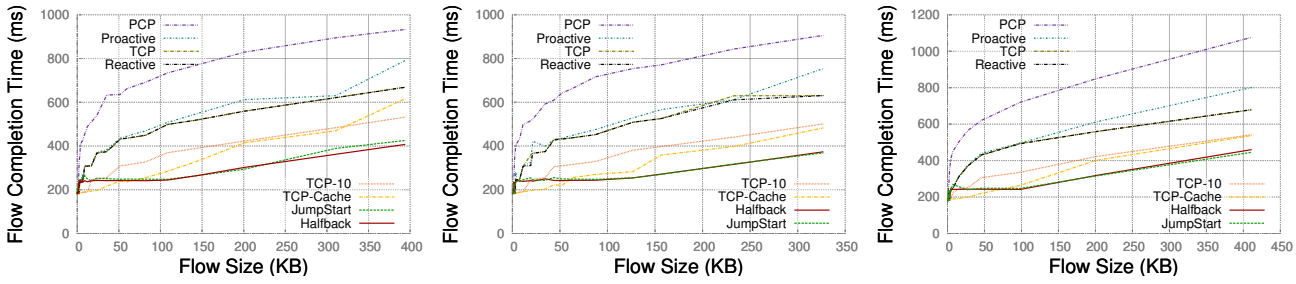
We begin with the most demanding environment: aggressive short flows competing with each other under high utilization. All flows run the same protocol, so there is no issue of TCP-friendliness, but all flows are short and hence all incur overhead to achieve low latency. This is a pessimistic scenario, because as noted in [1], 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. 12) 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.

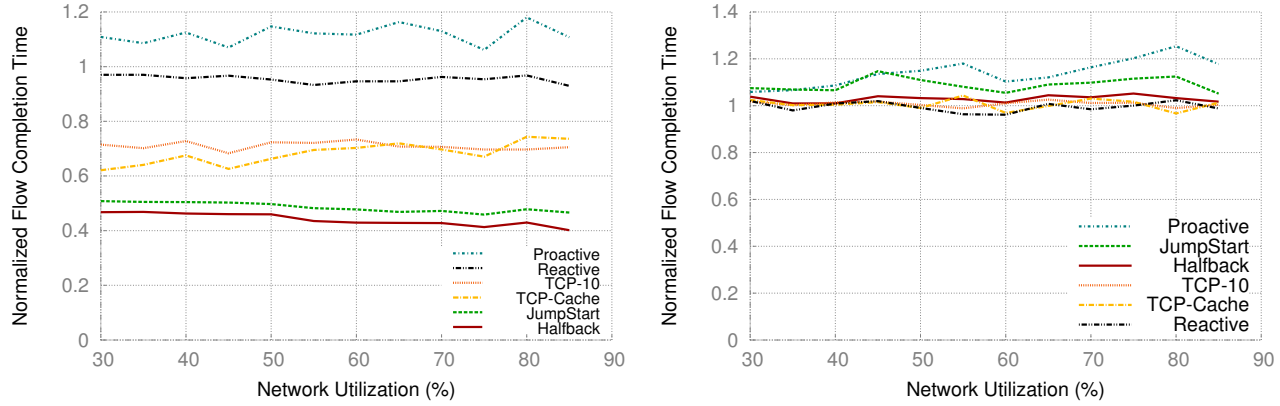
4.3.2 Short Aggressive vs. Long TCP

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 dif-



(a) Internet (b) Benson (c) VL2

Figure 11: ***Flow completion time for different flow size under 25% network utilization for different network traffic distributions***



(a) Short flow (b) Long flow

Figure 13: 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

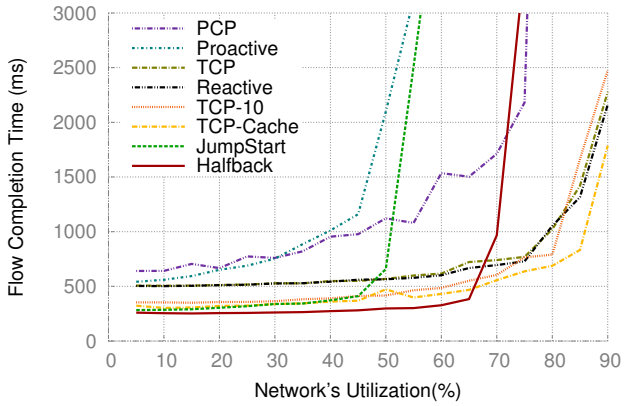


Figure 12: The performance of different mechanisms for different network utilizations while there are only short flows

ferent schemes use the same schedule of flow arrivals for each network utilization.

Fig. 13 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.

4.3.3 *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 14 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 the factor change in FCT for each kind of flow due to co-existence.

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

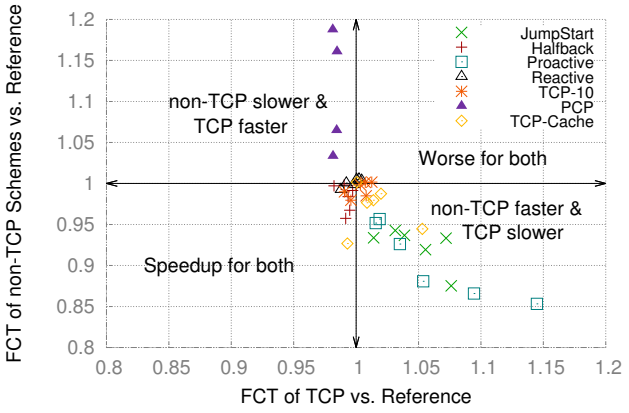


Figure 14: TCP-friendliness of different non-TCP mechanisms

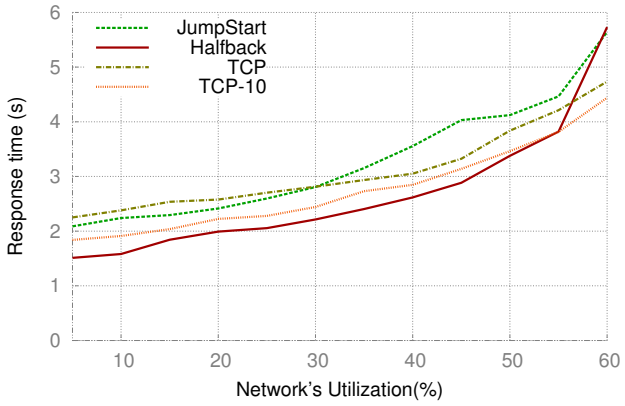


Figure 16: Average response time for different machanisms under different network utilization

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 §4.2.3, its FCT is increased while co-existing with TCP.

4.3.4 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. 15.

Ideally, we would like the throughput of the background flow and short flow to be like Fig. 15(a) in the sense that TCP recovers quickly and short flows finish transmission fast. When we employ Halfback for the short flows (Fig. 15(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 180ms to achieve half bandwidth and around 2s, 1s longer than that when we employ TCP for short flows (Fig. 15(c)), to achieve full bandwidth.

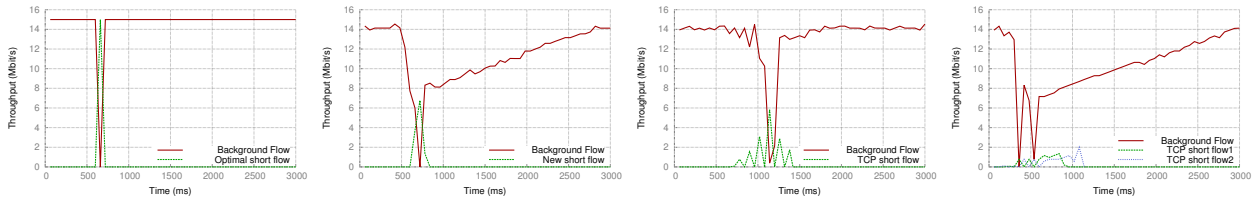
However, this effect of throughput is mitigated by several facts. First, the background flow can quickly achieve half its former bandwidth, and could recover even more quickly with a protocol like PCC [14]. Second, and most importantly, the same effect can be caused by short TCP flows. In the current 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. 15(d)). In this case, it needs about 2.7s to recover full bandwidth and the short flows still have longer FCT than Halfback.

4.4 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 [3] 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 16, 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-level benchmark result in §4.3.2. However, since the average net-



(a) a (b) b (c) c (d) d

Figure 15: ***Throughput of flows for (a) Optimal situation (b) Halfback (c) One TCP (d) Two TCP flows with half flow size***

work utilization is only 20% to 30% [19], 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.

5. 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 1, we list several different schemes that can be used for initial start-up 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. 17, 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. 17, Proactive TCP's feasible capacity is only $\tilde{4}5\%$ 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 trade-off 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 Halfback, will not work effectively. Finding an even better trade-off is conceivably possible and would be an interesting area of future work.

6. RELATED WORK

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

Startup Phase: Many projects have been proposed to accelerate the startup phase of TCP. These can be separated into five categories. **(1) Aggressive startup:** [26] uses a high initial sending rate and requires routers to support priority

	Startup phases	Lost packet recovery		
		additional bandwidth	retransmission direction	retransmission rate
Slow start	2-segment initial cwnd	0%	Original ordering	Pacing
	10-segment initial cwnd	50%		
Pacing	Pacing whole flow in one RTT	100%	Reverse-ordering	Line rate

Table 1: Different kinds of startup phase and lost packet recovery schemes.

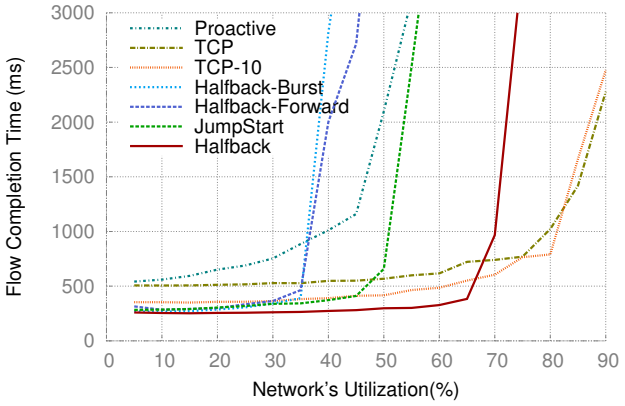


Figure 17: FCT and feasible capacity of mechanisms with different startup phases and lost packet recovery mechanisms.

queues to avoid negative effect. [6, 15] propose higher initial congestion window size, and [25] proposes JumpStart, all of which we have evaluated here. **(2) Sharing information** between connections or hosts to make flows start at a more appropriate rate: These mechanisms need complex cooperation [8] or even additional bandwidth [10]. They also increase the difficulty of deployment by introducing a new protocol. **(3) Bandwidth estimation** [13, 16] may not be accurate as end hosts lack real-time visibility into the network unless the routers explicitly mark available bandwidth in the header [32] which increases the difficulty of deployment. We experimentally evaluated one bandwidth estimation scheme [7]. **(4) Caching** schemes [28] 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. **(5) Faster connection setup** mechanisms like TCP Fast Open [31] and ASAP [37] 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 this paper include the connection setup time without optimization.

Packet Loss Recovery: Two proposals [23, 36] 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 [18] 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. We have quantitatively evaluated the proactive scheme of [18] in this paper.

Bufferbloat: Bufferbloat [20] 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, most recently PIE [29] and CoDel [27], have been proposed to reduce queueing delay. Note that reducing queuing delay (and thus RTT) is fully complementary to our study of reducing the *number of RTTs* in a flow; the improvements multiply. Also, the mechanisms we study here offer immediate benefit to a sender-receiver pair, without requiring router or network configuration changes. In addition to AQM algorithms, [17,24] reduce the effect of bufferbloat at the end host by adjusting the receiver's buffer size. This does not directly address FCT for short flows.

7. CONCLUSION

This paper performed a measurement study of how well existing proposals optimize flow completion time for short flows while remaining safe to deploy. Based on our measurement understanding, we designed Halfback, a new aggressive transport scheme for short flows. Halfback substantially reduces transmission 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. Some interesting open questions remain to be answered, including theoretical modeling and analysis of Halfback, emulation with more

complex topologies and larger scale evaluation on end-user access networks.

8. ACKNOWLEDGEMENTS

We sincerely thank our shepherd Alex Snoeren for detailed and valuable discussion and suggestions. We thank all our reviewers for their valuable comments. We gratefully acknowledge the support of NSF Grant 1149895, a Google Research Award, and an Alfred P. Sloan Research Fellowship.

9. REFERENCES

[1] Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper. goo.gl/rUcCIG.

[2] Google and Microsoft Cheat on Slow-Start. Should You? http://blog.benstrong.com/2010/11/google-and-microsoft-cheat-on-slow.html.

[3] Top 500 most popular website worldwide. www.alexa.com/topsites/global.

[4] UDT: UDP-based data transfer. http://udt.sourceforge.net/.

[5] User-level PCP test code. http://homes.cs.washington.edu/~arvind/pcp/.

[6] M. Al-Fares, K. Elmeleegy, B. Reed, and I. Gashinsky. Overclocking the Yahoo!: CDN for faster web page loads. *Proc. ACM SIGCOMM*, 2011.

[7] T. E. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan. PCP: Efficient Endpoint Congestion Control. In *Proc. NSDI*, 2006.

[8] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. *Proc. ACM SIGCOMM*, 1999.

[9] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. ACM SIGCOMM*, pages 267–280. ACM, 2010.

[10] Y. Bhumralkar, J. Lung, and P. Varaiya. Network adaptive TCP slow start. 2000.

[11] J. Brutlag. Speed matters for Google web search, July 2009. http://code.google.com/speed/files/delayexp.pdf.

[12] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and implementation of a routing control platform. *Proc. NSDI*, April 2005.

[13] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie, and M. Gerla. CapStart: An adaptive tcp slow start for high speed networks. *Proc. INTERNET*, 2009.

[14] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. Rethinking Congestion Control Architecure: Performance-oriented Congestion Control. *Proc. ACM SIGCOMM Demo Session*, 2014.

[15] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *ACM Computer Communication Review*, 2010.

[16] A. A. Elbery, E. Khalil, B. M. Nosier, and I. Z. Morsi. Swift Start TCP, Problems, Modification, Analytical and Simulation Studies. *Proc. IEEE SENSORCOMM*, 2008.

[17] W. Feng, M. Fisk, M. Gardner, and E. Weigle. Dynamic right-sizing: An automated, lightweight, and scalable technique for enhancing grid performance. *Proc. High Speed Networks*, 2002.

[18] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing web latency: the virtue of gentle aggression. *Proc. ACM SIGCOMM*, 2013.

[19] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. Diot. Packet-level traffic measurements from the Sprint IP backbone. *Network, IEEE*, 17(6):6–16, 2003.

[20] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 2011.

[21] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. *Proc. ACM SIGCOMM*, August 2009.

[22] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *Selected Areas in Communications, IEEE Journal on*, 21(6):879–894, 2003.

[23] N. Hu and P. Steenkiste. Improving TCP startup performance using active measurements: algorithm and evaluation. *Proc. ICNP*, 2003.

[24] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. *Proc. IMC*, 2012.

[25] D. Liu, M. Allman, S. Jin, and L. Wang. Congestion control without a startup phase. *Proc. PFLDNeT*, 2007.

[26] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Recursively cautious congestion control. *Proc. NSDI*, 2014.

[27] K. Nichols and V. Jacobson. A Modern AQM is just one piece of the solution to bufferbloat. *ACM Queue Networks*, 2012.

[28] V. Padmanabhan and R. Katz. TCP Fast Start: A technique for speeding up web transfers. *Proc. IEEE GLOBECOM*, 1998.

[29] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. *Proc. High Performance Switching and Routing*, 2013.

[30] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. 2009.

[31] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. Tcp fast open. *Proc. CoNEXT*, December 2011.

[32] M. Scharf and H. Strotbek. Performance evaluation of Quick-Start TCP with a Linux kernel implementation. In *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 703–714. Springer, 2008.

[33] A. Singla, B. Chandrasekaran, P. Godfrey, and B. Maggs. The Internet at the Speed of Light. *Proc. HotNets*, 2014.

[34] S. Souders. Velocity and the bottom line, June 2009. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.

[35] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. IMC*, pages 39–44. ACM, 2003.

[36] R. Wang, G. Pau, K. Yamada, M. Sanadidi, and M. Gerla. TCP startup performance in large bandwidth networks. *Proc. IEEE INFOCOM*, 2004.

[37] W. Zhou, Q. Li, M. Caesar, and P. Godfrey. Asap: A low-latency transport layer. *Proc. CoNEXT*, December 2011.