Transport Layer Optimization for Mobile Data Networks

WAN, Wing San

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Information Engineering

The Chinese University of Hong Kong

September 2010

ACKNOWLEDGEMENTS

I would like to take this opportunity to thanks my supervisor Prof. Jack Y. B. Lee who has provided me countless support and advice to my research. In particular, he has taught me the correct way in prioritizing tasks.

I would also like to thank my colleagues including Lok, Stanley, Sam and Rudolf for their help and cheer during the research.

ABSTRACT

The emergence of mobile data networks such as 3G, 3.5G, and 4G is progressively reshaping the Internet landscape from almost entirely wired broadband Internet users, to more and more users connected via wireless links. This thesis investigated the performance limitations of the core Internet transport protocol – Transmission Control Protocol (TCP), when operated in mobile data network environments; and developed a novel network-centric mobile accelerator to overcome such limitations without changing the core TCP transport module at either end of the connection (i.e., server and client). We investigated three fundamental problems of running TCP in modern mobile data networks: (a) flow-control-limited throughput due to larger bandwidth-delay product (BDP); (b) wireless link capacity estimation; and (c) false congestion avoidance due to random loss, and developed novel solutions for implementation in the network-centric accelerator. Experimental results conducted in production 3G HSPA networks show that the accelerator can increase the throughput performance of TCP by up to 2.5 times of the unaccelerated TCP. The proposed accelerator does not require modification to the applications, TCP implementation at the hosts, or operating system; and thus can be readily deployed in current and future mobile data networks.

摘要

流動數據網絡的出現，如 3G、3.5G 和 4G 的，逐步改造了互聯網的形態。從幾乎完全通過有線連接，至越來越多的用戶通過無線連接。本論文探討了核心互聯網傳輸協議 - 傳輸控制協議（TCP）的性能限制，當運行在流動數據網絡環境，並開發一種新的基於網絡的流動加速器，以克服這些限制而不改變連接端（即，服務器和客戶端）的核心TCP傳輸模塊。我們調查了三個在現代流動數據網絡運行 TCP 的基本問題：（一）流量控制於較大的帶寬延遲積（bandwidth-delay product）時對吞吐量的限制，（二）無線網絡可得頻寬的估計; 及（三）因隨機丟包錯誤觸發的擁塞避免機制，並開發新的解決方案以實施以基於網絡的加速器。在實際的 3G HSPA 網絡進行實驗的結果表明，與沒有加速的TCP比較，該加速器可以提高的TCP吞吐量達 2.5倍。建議的加速器並不需要修改應用程序、在主機或操作系統的TCP實踐方式，因此可以容易地部署和應用在當前和未來的流動數據網絡。

CONTENTS

Chapter 1

INTRODUCTION

The emergence of mobile data networks such as 3G, 3.5G, and 4G is progressively reshaping the Internet landscape from almost entirely wired broadband Internet users, to more and more users connected via wireless links. As we will demonstrate in this thesis, this change could have profound impact on the performance of many Internet applications and services given the many fundamental differences between wired and wireless networks. This thesis investigated the performance limitations of the core Internet transport protocol – Transmission Control Protocol (TCP), when operated in mobile data network environments; and then develop a novel network-centric mobile accelerator to overcome such limitations without changing the core TCP transport module at either end of the connection (i.e., server and client).

Specifically, we investigate three fundamental problems of running TCP in modern mobile data networks: (a) flow-control-limited throughput due to larger bandwidth-delay product (BDP); (b) wireless link capacity estimation; and (c) false congestion avoidance due to random loss. For (a) we developed a novel virtual advertised window mechanism to decouple flow control between the client and the server so that the server's transmission throughput is not limited by the receiver advertized window size. For (b) we develop a rate-based algorithm (as oppose to credit-based approach in TCP) to continuously estimate and adapt the data

transmission rate over the mobile data network. For (c) we developed a novel RTT modulation technique to effectively suppress packet loss events from the TCP sender so that high throughput can be sustained despite the existence of random packet loss.

Unlike previous works which rely on modifications to the TCP protocol stack at the sender [4]-[8], at the receiver [10],[11],[12], at both ends [1],[2],[3], or at the network [13],[14], we propose a network-centric approach where the above-mentioned protocol processing algorithms are to be implemented in a network device - mobile accelerator, which is deployed at the edge of the mobile data network linking it to the rest of the Internet.

This network-centric approach has four significant advantages over previous works. First, it eliminates the need to modify the TCP module in either the TCP sender or the TCP receiver. Considering that many modern operating systems are proprietary and slow to adopting any protocol modifications, this network-centric approach allows a far more rapid adoption and deployment of the proposed protocol optimizations. Second, even if we can modify the TCP module, some of the optimizations such as local retransmission (c.f., Chapter 5) simply cannot be effectively implemented in the end hosts. Third, the proposed protocol optimization algorithms are designed specifically for mobile data networks. Thus if it is implemented in the end host such as the server, it will need to determine the type of network a client is accessing before the appropriate protocol optimizations can be applied. Finally, in practice many mobile operators deploy their own web proxies to reduce Internet bandwidth consumption and to implement proprietary value added services. Consequently, the Internet server (e.g., web server) may not even be communicating with the client device directly at all. In this case the TCP connection between the server and the proxy is independent from the TCP connection between the proxy and the client. Hence protocol optimization

performed at the server end will never reach the mobile data network, rendering it useless.

In addition to TCP throughput improvement, the proposed mobile accelerator also opens up a new platform for implementing sophisticated network traffic controls. For example, we developed a packet scheduling mechanism which can perform precise bandwidth allocation among competing TCP flows. This opens the way to many other interesting traffic regulation applications of which some will be discussed in Chapter 9.

The rest of the thesis is organized as follows: Chapter 2 reviews previous related works. Chapter 3 investigates the performance problem of TCP's flow control mechanism in mobile data networks. Chapter 4 presents the virtual advertized window mechanism to overcome TCP's flow control bottleneck. Chapter 5 investigates packet loss in mobile data networks and presents a local retransmission algorithm. Chapter 6 presents the RTT modulation mechanism for loss event suppression. Chapter 7 presents a packet scheduling mechanism for precise bandwidth allocation. Chapter 8 reports performance of the proposed mobile accelerator in production 3G networks. Chapter 9 outlines some future work and Chapter 10 concludes the study.

Chapter 2

BACKGROUND AND RELATED WORK

Much research had been done to improving the performance of TCP in large BDP networks. We can classify the existing works into four categories: modifying both sender and receiver; modifying the sender only; modifying the receiver only; and network-centric approach.

2.1 Sender-receiver-based approaches

We first consider approaches where both the sender and the receiver are modified. Jacobson et al. proposed in RFC 1323[1] the LWS extension to TCP which is currently the most widely supported solution. It works by scaling the Advertised Window (AWnd) by a constant factor throughout the connection. With the maximum LWS factor 14, the maximum AWnd can be increased up to 1 GB ((2^16–1)*2^14 ≈2^30). We will discuss the strengths and weaknesses of LWS in more detail in Chapter 3.

Alternatively, the application can be modified to initiate multiple TCP connections in parallel [2] to increase throughput by aggregating multiple TCP connections. This approach effectively multiplies the AWnd and Congestion Window (CWnd) by the

number of TCP flows and so can mitigate the AWnd limitation. However, aggregating multiple TCP connections will also allow the application to gain unfair amount of bandwidth from competing TCP flows and the aggregate CWnd may increase too rapidly. Hacker et al. [3] solved this problem by deferring CWnd increase until multiple ACKs are received so as to compensate for the rapid CWnd growth.

2.2 Sender-based approaches

Apart from AWnd limit, the CWnd maintained by the sender may also limit TCP's throughput in large BDP networks. Specifically, the growth of the CWnd is triggered by the reception of ACKs. Thus in a long delay path it may take longer time for the CWnd to grow to sufficiently large value so that the link bandwidth can be fully utilized.

To tackle this problem Allman et al. proposed in RFC 3390 [4] to initialize the CWnd to a larger value (as opposed to one TCP segment) so that it can grow more quickly in large delay networks to ramp up TCP's throughput. Since then much effort had been put into developing more sophisticated congestion control algorithms such as CUBIC [5], BIC [6], FAST [7], H-TCP [8] to further shorten TCP's throughput ramp up time and provide better fairness among different TCP connections. These solutions tackled the limitation of CWnd growth and are complementary to our work.

Another problem affecting the CWnd and throughput is the existence of non-congestion-induced packet loss in mobile data networks. As TCP would reduce CWnd upon detection of packet loss, its throughput performance will be severely degraded in the presence of non-congestion-induced packet losses. Lai et al. [9] proposed a TCP-NCL protocol for wireless networks such as WiFi to differentiate

between congestion loss from random packet loss; and to handle packet reordering in case the link layer also retransmit packets. Nevertheless the performance of these sender-based approaches in mobile data networks is still subject for further investigation, and in cases where the mobile network implemented proxy servers then they will have no effect at all as the sender will only be communicating with the proxy server.

2.3 Receiver-based approaches

At the receiving end, Fisk and Feng [10] proposed dynamic right-sizing of the AWnd by estimating the CWnd at the receiver and then dynamically adapt the receiver buffer size, i.e., the AWnd, to twice the size of the estimated CWnd. This ensures that when the sender's CWnd doubles (behavior of TCP New Reno [11] after receiving an ACK) the AWnd will not become the bottleneck.

More recent Operating Systems such as Linux 2.4 and Microsoft Windows Vista [12] also implemented receiver buffer size auto-tuning by estimating the BDP of the connection and the data consumption rate of the application. In comparison, our accelerator does not require any modification to the receiver application or require support from the receiver operating system, and so can be more readily deployed by an ISP or a satellite operator to accelerate all bandwidth demanding TCP traffics.

2.4 Network-centric approach

The fourth approach is to implement protocol optimizations within the network. The Snoop [13] protocol adopts this approach by keeping copies of transmitted packets in the base station's cache, and then retransmit them on behalf of the sender

when packet loss occurs. However the extra delay incurred by the retransmission process could also trigger the sender to timeout which will lead to severe throughput degradation. We tackled this problem through a novel RTT modulation mechanism (c.f. Chapter 6) which can actively prevent sender timeout in case of local retransmission.

Another study by Hu and Yeung [14] proposed a new active queue management protocol whereby the network device such as a router or a base station, generates duplicate ACKs to trigger CWnd reduction at the sender whenever the device's packet queue length exceeds a certain threshold, thus preventing congestion from taking place. While this work share the same network centric approach it is solving a different problem compared to our study.

Chapter 3 TCP FLOW CONTROL REVISITED

TCP's built-in flow control mechanism is designed to prevent fast sender from overflowing slow receiver. It works by reporting the receiver's buffer availability, i.e., the AWnd, back to the sender via a 16-bit field inside the TCP header so that the sender would not send more data than the receiver's buffer can store.

Over the years computer processing power have grown tremendously such that even today's modest computers can easily keep up with the arriving stream of data at relatively high data rates (e.g., tens of Mbps). Thus an arrived packet will quickly be retrieved by the application from the receiver buffer, and in most cases this can be completed even before the next packet arrives. As a result, the reported AWnd simply stays at the maximum receiver buffer size as illustrated in Fig. 1 which plots the actual AWnd of a receiver at a throughput of 1.5 Mbps. In this case TCP's flow control mechanism is clearly not necessary as the sender never transmit data faster than the receiver's processing rate.

Due to the delayed AWnd the sender cannot send more than the reported AWnd and thus cannot make use of the new freed buffer space at the receiver. In cases where the BDP is larger than the maximum AWnd, the sender will operate in a stop-and-go manner as illustrated in Fig. 2, resulting in severe underutilization of the network channel.

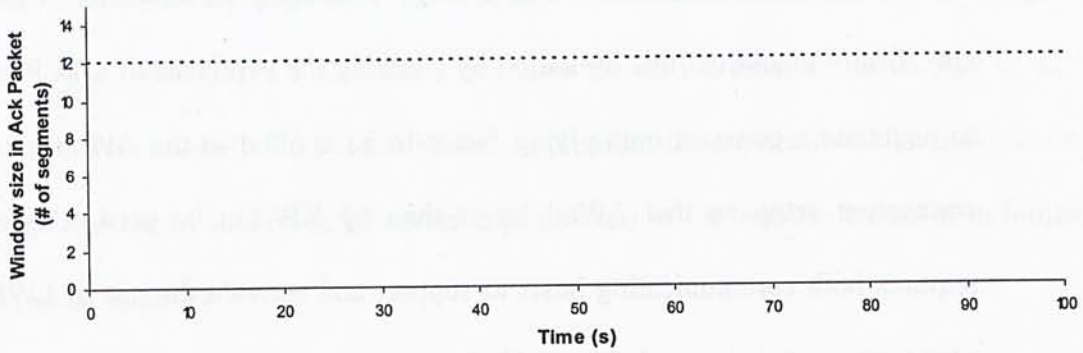

Fig. 1. Dynamics of reported window size

For example, a 3G HSPA data network has bandwidth ranging from 3.6 Mbps to 14.4 Mbps. Take the commonly offered bandwidth 7.2 Mbps with round-trip-delay (RTT) of 100 ms as an example. This set of network parameters will lead to a BDP of 90 KB which already exceeds TCP's maximum advertised window size of 64 KB. In this case TCP's flow control mechanism will limit the throughput to no more than 5.1 Mbps which leave about 29% of bandwidth unused even if there are no competing traffics in the network.

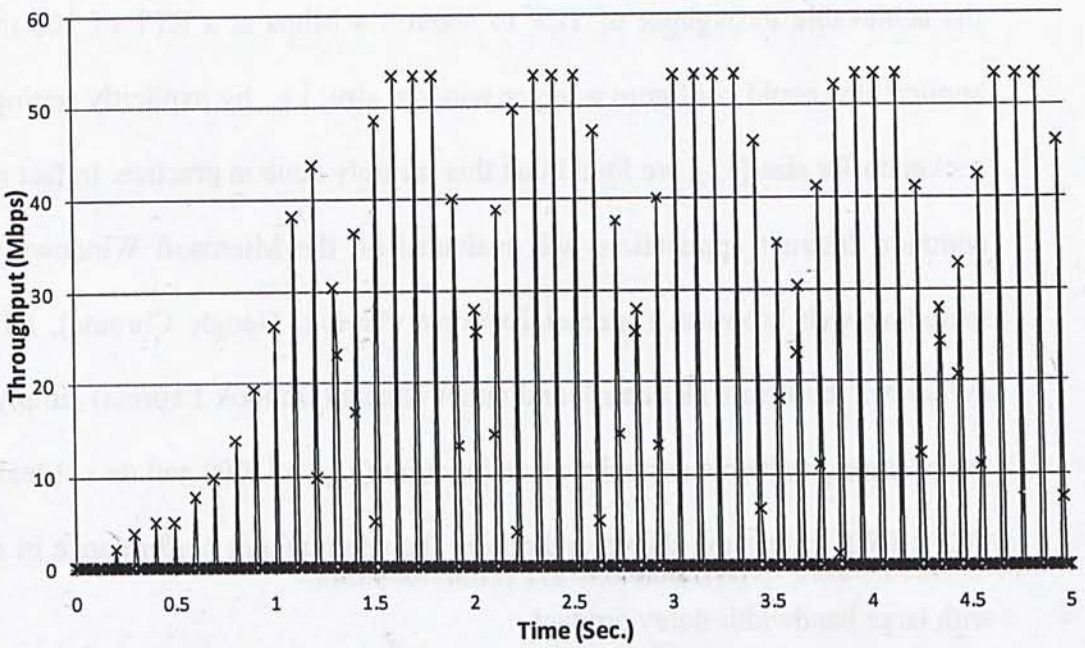

Fig. 2. Stop-and-go behavior due to large BDP compared with Awnd

The conventional solution – TCP's Large Window Scale extension, was designed specifically to address this limitation by allowing the two hosts of a TCP connection to negotiate a constant multiplying factor to be applied to the AWnd value during connection setup so that AWnd larger than 64 KB can be used. Obviously this requires both communicating hosts to support and activate the use of LWS prior to connection setup is carried out. Unfortunately although most modern operating systems support LWS, there is no standard way for the application to activate the use of LWS. Some operating systems activate LWS by default (e.g., Windows Vista), some requires manual configuration (e.g., Windows XP), and yet others require the application to explicitly configure a large (e.g., >64 KB) socket buffer to activate it [12].

For example, the common operating system Microsoft Windows XP defaults its advertised window size to 17 KB for connections with link capacity at or below 10 Mbps. Running over a 3G HSPA data network this advertised window size will limit the achievable throughput of TCP to about 1.4 Mbps at a RTT of 100 ms. While applications could configure a larger window size, i.e., by explicitly setting a larger socket buffer size [12], we found that this is rarely done in practice. In fact almost all common Internet applications we evaluated on the Microsoft Windows platform, including web browsers (Internet Explorer, Firefox, Google Chrome), FTP clients (Windows' built-in FTP client), and email clients (Outlook Express) simply employ the operating system's advertized window size (e.g., 17 KB) and do not make use of TCP's LWS extension, therefore limiting their throughput performance in networks with large bandwidth-delay product.

Another limitation of LWS is that the AWnd value is still interpreted as the amount of buffer physically available at the receiver (and sender as well), thus for

networks with very large BDP the resultant buffer requirement can be very large as well (e.g., with LTE's 172.8 Mbps over 100 ms RTT a receiver buffer size of 21.6 MB will be needed). This could become a problem for applications that make use of large number of sockets (e.g., server applications) or for mobile devices with limited physical memory.

If we reconsider Fig. 1 then we can expect that such a large buffer at the receiver will end up mostly unused anyway as the received data will be retrieved out of the TCP receive buffer quickly by the application. Thus instead of strictly following the reported receiver buffer size, we propose to reinterpret it as an indicator of receiver processing capacity and employ it for use in a rate-based flow control mechanism to be discussed in the next Chapter.

Chapter 4 OPPORTUNISTIC TRANSMISSION

Results from Chapter 3 reveal that the AWnd turns out to become the bottleneck to achieving high throughput in modern mobile data networks. A trivial solution would be to eliminate the AWnd altogether as it is no longer serving its intended function. To experiment with this idea we implemented a simple accelerator between the sender and the receiver as depicted in Fig. 3. The accelerator's sole function is to modify the AWnd field inside the TCP header of ACK packets from the receiver, to a value much larger than the BDP of the network to prevent it from becoming the bottleneck. In this experiment we make use of the Linux Netem module [15] to emulate a typical 3G HSPA data network. The emulation parameters including network delay and bandwidth are obtained from measurement of a production 3G HSPA data network.

The sender, the accelerator, and the emulator all run on Linux with kernel 2.6 (with CUBIC as the default TCP congestion control module) and the receiver is running on Windows XP SP2. This network configuration has a BDP of 90 KB which is larger than Windows XP's default receiver buffer size of 17 KB. The receiver initiates the connection and then the sender would keep sending data to the receiver as fast as TCP allows.

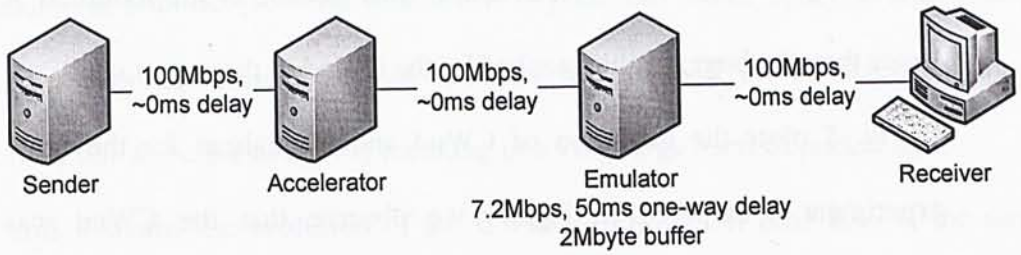

Fig. 3. Emulation platform

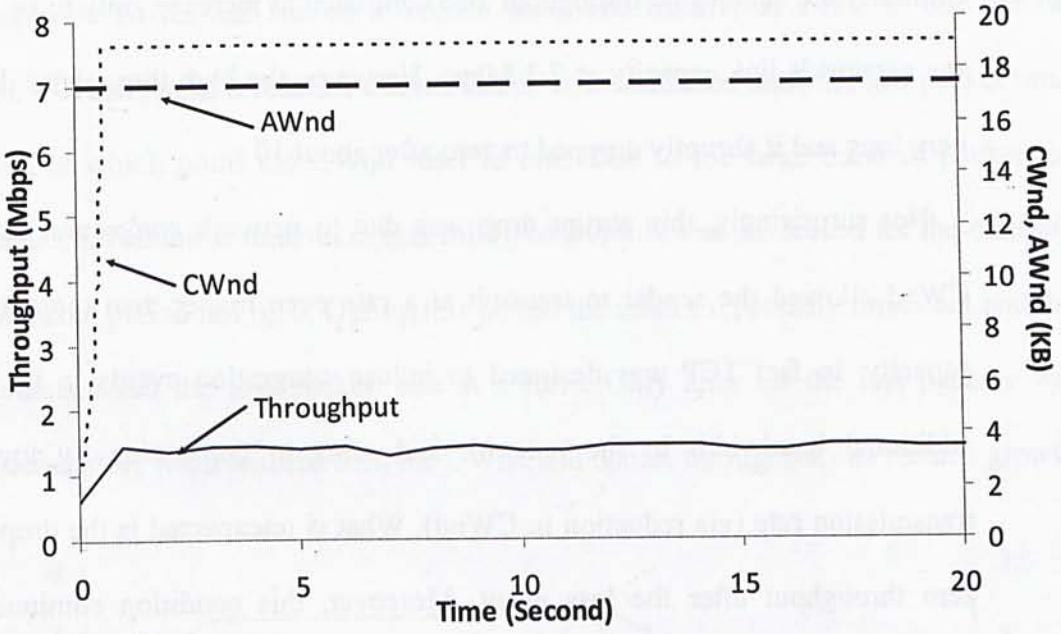

Fig. 4. CWnd and Throughput dynamics without accelerator

As a benchmark for comparison we first conducted an experiment without any modification to the AWnd reported by the receiver. Fig. 4 shows the evolution of AWnd, CWnd, and throughput (as measured by the receiver) of the first 20 seconds of the experiment. As expected the AWnd stayed at the maximum value at all times. The CWnd, once increased pass the AWnd value, stopped increasing further even though there is no packet loss during the experiment. As a result the achievable throughput is limited by the window size and in this case is significantly lower than the network's capacity (i.e., 1.41 Mbps versus 7.2 Mbps).

In the second experiment the accelerator will modify the AWnd reported by the receiver to 10 MB, regardless of the actual value of the AWnd reported. This

effectively disables TCP's flow control mechanism as the modified AWnd size is larger than the largest value reached by the CWnd in the experiment.

Fig. 5 plots the evolution of CWnd and throughput for the first 15 s of the experiment. Compared to Fig. 4 we observe that the CWnd was no longer constrained by the AWnd and thus could continue to grow all the way up to 3 MB. Similarly the achievable throughput also continued to increase, only to be limited by the network's link capacity at 7.2 Mbps. However, the high throughput did not last very long and it abruptly dropped to zero after about 10 s.

Not surprisingly, this abrupt drop was due to network congestion as the large CWnd allowed the sender to transmit at a rate even higher than the network link capacity. In fact TCP was designed to induce congestion events in its probe for additional bandwidth in the network and react to congestion by lowering its transmission rate (via reduction in CWnd). What is unexpected is the drop to nearly zero throughput after the loss event. Moreover, this condition continued for an extensive duration (over 600 s) as depicted in Fig. 6 before the TCP flow could recover from it. This suggests that TCP congestion control algorithm could not function well without its flow control mechanism.

To understand the cause of the blackout period after a loss event, we need to consider the behavior of the receiver. Normally when there is no packet loss, TCP segments arriving at the receiver are quickly processed and passed to the application, thus releasing the occupied buffer space. However when a packet loss occurs, all the subsequent data will need to be buffered until the lost packet is successfully retransmitted as TCP guarantees in-sequence data delivery. In normal TCP the sender will stop transmitting data once the AWnd is used up and thus buffer overflow will never occur. Now in case the AWnd is set to a value larger than the actual one, the

sender will continue to send data, even beyond the buffer space available at the receiver. Consequently, these out-of-bound packets will all be discarded by the receiver due to buffer overflow, resulting in a very large burst of packet loss.

This explains the evolution of the CWnd in Fig. 5. At time 8.49 s, the sender (running the default CUBIC [5] congestion module in Linux) began receiving duplicate ACKs and started to reduce its CWnd linearly at a rate of one per two DUP-ACKs, until it reached a lower limit. This continued until the lost packet timed out, at which point the CWnd reset to one. Due to the large burst of packet loss induced, numerous timeout events followed and this was the reason for the extensive blackout period in Fig. 6. During this period the sender repeatedly timed out and then retransmitted the lost packet, one at a time. Only after all the lost packets were successfully retransmitted then the CWnd and hence, throughput can resume growth.

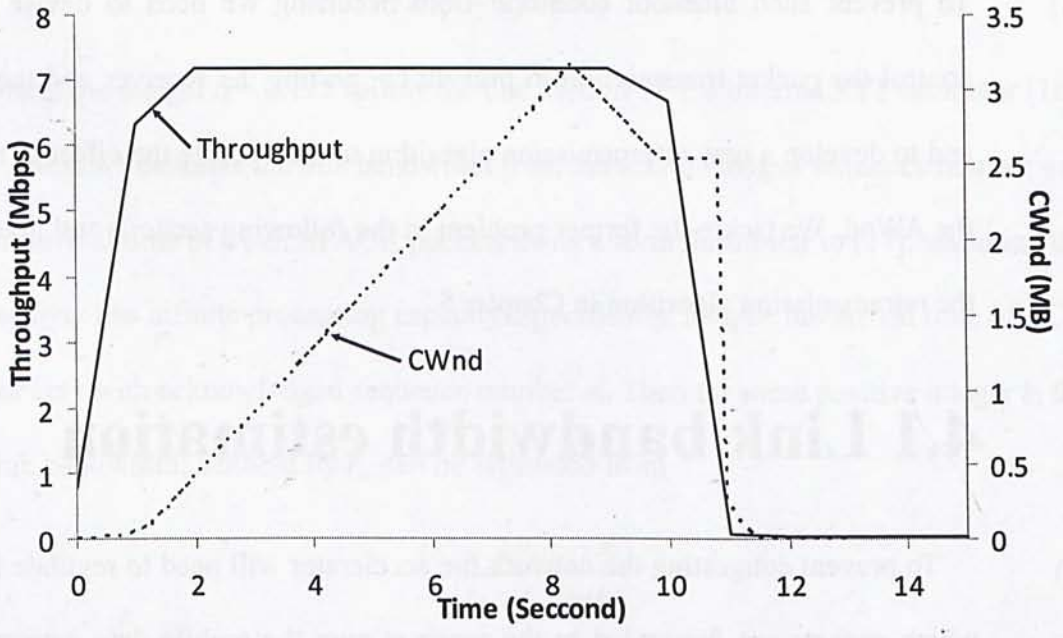

Fig. 5. CWnd and Throughput dynamics of first 15 s with sender's flow control mechanism disabled

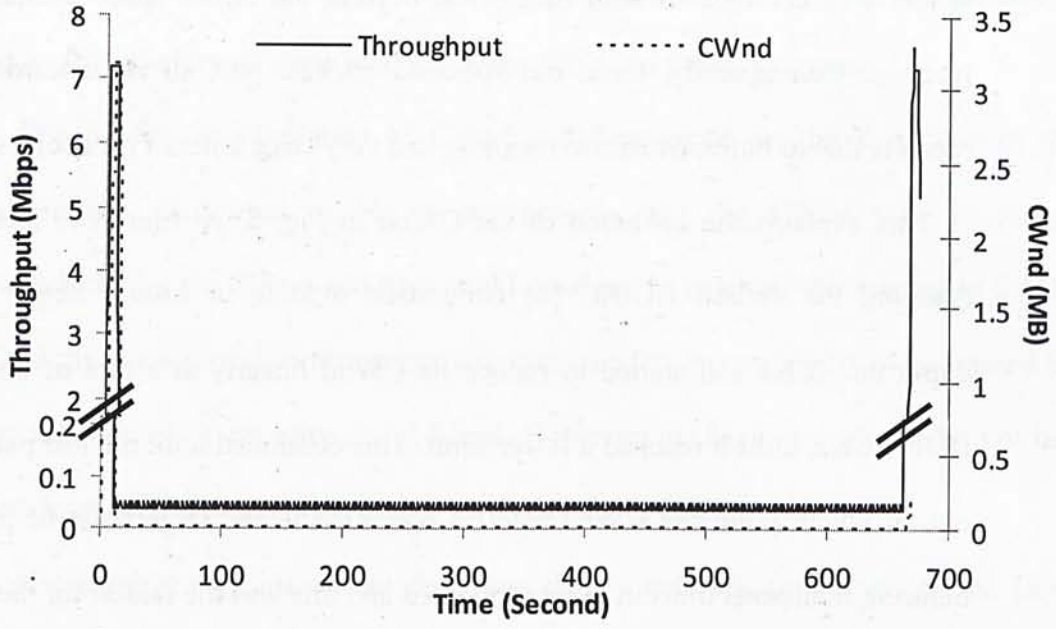

Fig. 6. CWnd and Throughput dynamics with sender's flow control mechanism disabled

The previous experiment clearly shows that the trivial solution of disabling TCP's flow control will end up degrading throughput performance rather than improving it. To prevent such blackout condition from occurring we need to devise a way to control the packet transmission to prevent congesting the receiver and the network, and to develop a new retransmission algorithm to incorporate the effect of modifying the AWnd. We tackle the former problem in the following sections and investigate to the retransmission algorithm in Chapter 5.

4.1 Link bandwidth estimation

To prevent congesting the network the accelerator will need to regulate the rate at which packets are forwarded to the receiver over the mobile data network. In the following we develop a system model to estimate the link bandwidth by assuming that (a) the accelerator always has data to forward; (b) network delays and receiver's processing capability remain constant; (c) network delays of the channel between the accelerator and the receiver are symmetric in the forward and the reverse direction;

(d) the receiver generates an ACK immediately upon the arrival of a TCP segment, i.e., zero processing delay; and (e) the uplink, i.e., from the receiver to the accelerator, is not the bottleneck.

Let *rtt* be the RTT between the accelerator and the receiver. Obviously the RTT is not known *a priori* and thus will need to be estimated from passive measurements of packets traversing the accelerator. Let f_i be the time packet i was forwarded by the accelerator to the receiver, and let t_i be the time at which the corresponding ACK arrived at the accelerator. Then the RTT as measured by packet i, denoted by rtt_i, can be computed from:

$$rtt_i = t_i - f_i \tag{1}$$

To smooth out random fluctuations in the rtt_i, the accelerator applies exponentially weighted moving average to the measured values to obtain the smoothed RTT *rtt*:

$$rtt = (1-\alpha)\times rtt + \alpha \times rtt_i \tag{2}$$

where the weight $\alpha = 0.125$ follow the one used in TCP's internal RTT estimator [16].

Next we estimate the link bandwidth from the acknowledged sequence number and the arrival time of a pair of ACK packets using a formula similar to [17], assuming the receiver has infinite processing capacity. Specifically, let t_i be the arrival time of ACK packet i with acknowledged sequence number n_i. Then for some positive integer k, the link bandwidth, denoted by r_i, can be estimated from

$$r_i = \frac{(n_{i+k} - n_i)}{t_{i+k} - t_i} \tag{3}$$

where the numerator is the amount of data acknowledged during the time interval $(t_i, t_{i+k}]$. The intuition behind (3) is that the receiver, having infinite processing capacity, will generate an ACK packet immediately upon receiving a packet from the network.

Thus the transmission rate is faster than the link bandwidth then the rate at which packets departs from the network will be determined by the link bandwidth available.

The parameter k controls the duration of the estimation interval (in number of ACK packets) and can be adjusted to tradeoff between accuracy and timeliness of rate estimation. We also apply exponentially weighted moving averaging similar to (2) to smooth out random fluctuations and obtain the smoothed estimated link bandwidth denoted by r.

The assumption of infinite receiver processing capacity is obviously not valid in practice so we relax this assumption in the next section.

4.2 Reception rate estimation

When packets arrive at a rate higher than the receiver's processing capacity, the received data will be buffered at the receiver buffer awaiting processing. The lower buffer availability then will result in smaller reported AWnd. Thus by monitoring the AWnd reported from the receiver the accelerator can incorporate the receiver's processing capability into the system model described by (3).

Specifically, let a_i be the value of AWnd reported by ACK packet i. Then the amount of data processed by the receiver between time t_i and t_{i+k} is given by:

$$(a_{i+k}+n_{i+k})-(a_i+n_i) \tag{4}$$

which is the difference in the acknowledged sequence number plus the difference in the reported AWnd. In case the receiver is slower than the incoming data rate then a_{i+k} will decrease, thus reducing the amount.

The reception rate, denoted by R_i, can then be computed from

$$R_i = \frac{(a_{i+k}-a_i)+(n_{i+k}-n_i)}{t_{i+k}-t_i} \tag{5}$$

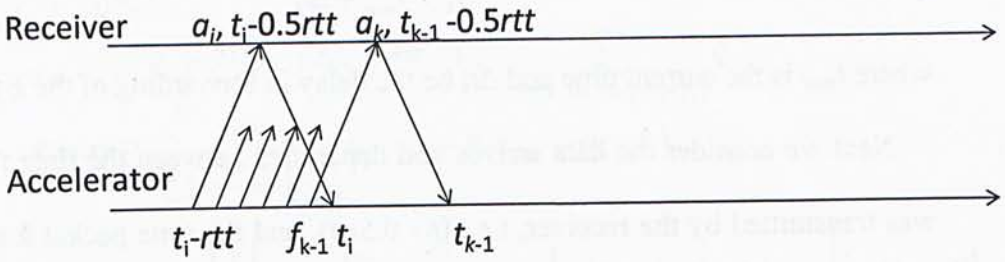

Fig. 7. Transmission timing diagram between receiver and accelerator

Similar to (3) the parameter k controls the width of the estimation interval and we also apply exponentially weighted moving averaging to (5) to smooth out the random fluctuations and obtain the smoothed reception rate denoted by R. Note that if the receiver processing capacity is infinite, then $a_{i+k}=a_i$ for all i and k, and (5) will reduce to the special case in (3). The estimated reception rate incorporated the effect of both link bandwidth availability and receiver processing capacity. This will be used to schedule the forwarding of packets in the accelerator.

4.3 Transmission scheduling

Armed with an estimate of the reception rate, our goal is to schedule the transmission of packets from the accelerator to the receiver such that it will not cause buffer overflow at the receiver. The challenge is that the AWnd reported in a ACK packet is delayed information - it was the receiver's buffer availability 0.5*rtt* s ago. During the time the ACK took to travel to the accelerator additional TCP segments may have arrived at the receiver, and the receiver application may also have taken out more data from the receive buffer.

Consider the timing diagram in Fig. 7, we want to find the minimum f_k for the k-th packet such that the expected buffer availability b_k when the k-th packet arrives at the receiver is non-negative (i.e., no buffer overflow). f_k may further be expressed as

$$f_k = t_{now} + \Delta t_k \tag{6}$$

where t_{now} is the current time and Δt_k be the delay in forwarding of the k-th packet.

Next we consider the data arrives and departures between the time the last ACK was transmitted by the receiver, i.e., (t_i - 0.5rtt), and the time packet k arrives at the receiver, i.e., (f_k + 0.5rtt). First, during this time the receiver will continue to process packets and free-up buffers at a rate of R. Thus the expected amount of buffer space being freed up in this period, denoted by D_{proc}, is given by

$$\begin{aligned} D_{proc} &= R\left(f_k + 0.5rtt - t_i + 0.5rtt\right) \\ &= R\left(t_{now} + \Delta t_k - t_i + rtt\right) \end{aligned} \tag{7}$$

Similarly, the expected amount of data arriving at the receiver during this period, denoted by D_{recv}, is given by

$$D_{recv} = \sum_{\forall j \left| \{ f_j > (t_i - rtt) \cap f_j \le f_k \} \right.} q_j \tag{8}$$

Hence the buffer availability b_k at time (f_k + 0.5rtt) can be computed from the buffer availability at the beginning of the period, i.e., a_i, plus the amount of buffer freed up, i.e., D_{proc}, minus the amount of data arrivals, i.e., D_{recv}:

$$\begin{aligned} b_k &= \min\left\{a_i + D_{proc} - D_{recv}, a_{\max}\right\} \\ &= \min\left\{a_i + R\left(t_{now} + \Delta t_k + 0.5rtt - t_i + 0.5rtt\right) - \sum_{\forall j \left| \{ f_j > (t_i - rtt) \cap f_j \le f_k \} \right.} q_j, a_{\max}\right\} \\ &= \min\left\{a_i + R\left(t_{now} + \Delta t_k + rtt - t_i\right) - \sum_{\forall j \left| \{ f_j > (t_i - rtt) \cap f_j \le f_k \} \right.} q_j, a_{\max}\right\} \end{aligned} \tag{9}$$

where a_{max} is the maximum AWnd of the receiver. Assuming $b_k < a_{max}$, then the packet forwarding delay can be computed from

$$0 \le b_k$$

$$0 \le a_i + R\left(t_{now} + \Delta t_k + rtt - t_i\right) - \sum_{\forall j \mid \{f_j > (t_i - rtt) \cap f_j \le f_k\}} q_j$$

$$R \times \Delta t_k \ge \sum_{\forall j \mid \{f_j > (t_i - rtt) \cap f_j \le f_k\}} q_j - a_i - R\left(t_{now} + rtt - t_i\right) \tag{10}$$

$$\Delta t_k \ge \max\left(\frac{1}{R}\left(\sum_{\forall j \mid \{f_j > (t_i - rtt) \cap f_j \le f_k\}} q_j - a_i\right) - \left(t_{now} + rtt - t_i\right),\ 0\right) \left(\because \Delta t_k \ge 0\right)$$

The accelerator computes the packet forwarding time for each head-of-line packet in a TCP flow, and transmits them according to a round-robin scheduler in Chapter 7.

4.4 Performance

To evaluate the performance of the developed estimation and scheduling algorithms we repeated the experiment depicted in Fig. 3. The accelerator is equipped with the reception rate estimation algorithm and will schedule packet forwarding according to the transmission scheduler described in Section 4.3.

Fig. 8 shows the evolution of the AWnd, CWnd, Virtual AWnd, buffer occupancy at accelerator, sending rate and the throughput achieved by the TCP flow. Unlike the previous experimental results in Fig. 5, the bottleneck link was not congested even though the sender's CWnd grew to very large value (~10 MB). As a result the TCP throughput could reach and sustain at the link's capacity limit. Note that the initial sending rate is higher than the TCP throughput as the accelerator reports its own buffer availability in the AWnd sent back to the sender. Once the accelerator buffer becomes full, i.e., around 25 seconds, the AWnd will realize flow control between the sender and the accelerator, of which the sender quickly reduces its sending rate to the TCP throughput. Compared to the case without the accelerator, the achievable throughput (payload only) is increased from 1.34 Mbps to 6.71 Mbps.

Nevertheless, unlike wired networks where packet losses are primarily due to congestion, our measurements show that mobile data networks do exhibit random packet losses which are not related to congestion. Thus although the transmission scheduler can prevent congestion-induced packet losses, the occurrences of random packet loss may still degrade TCP's throughput performance. We investigate this loss recovery problem in the next chapter.

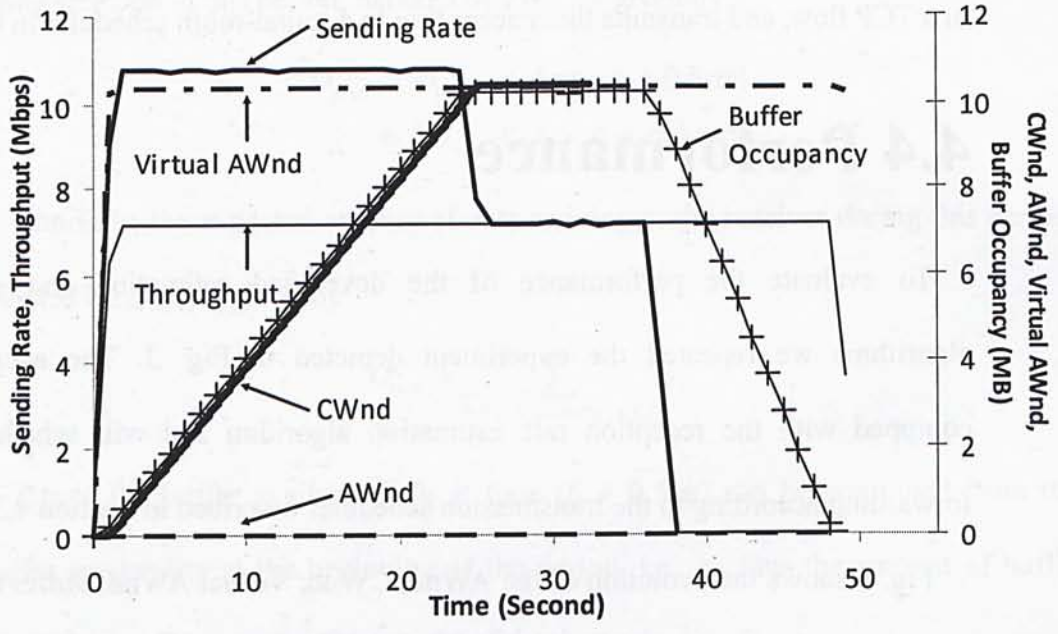

Fig. 8. Evolution of the AWnd, CWnd, Virtual AWnd, buffer occupancy, sending rate and throughput achieved with transmission scheduling

Chapter 5 Local Retransmission

As shown in Chapter 4 the introduction of a virtual AWnd, which is purposely made larger than the actual receiver AWnd, can lead to large packet loss bursts which could blackout the throughput performance for hundreds of seconds (c.f. Fig. 9). The transmission scheduler in Chapter can prevent loss induced by congestion but our experiments revealed that even a very low level of residual packet loss could still cause the blackout phenomena. We develop in this Chapter a new local retransmission algorithm to tackle this burst-loss problem so that the throughput of the TCP flow can be maintained in the event of packet loss.

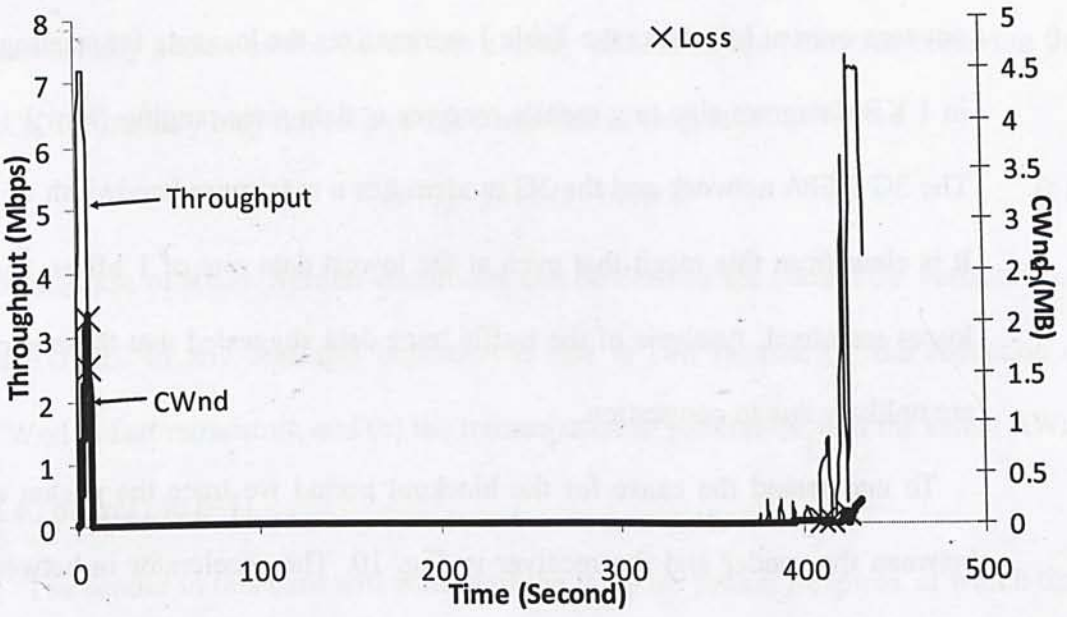

Fig. 9. Dynamic of CWnd and throughput in the present of packet loss

5.1 The blackout period

To illustrate the problem we conducted an experiment using the same network topology as in Fig. 3. The accelerator implements the transmission scheduler to prevent network congestion from occurring. We configure the network emulator to introduce random loss at 0.01%. For ordinary TCP packet loss at this low loss rate will not cause significant throughput degradation as such losses can be readily recovered via TCP's fast retransmit algorithm [18].

By contrast, as shown in Fig. 9 which plots the evolution of the CWnd versus time, the accelerated TCP flow did suffer from the same blackout period as reported in Chapter 4, even if there are only two packet losses before the blackout period began. The blackout period continues for over 400 s while the sender repeatedly timeout and retransmit lost packets, one at a time, to the receiver.

Our measurements of production 3G HSPA networks revealed that the loss rate is not zero even at low data rate. Table 1 summarizes the loss rate for sending UDP data in 1 KB datagram size to a mobile receiver at data rates ranging from 1 to 7 Mbps. The 3G HSPA network and the 3G modem has a maximum bandwidth of 7.2 Mbps. It is clear from this result that even at the lowest data rate of 1 Mbps, some packet losses remained. Analysis of the traffic trace data suggested that these packet losses are unlikely due to congestion.

To understand the cause for the blackout period we trace the packet exchanges between the sender and the receiver in Fig. 10. The accelerator in-between simply forward packets between the sender and receiver, except for rewriting the AWnd field in ACKs sent by the receiver to a large value (e.g., 10 MB). At the beginning of Phase I the receiver buffer is empty as we assume it takes zero time to process the

incoming packet and passes the data to the receiver application. Now suppose packet i is lost. Then while packets i+1, i+2, etc., arrive at the receiver successfully, they cannot be passed to the application as packet i has not been received. Instead the subsequent packets must be buffered up until the buffer is full when packet j (j>i) arrives as depicted in Phase II of Fig. 10. In this case the receiver will discard packet j and all subsequent packets as well.

Eventually the sender will use up the CWnd, say, after packet k–1 (assuming k>j+1) is transmitted and then it has to halt transmission. Meanwhile when three duplicate ACKs for packet i is received the sender will trigger fast retransmit by retransmitting packet i and decreasing the CWnd (e.g., to half in Reno). Suppose the retransmission is successful then the ACK for packet j (i.e., the last buffered packet) will return to the sender at the end of Phase II. At the receiver side all the buffered data can now be passed to the application, thus freeing up the receiver buffer.

The blackout period begins in Phase III. Here the sender may not be able to transmit any packet as the CWnd has already been used up. Note that receiving the ACK for packet j may not relieve this condition as long as

$$(k-1)-j+1 \geq (k-i+1)/2 \tag{11}$$

for the case of Reno. Similar conditions can be derived for other TCP variants such as CUBIC. In any case this condition is due to two factors: (a) the reduction of CWnd in fast retransmit; and (b) the transmission of packets beyond the actual AWnd (i.e., packet j to k–1).

The sender in this case will wait until the RTO for packet j expires, at which time it retransmits packet j and reset the CWnd to 1. By the time the ACK returns the RTO for packet j+1 will likely have expired as well and so the sender will retransmit packet j+1, and the process repeats. During Phase III the sender effectively can

transmit just one packet per RTT, resulting in the long stretch of very low throughput observed in Fig. 9. Eventually all the lost packets up to packet k–1 will be successfully retransmitted and then the sender will begin slow-start again in Phase IV of Fig. 10. Note that this black-out behavior occur because the accelerator can send packets beyond the receiver's AWnD. We verified via experiments that the same behavior does not occur in ordinary TCP, even when a large burst of consecutive packets are dropped.

Therefore while rewriting the AWnd eliminates the throughput limit constrained by AWnd, it introduces a new problem in packet loss recovery. We tackle this problem by developing a new predictive retransmission algorithm in the next section.

Sending rate (Mbps)	1	2	3	4	5	6	7
Loss ratio	0.01	0.01	0.01	0.04	0.21	0.33	0.47

Table 1. Average loss ratio at various sending rate

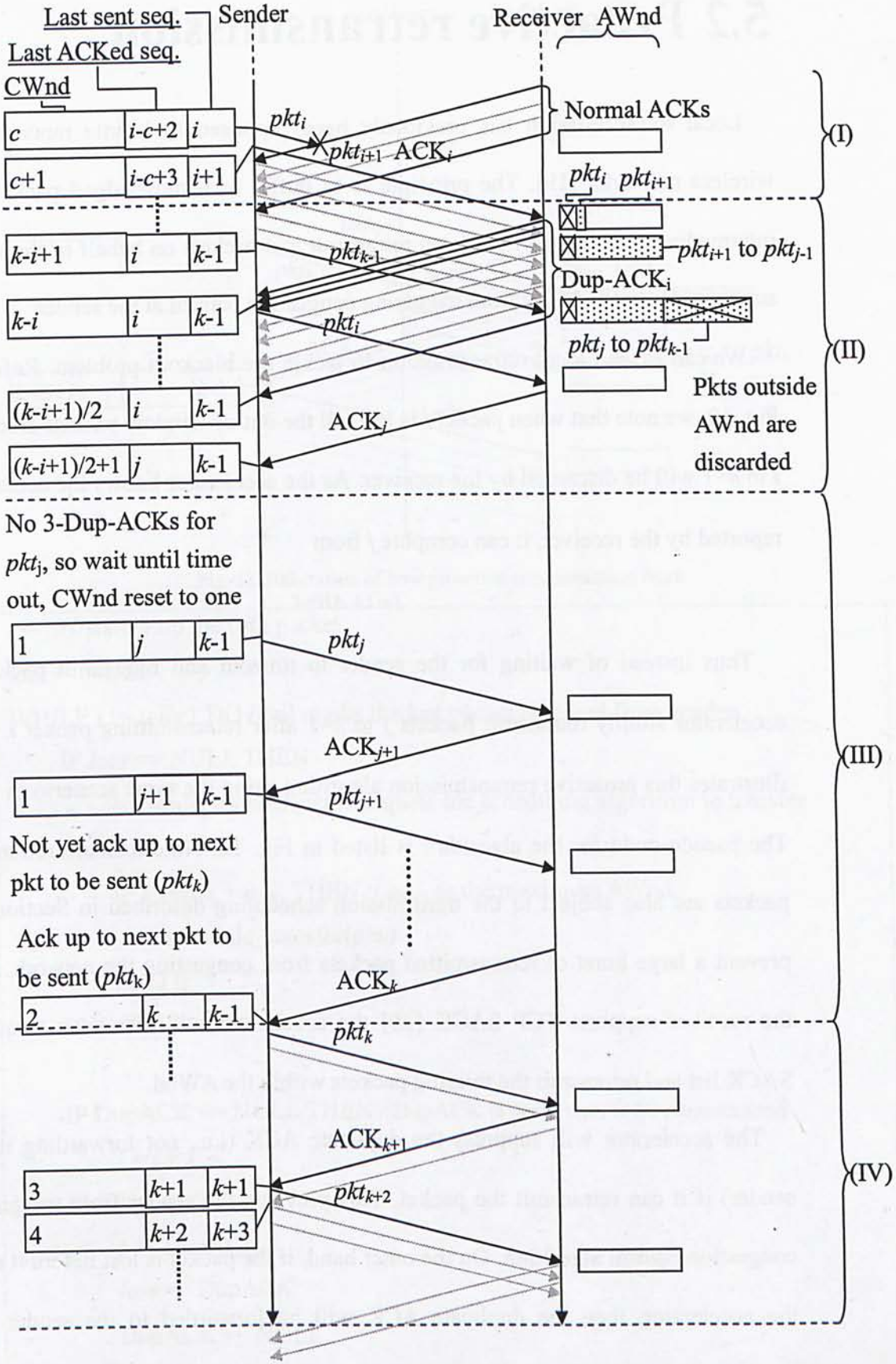

Fig. 10. Illustration of slow recovery of discarded packets caused by disabling flow control

5.2 Proactive retransmission

Local retransmission has previously been proposed to handle random loss in wireless networks [19]. The principle is to buffer unacknowledged packets in the intermediate gateway so that it can retransmit lost packets on behalf of the sender to suppress duplicate ACKs from triggering congestion control at the sender.

We can extend local retransmission to tackle the blackout problem. Referring to Fig. 10, we note that when packet i is lost, all the out-of-window packets from packet j to k–1 will be discarded by the receiver. As the accelerator knows the actual AWnd reported by the receiver, it can compute j from

$$j = i + AWnd \tag{12}$$

Thus instead of waiting for the sender to timeout and retransmit packet j the accelerator simply retransmit packets j to k–1 after retransmitting packet i. Fig. 11 illustrates this proactive retransmission algorithm using the same scenario in Fig. 10. The pseudo-code for the algorithm is listed in Fig. 12. Note that the retransmitted packets are also subject to the transmission scheduling described in Section 4.3 to prevent a large burst of retransmitted packets from congesting the network. In case the receiver supports TCP SACK [20] the accelerator will also scan through the SACK list and retransmit the missing packets within the AWnd.

The accelerator will suppress the duplicate ACK (i.e., not forwarding it to the sender) if it can retransmit the packet. This prevents the sender from triggering its congestion control algorithm. On the other hand, if the packet is lost before it reaches the accelerator, then the duplicate ACK will be forwarded to the sender as the accelerator cannot retransmit a packet never received. This also preserves TCP's congestion control of the path between the sender and the accelerator.

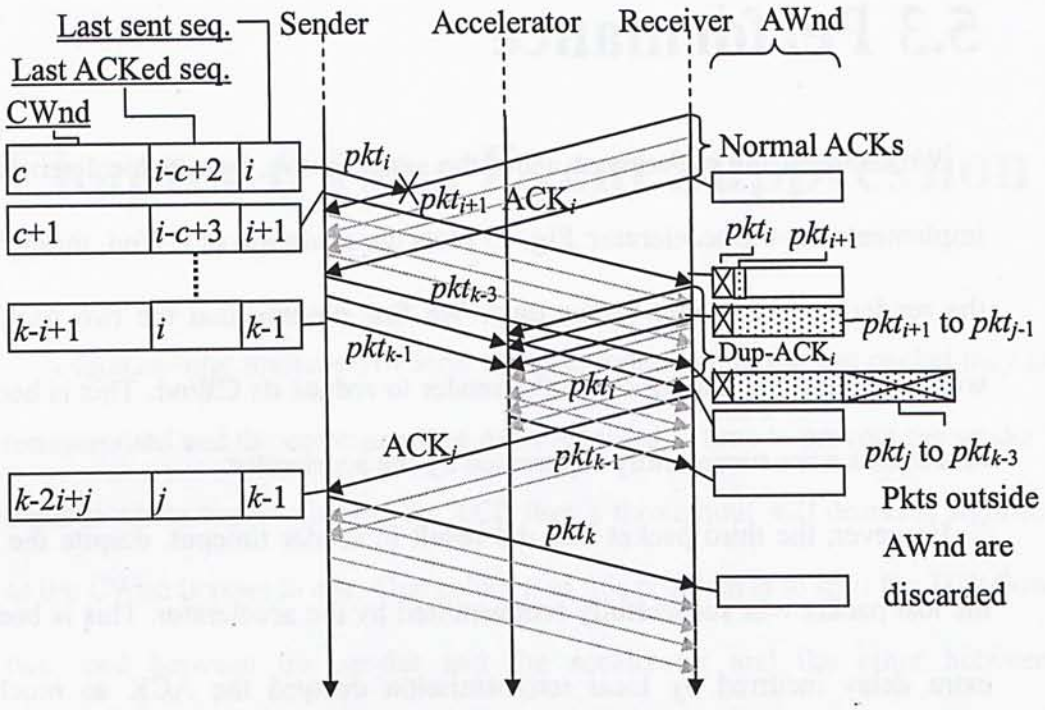

Fig. 11. Illustration of how proactive retransmission work

```
i ← 0 //start from the 0-th packet
loss ← NULL
WHILE i != tail+1 DO //tail marks the last packet received from sender
    IF loss == NULL THEN
        schedule_transfer(pkt_i) //request the scheduling algorithm to transfer
    ELSE
        IF i ≥ loss + a_max THEN // a_max as the maximum AWnd
            schedule_transfer(pkt_i)
        END IF
    END IF

    IF DupACK == NULL THEN //DupACK is packet no. to be retransmitted
        i = i + 1
    ELSE
        i ← DupACK
        loss ← DupACK
        DupACK ← NULL
    END IF
END DO
```

Fig. 12. Pseudo code for data transmission with the local retransmission capability

5.3 Performance

We repeated the experiment using the setup in Fig. 3 with local retransmission implemented at the accelerator. Fig. 13 plots the evolution of CWnd, throughput, and the sender's sending rate versus time. We first observe that the two packet losses within the first 5 s did not cause the sender to reduce its CWnd. This is because the loss events were successfully suppressed by the accelerator.

However, the third packet loss did result in sender timeout, despite the fact that the lost packet was successfully retransmitted by the accelerator. This is because the extra delay incurred by local retransmission delayed the ACK so much that it exceeded the RTO of the sender. This shows that local retransmission alone may not be able to suppress all loss events. We develop a novel RTT modulation technique in the next section to tackle this problem.

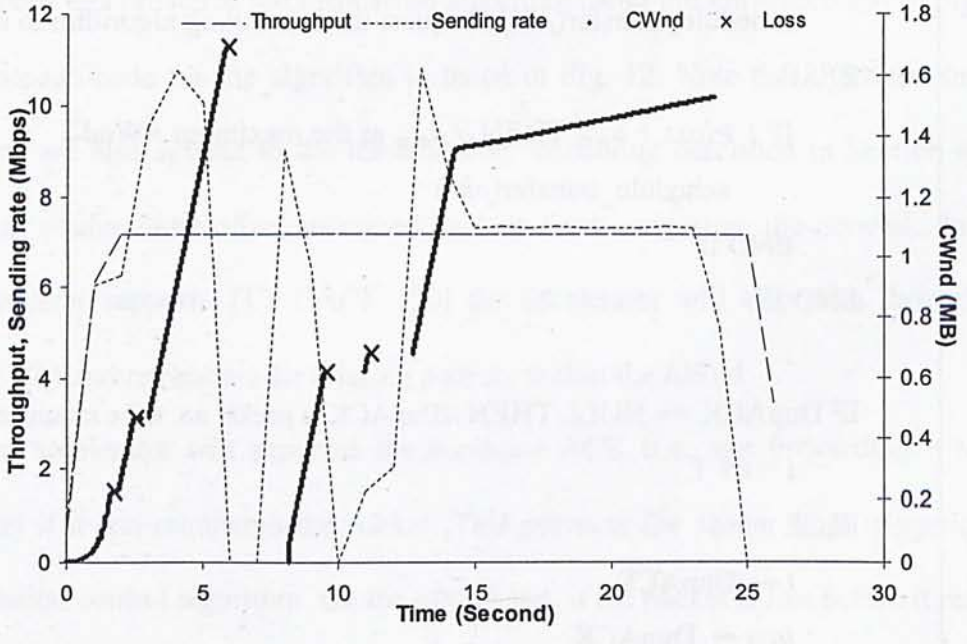

Fig. 13. Evolution of CWnd, throughput and sending rate with proactive retransmission

Chapter 6 Loss Event Suppression

A fundamental limitation of local retransmission is that the lost packet may not be retransmitted and the corresponding ACK received in time to prevent the sender from timeout. Once timeout occurs the TCP flow's throughput will decrease significantly as the CWnd is reset to one. One solution to this problem is to split the TCP flow into two, one between the sender and the accelerator and the other between the accelerator and the receiver [21]. This split-TCP approach decouples loss recovery between the sender and the receiver by generating ACKs independently of the receiver. Specifically, a split-TCP gateway will return ACKs to the sender as if it is the receiver. The received packets are then forwarded to the receiver in a separate TCP connection. Thus loss events between the gateway and the receiver are handled completely by the gateway, without involving the sender at all. This prevents the sender from triggering its congestion control algorithm.

However the split-TCP approach suffers from a tradeoff – it breaks the reliable service guarantee provided by TCP. In particular, it is possible for a packet to be acknowledged by the gateway but ultimately failed to deliver to the receiver (e.g., the network link goes down before the gateway can forward the packet to the receiver). In this case the sender and the receiver will be in an inconsistent state, i.e., the sender assumes successful delivery of a packet which is in fact not received by the receiver.

To avoid this problem we propose a novel RTT modulation mechanism to enable the suppression of loss events without the need for splitting the TCP flow.

6.1 RTT modulation

The challenge in retransmitting lost packet locally by the accelerator is the time needed for the retransmissions. In particular, if the retransmission cannot be done so that the ACK can reach the sender before it times out, congestion control will kick in to reduce the sender's transmission rate significantly (by reducing CWnd to 1 and then restart from slow-start again). Specifically, TCP sets its timeout threshold, denoted by RTO, according to:

$$rto = srtt + 4 \times rtt_{dev} \tag{13}$$

where *srtt* is the smoothed RTT and rtt_{dev} is the smoothed mean deviation of RTT [16]. Thus to avoid timeout the ACK must arrive at the sender no later than the average RTT plus four times the mean RTT deviation.

Consider the scenario in Fig. 14, let RTT_{SR}, RTT_{SA}, RTT_{AR} be the RTT between sender and receiver, sender and accelerator and accelerator and receiver respectively. Normally RTT_{SR} is the sum of RTT_{SA} and RTT_{AR}. However, once local retransmission takes place, measured RTT of the lost packet will increase to $RTT_{SA}+2RTT_{AR}$. Depending on the extent of RTT variation of the mobile data network, the extra delay RTT_{AR} could be larger than four times the mean deviation of the RTT as measured by the sender.

Our measurement of a production 3G HSPA network recorded a mean RTT deviation at around 7 ms while the mean RTT_{AR} is around 100 ms. Thus in this case even the packet can be successfully retransmitted in one attempt, the extra delay incurred (i.e., 100 ms) will still trigger timeout at the sender, leading to a significant drop in the sender's CWnd.

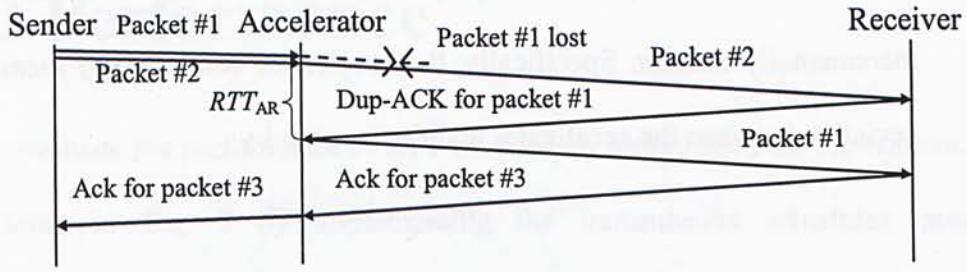

Fig. 14. RTT incurred by local retransmission

Now the time needed for retransmission depends on a number of factors, including the RTT between the accelerator and the receiver, transmission time, and accelerator processing time. The first two of these factors are network dependent and thus cannot be reduced further by the transport layer. On the other hand, we note that the timeout threshold at the sender is continuously updated through measurement of the average RTT and mean RTT deviation between the sender and the receiver whenever an ACK comes back.

In (13) we note that *srtt* is computed from measured RTT, which is in turn computed from the difference between the ACK packet arrival time and the transmission time of the original data packet. While we cannot change the way the sender computes the *srtt*, the accelerator could change the RTT as measured by the sender, simply by delaying the forwarding of ACK packets received from the receiver – *RTT modulation*.

The next question is the amount of delay to add. Intuitively a longer delay will allow more time to carry out local retransmission. However if the delay is too long then it may exceed the RTO threshold at the beginning of the connection, resulting in false triggering of timeout. Moreover, a long RTT will also lower the rate at which the sender ramps up the CWnd, thus degrading throughput at the beginning.

To tackle this dilemma we propose to increase the forwarding delay in an incrementally manner. Specifically, the accelerator continuously measures the RTT deviation between the accelerator and the receiver by

$$rtt_{dev\_i} = |rtt_i - srtt| \tag{14}$$

and then compute the smoothed mean RTT deviation from

$$rtt_{dev} = \frac{3}{4} rtt_{dev} + \frac{1}{4} rtt_{dev\_i} \tag{15}$$

Although the measurement does not include the delay and delay variations between the sender and the accelerator, the impact is generally insignificant as (a) the mobile link typically exhibit larger RTT and RTT variations than the wired link; and (b) RTT variations of the wired link is generally independent of the RTT variations of the mobile link, thus the mean RTT deviation measured by the sender will in general be not smaller than the mean RTT deviation of the mobile link.

Define L to be the number of retransmission attempts to accommodate. Then our goal is to add a ACK-forwarding delay, denoted by D, such that

$$D \geq L \times rtt \tag{16}$$

where rtt is the smoothed RTT between the accelerator and the receiver.

Denote the initial ACK-forwarding delay as D_0=0. Then whenever an ACK arrives at the accelerator, it will increase the ACK-forwarding delay by

$$D_{i+1} = D_i + \beta \times rtt_{dev} \tag{17}$$

where rtt_{dev} is the smoothed mean RTT deviation between the accelerator and the receiver. The parameter β controls how fast we increase the ACK-forwarding delay. An obvious limit is β<4 as the sender RTO will expire if the ACK arrives 4 times the smoothed mean RTT deviation later than the mean RTT.

6.2 Performance

To evaluate the performance of RTT modulation we repeated the experiment using the setup in Fig. 3 by implementing the transmission scheduler, proactive retransmission, and RTT modulation algorithms in the accelerator. We set $\beta = 2$ in (17). Fig. 15 plots the evolution of modulated RTO (RTOM), RTO and RTT versus time. Random packet loss occurred at 8.21 s. Fig. 16 shows an enlarged view of Fig. 15 around the time the loss event occurred. As the lost packet was retransmitted by the accelerator, the RTT of the locally retransmitted packet is increased beyond the original RTO. Thus without RTT modulation this will trigger sender timeout. By contrast, modulated RTT allows the extra margin for the local retransmission to complete, and in this case the increased RTT stays within the modulated RTT and thus the loss event can be successfully suppressed.

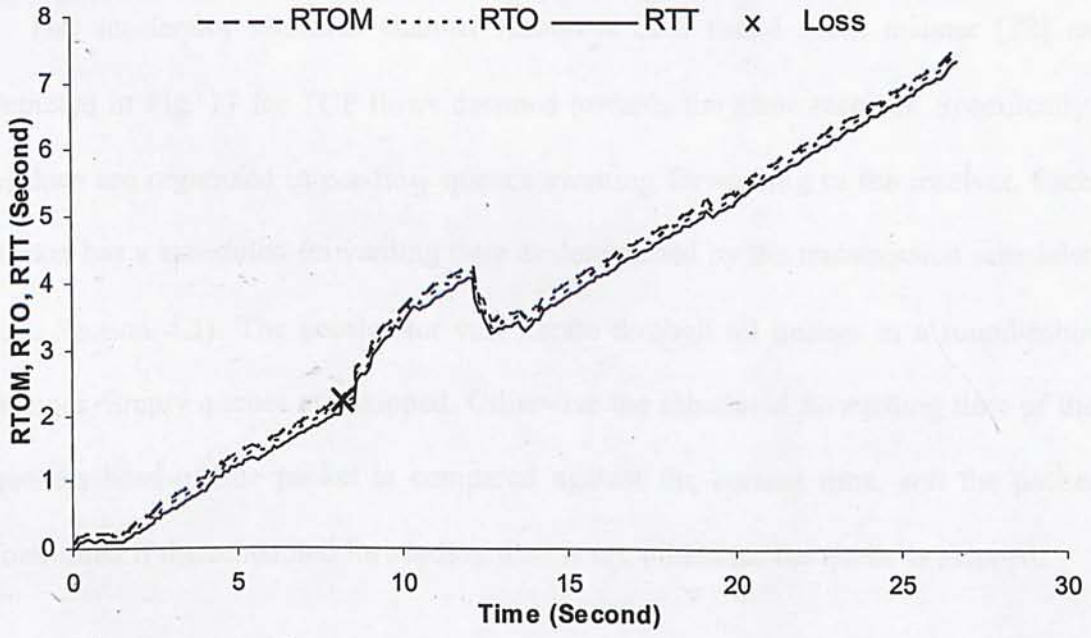

Fig. 15. Evolution of modulated RTO (RTOM), RTO and RTT versus time

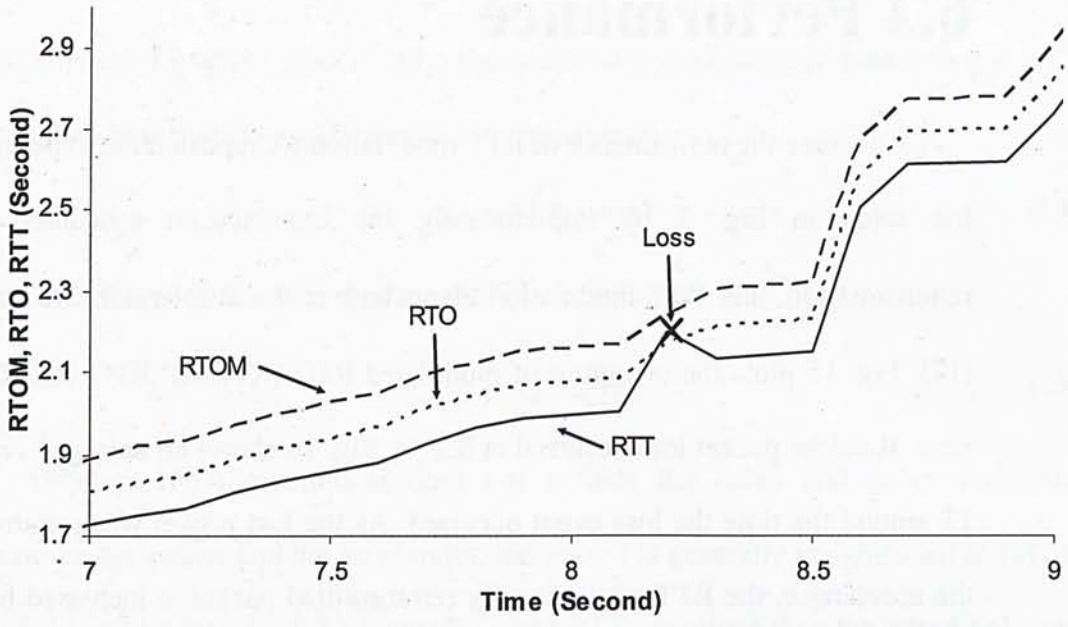

Fig. 16 Evolution of modulated RTO (RTOM), RTO and RTT versus time (enlarged around loss)

Chapter 7 Fairness

Congestion control between the accelerator and the receiver no longer follows the conventional TCP AIMD algorithm. In a mobile data network although network channel resources are allocated in a per-device basis, TCP flows destined to the same receiver[1] will still compete against each other for bandwidth. We investigate in this chapter the bandwidth sharing behavior of conventional TCP and that of the accelerator.

7.1 Packet forwarding

The accelerator allocates channel resources in a round robin manner [22] as depicted in Fig. 17 for TCP flows destined towards the same receiver. Specifically, packets are organized in per-flow queues awaiting forwarding to the receiver. Each packet has a scheduled forwarding time as determined by the transmission scheduler (c.f. Section 4.3). The accelerator will iterate through all queues in a round-robin manner. Empty queues are skipped. Otherwise the scheduled forwarding time of the queue's head-of-line packet is compared against the current time, and the packet forwarded if the scheduled forwarding time is up, otherwise the queue is skipped.

---

[1] The mobile data network may also be shared by multiple hosts using an NAT. In this case TCP flows destined to all hosts connected via the same mobile data link will compete against each other for bandwidth.

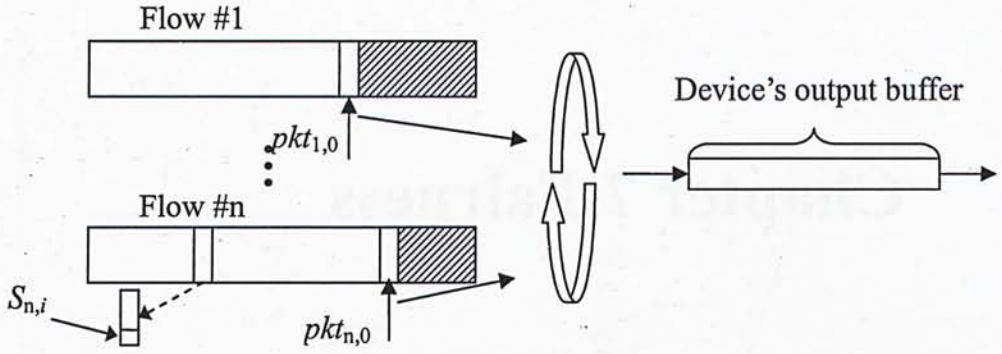

Fig. 17. Logic of Packet scheduling

To evaluate this component, we use two connections transferring 16 MByte of data while one of the connections starts 5 s earlier than another one such that the earlier link can achieve its maximum throughput and let the latter one compete with it. For the case of not using the accelerator, we manually set the receiver buffer size to 128 KB (which is larger than the BDP of the emulated link at 108 KB) and set random loss to zero to ensure that the two ordinary TCP connections (using Linux's default CUBIC congestion module) can fully utilize the link capacity.

Fig. 18 and Fig. 19 plot the 1-s averaged throughout of the two TCP flows for the case without and with accelerator respectively. The throughput is measured at the receiver. We observe that normal TCP flows without the accelerator exhibit more throughput fluctuations than the case with the accelerator. The throughput fluctuations are due to the bandwidth probing actions of the two TCP flows. By contrast the accelerator's round-robin scheduler allocates near equal bandwidth to the two competing TCP flows with negligible fluctuations.

The differences widen even further if we average the throughput over a shorter timescale, e.g., 100 ms in Fig. 20 and Fig. 21. In terms of fairness we employ Jain's fairness index [23] to quantify the comparisons. The fairness index can range from $1/n$ to 1 where n is the number of competing flows. Larger index value represents better fairness among the competing TCP flows. Table 4 lists the fairness indices for

three timescales, i.e., 1000 ms, 100 ms, and 10 ms. We can see that the accelerator can maintain a high fairness index even for timescale as short as 10 ms, at which point normal TCP performed poorly. These results demonstrate that although the accelerator does not implement TCP's AIMD congestion control algorithm it nevertheless can achieve even better fairness in bandwidth sharing.

Scales	1000 ms	100 ms	10 ms
Normal TCP	0.95	0.76	0.82
Using accelerator	0.95	0.98	0.99

Table 2 Jian's fairness index with different scale

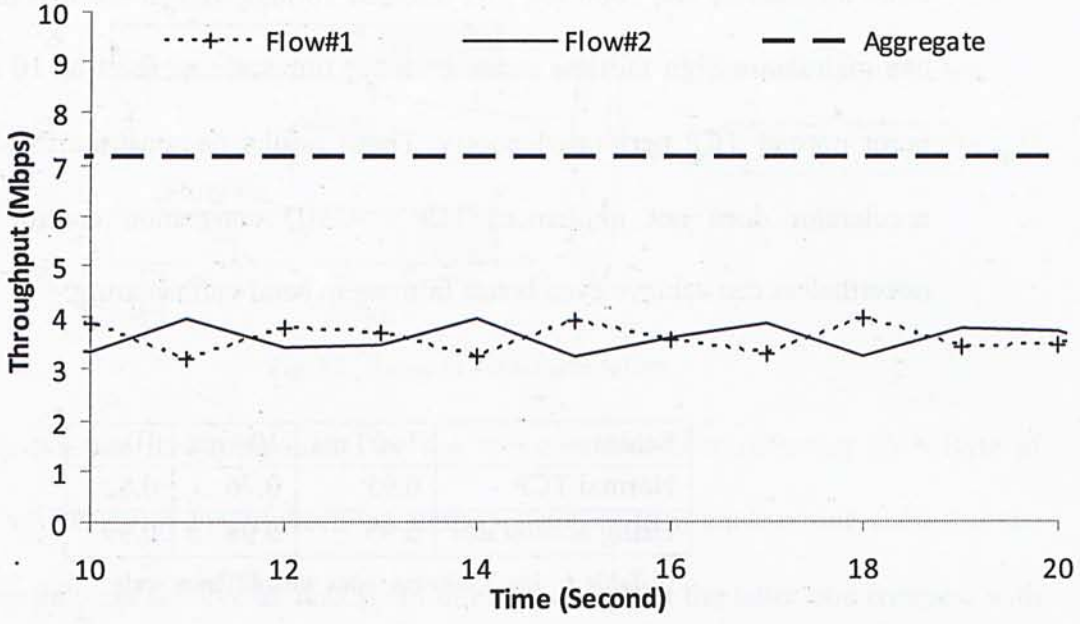

Fig. 18. Throughput variation of two competing flows without accelerator

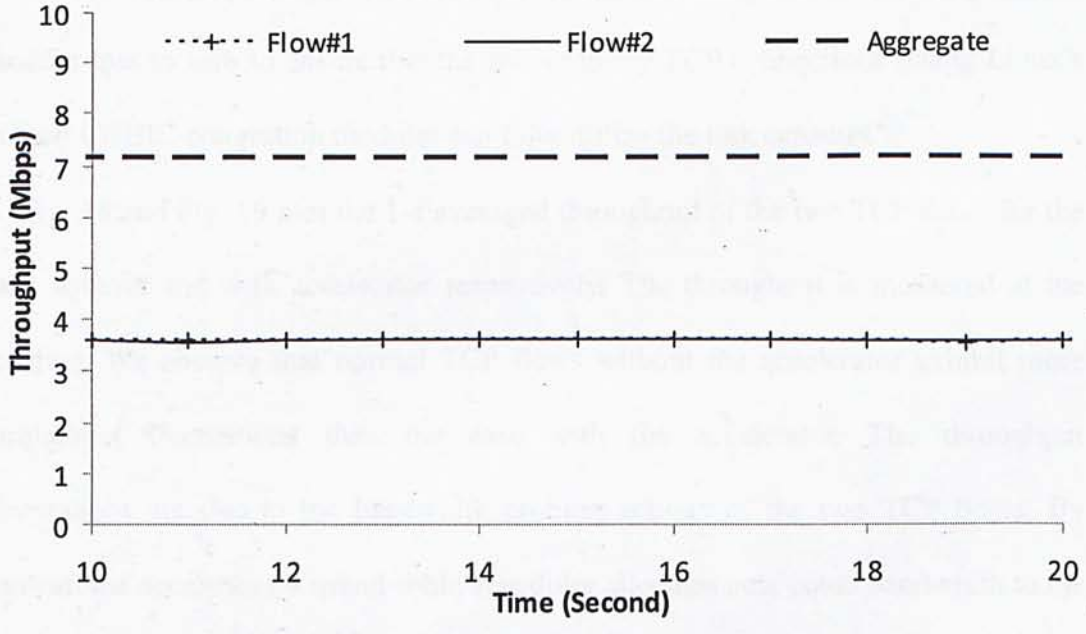

Fig. 19. Throughput variation of two competing flows with accelerator

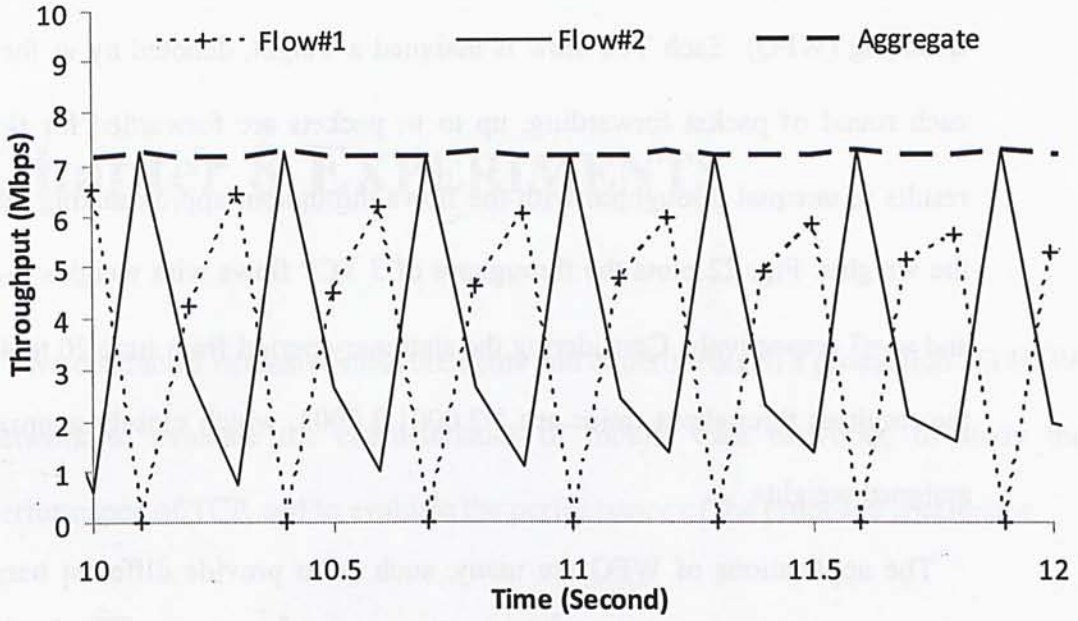

Fig. 20. Throughput variation of two competing flows without accelerator

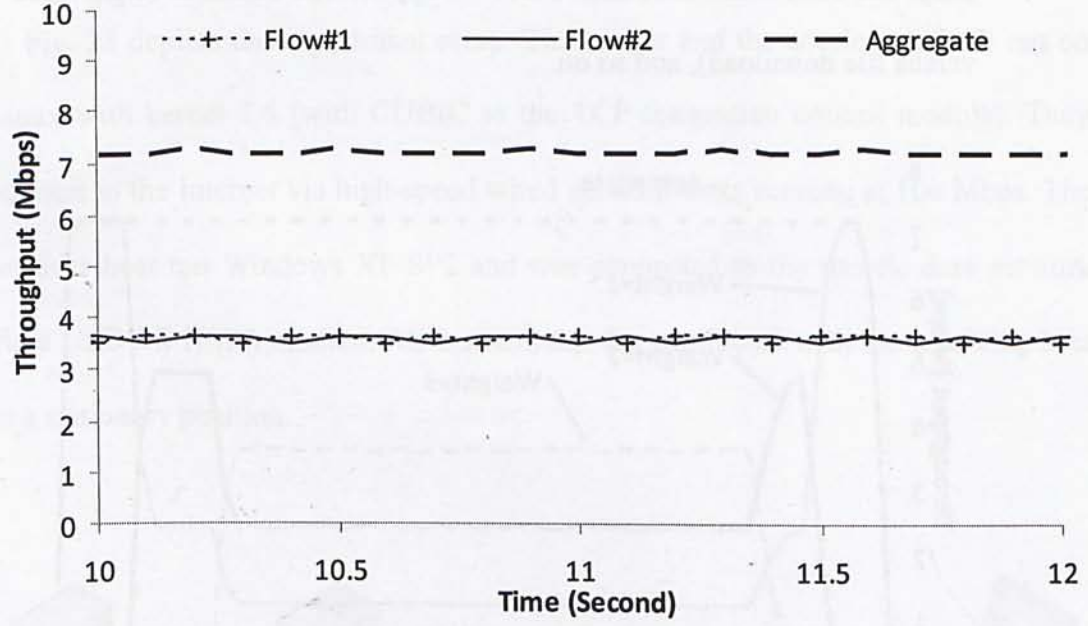

Fig. 21. Throughput varaition of two competing flows with accelerator

7.2 Non-uniform bandwidth allocation

In addition to round-robin scheduling, the accelerator opens the door to implement more sophisticated bandwidth allocation and channel scheduling algorithms. As an

example we implemented a generalized queueing discipline called weight fair queueing (WFQ). Each TCP flow is assigned a weight, denoted by w_i for flow i. In each round of packet forwarding, up to w_i packets are forwarded for flow i. This results in unequal throughput with the flows throughput approximating the ratios of the weights. Fig. 22 plots the throughput of 3 TCP flows with weights w_0=1, w_1=2, and w_2=3 respectively. Considering the stationary period from time 20 to 40 seconds the resultant throughput ratios are 1:2.0001:3.0003, which closely approximate the assigned weights.

The applications of WFQ are many, such as to provide different bandwidth to different applications (e.g., more bandwidth for web browsing versus P2P), or to adapt the bandwidth allocation based on application demands (e.g., video streaming versus file download), and so on.

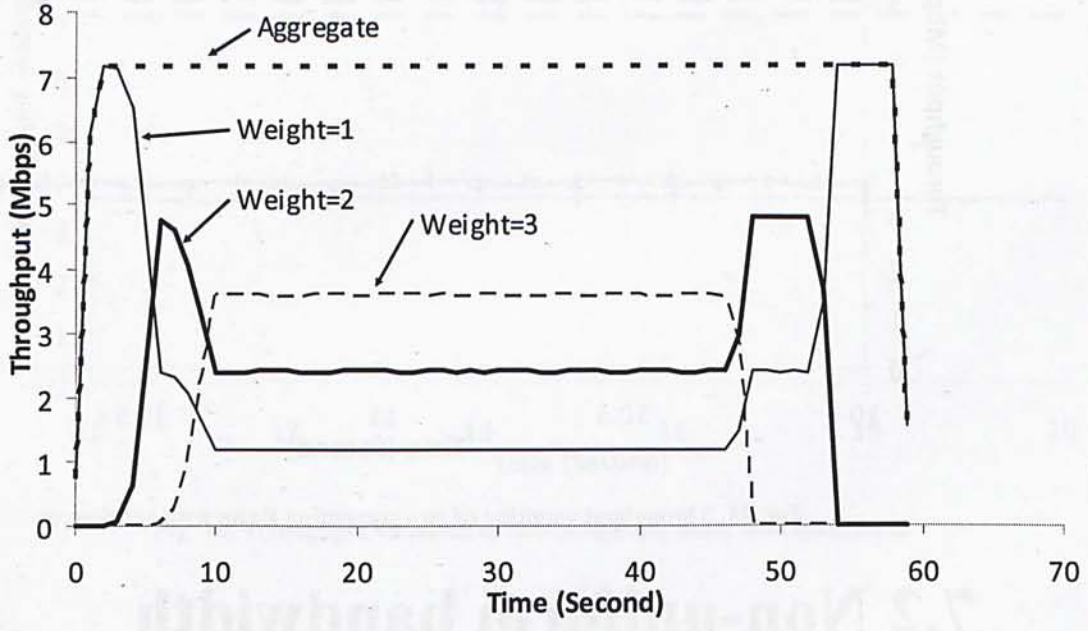

Fig. 22. Throughput variation of flows with different weight

Chapter 8 EXPERIMENTS

We conducted extensive measurements and experiments in a production 3G HSPA network to evaluate the characteristics of mobile data networks, to study the performance of TCP, and to evaluate the performance of the proposed accelerator.

8.1 Experiment setup

Fig. 23 depicts the experiment setup. The sender and the accelerator both ran on Linux with kernel 2.6 (with CUBIC as the TCP congestion control module). They connect to the Internet via high-speed wired network links running at 100 Mbps. The receiver host ran Windows XP SP2 and was connected to the mobile data network via a USB 3G HSPA modem. All experiments are conducted with the receiving host in a stationary position.

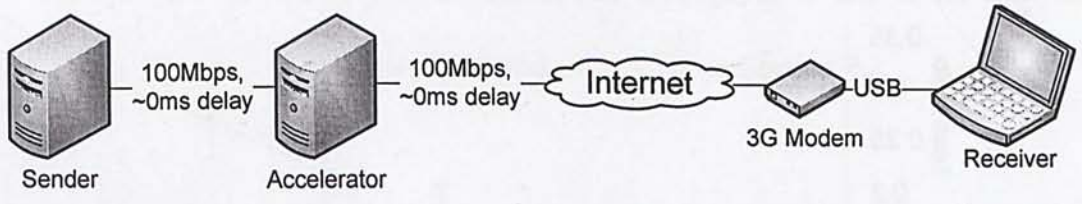

Fig. 23 Practical network setup

8.2 Packet loss

We first study the packet loss rate of the mobile link by sending UDP datagrams directly from the sender to the receiver, bypassing the accelerator, at various fixed rates ranging from 3 to 6.5 Mbps. Fig. 24 plots the packet loss ratio with 95% confidence interval versus various UDP data rates. Our measurement shows that the mobile link has an upper throughput limit at approximately 5.40 Mbps. Nevertheless as the results in Fig. 24 show the packet loss ratio is significant even at data rates lower than the mobile link's throughout limit. Moreover our results consistently show that the loss rates are data rate dependent – higher data rate generally results in higher packet loss ratio.

A second observation is that packet loss remains even if the data rate is low. For example, at a data rate or 3 Mbps we consistently measured a loss rate of approximately 0.6%. These losses are likely due to radio transmission errors which cannot be recovered by the link layer's retransmission mechanism such as HARQ [24].

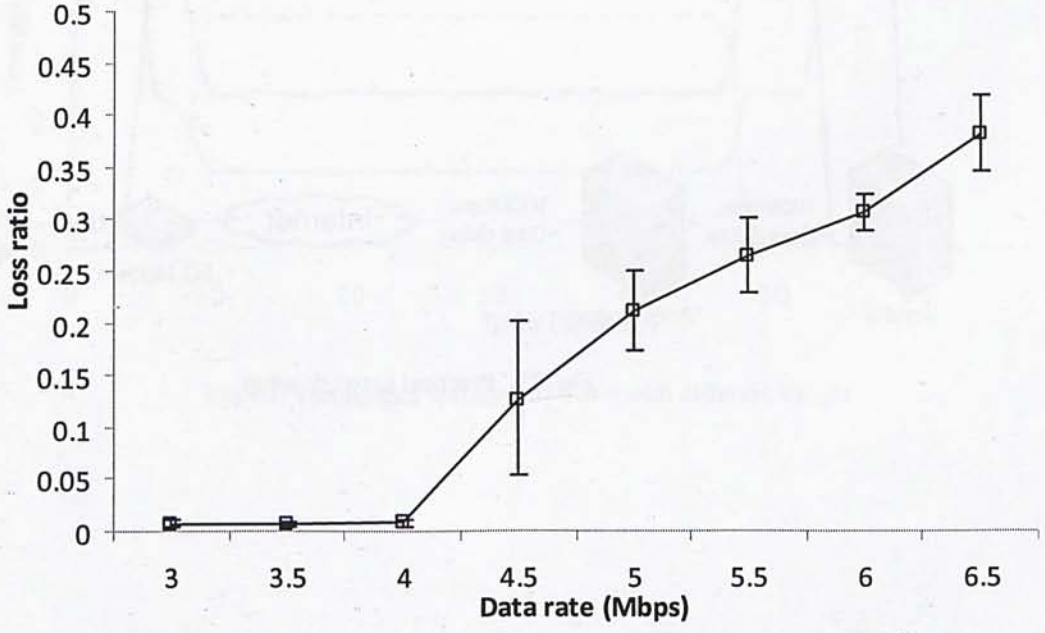

Fig. 24. Loss ratio with 95% confidence interval for data rates 3-6.5 Mbps

It is also worth noting that the extend of packet losses depends on many factors, such as the location of the mobile device, mobility of the mobile device, amount of radio interference, etc. We also conducted similar measurements in other mobile data networks and found different levels (higher) of packet losses. Nevertheless the previous two observations are consistent even across different mobile operators.

8.3 Unaccelerated TCP throughput

We first evaluate the throughput performance of normal TCP over the mobile data network. In addition to using the default receiver window size, which is 17 KB for Microsoft Windows XP, we also developed a custom application which explicitly increased the receiver window size by increasing the socket buffer size via the sockets API [12].

Fig. 25 plots the achievable TCP throughput versus receiver buffer size ranging from 17 KB to 128 KB. The BDP of the path is approximately 90 KB thus the 128 KB and the 192 KB settings already exceeded the BDP. Not surprisingly the throughput increases with larger receiver buffer size but we note that even with a receiver buffer size of 192 KB the achievable throughput is still lower than the throughput limit of 5.40 Mbps as measured using UDP in Section 8.2.

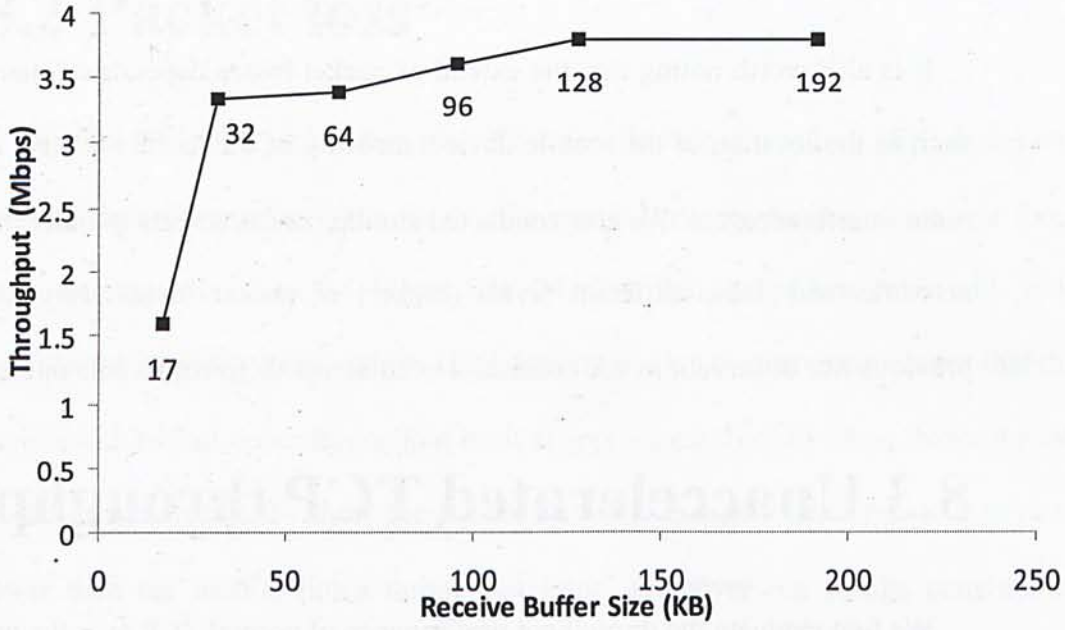

Fig. 25. Achievable throughput versus various receive buffer size

8.4 Accelerated TCP throughput

Next we repeat the throughput test by introducing the accelerator between the sender and the receiver. The receiver in this case employed the default receiver buffer size of 17 KB. The test transferred a 32 MB file from the sender to the receiver.

Table 3 summarizes the overall throughput and steady state throughput for both accelerated and unaccelerated cases. The former includes the ramp-up period of TCP while the latter does not. In either case the accelerator increases the achievable throughput by approximately 2.5 times. Moreover, the accelerated TCP achieved a throughput (4.03 Mbps) higher than the case for normal TCP with 192 KB receiver buffer size (3.80 Mbps). This is because the accelerator not only eliminated the AWnd-induced bottleneck, but also suppressed packet loss events to keep the sender CWnd at a high level.

	Overall Throughput (bps)	Steady State Throughput (bps)
Without Accelerator	1617918.03	1696274.89
With Accelerator	4033170.61	4255799.80
Ratio (With / Without)	2.49	2.51

Table 3 Improvement in both overall throughput and steady state throughput

8.5 Fairness

To evaluate the fairness in bandwidth sharing across competing TCP flows, we initiate two concurrent TCP connections between the sender and the receiver, both transferring data as fast as TCP allows. For the unaccelerated case we explicitly set the receiver window size to 128 KB while for the accelerated case we use the default window size of 17 KB.

We captured and measured the throughput of the two TCP flows at the receiver and then compute the Jain's fairness index using throughput data averaged over 100 ms intervals. The resultant fairness indices are 0.74 and 0.99 for the unaccelerated and accelerated cases respectively. Consistent with the results in Section 7.1 the accelerator can achieve much better fairness in sharing bandwidth across the competing TCP flows.

8.6 Mobile handset performance

In this section, we change the experiment setup to replace the PC-based receiver host by a mobile handset. We tested three mobile handsets: iPhone 3G, iPhone 3GS, and Nexus One. All three mobile handsets have built-in support for 3G HSPA. Throughput is measured by downloading a 4 MB image file from a web server running in the sender host. The measured throughput is summarized in Table 4.

	Without accelerator	With accelerator
iPhone 3G throughput (Mbps)	2.50	2.63
iPhone 3GS throughput (Mbps)	3.56	3.69
Nexus One throughput (Mbps)	2.93	4.02

Table 4 Throughput obtained with handsets with and without the accelerator

From the packet trace data we observe that both iPhone 3G and iPhone 3GS have a fixed receiver buffer size of 128 KB while Nexus One appeared to implement adaptive buffer sizing, with a dynamically variable receiver buffer size up to 81 KB.

Surprisingly, for iPhone 3G and iPhone 3GS the accelerator did not offer significant improvement in TCP throughput. Analysis of the traffic trace data revealed that the RTT measured by the sender was about 400 ms. However if we replace the mobile handset by a PC-based receiver the measured RTT was only 120 ms at the same data rate. As the same mobile data network is used in both cases we conclude that the differences in RTT must be due to processing delays incurred by the mobile handsets. In other words in the cases of iPhone 3G and iPhone 3GS the achievable throughput is in fact limited by the processing capacity of the receiver. It is worth nothing that this capacity limit has been incorporated into the accelerator's transmission scheduler and thus even without TCP's AIMD congestion control the accelerator can still correctly estimate the receiver's processing capacity and achieved its throughput limit.

For the Nexus One the accelerator did increase the achievable throughput from 2.93 Mbps to 4.02 Mbps. This suggests that the Nexus One has a higher processing capacity which allows the accelerated TCP flow to reach a higher throughput.

Chapter 9

FUTURE WORK

In this chapter we discuss three directions for future work: (a) dynamic AWnd control; (b) split-TCP; (c) dynamic resource allocation; and (d) sender-based acceleration.

9.1 Dynamic AWnd control

In the proposed opportunistic transmission the AWnd is rewritten with a large constant (e.g., 10MB) by the accelerator before it is forwarded to the sender. On one hand, while the constant AWnd works well in the 3G HSPA environment it may not be large enough for networks with even higher bandwidth. On the other hand, the accelerator must reserve buffer space of size AWnd bytes. If an accelerator needs to handle 10K flows than the buffer requirement alone will become 100 GB, which is not cost-effective.

Therefore one future direction is to investigate adaptive algorithms to monitor and estimate the throughput of on-going TCP flows and then dynamically adjust the AWnd to reduce buffer consumption, while still ensuring data are always available for forwarding to the receiver. Moreover, buffer sharing techniques may also provide further reduction in buffer requirement, especially for flows destined to the same receiver.

9.2 Split-TCP

The accelerator developed in this thesis maintains TCP's end-to-end performance guarantees such that data acknowledged are guaranteed to be received by the receiver. This is an important property in many applications, including finance, trading, e-banking, e-healthcare, and so on. Nevertheless if split-TCP can be applied, e.g., for non-critical applications, the accelerator can then further modify the end-to-end congestion control algorithm which may provide further performance gains.

Specifically, similar to flow control the accelerator can decouple congestion control between the sender and the receiver altogether. In this split-TCP approach the accelerator may acknowledge packets before the receiver does, thus substantially speeding up the growth of CWnd at the sender. This will likely improve the performance of short-lived TCP flows, such as web browsing. In addition, packet loss events can also be suppressed completely by the accelerator, thereby keeping the sender from triggering congestion control in the event of random or even congestion-induced packet losses in the mobile link.

9.3 Dynamic resource allocation

In Chapter 7.2 we described a modification to the packet forwarded in the accelerator to achieve non-uniform bandwidth allocation. Beyond WFQ the accelerator can also be implemented to provide priority scheduling (e.g., higher priority for real-time traffic), to guarantee bandwidth availability (e.g., for streaming video), or to perform traffic policing (e.g., to limit throughput of P2P traffics). This provides a fertile ground for further investigation.

9.4 Sender-based acceleration

While the proposed acceleration algorithms are designed for use in a network-centric accelerator. Some of them in fact may also be implemented at the sender TCP module. For example, opportunistic transmission and to a lesser extent loss event suppression are candidates for implementation in the sender. However sender-based approach does face new challenges, including the differentiation of mobile and non-mobile TCP flows, the existence of proxied servers in mobile operators, and so on. More research is warranted to investigate the applicability and performance of sender-based acceleration.

Chapter 10

CONCLUSION

This thesis tackled the performance problem of running TCP over mobile data networks by introducing an accelerator between the sender and the receiver. The accelerator implements three acceleration algorithms: (a) opportunistic transmission to overcome the AWnd-induced throughput bottleneck; (b) local retransmission to prevent throughput blackout in the event of packet loss; and (c) loss event suppression to keep the sender CWnd at a high level.

Experimental results conducted in production 3G HSPA networks show that the accelerator can increase the throughput performance of TCP by up to 2.5 times of the unaccelerated TCP. Moreover, the accelerator achieves better fairness among competing TCP flows and can also be equipped with dynamic resource allocation algorithms to offer more sophisticated traffic control for the TCP flows.

Last but not least, the proposed accelerator does not require modification to the applications, TCP implementation at the hosts, or operating system; and thus can be readily deployed in current and future mobile data networks.

BIBLIOGRAPHY

[1] V. Jacobson, R. Braden and D. Borman, "RFC 1323: TCP extensions for high performance," May 1992

[2] H. Sivakumar, S. Bailey and R. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," *Proceedings of Super Computing*, November 2000

[3] T. Hacker, B. Noble and B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP," *Proceedings of IEEE Infocom 2004*, March 2004

[4] M. Allman, S. Floyd and C. Partridge, "RFC 3390: Increasing TCP's Initial Window," October 2002

[5] I. Rhee and L. Xu "CUBIC: A new TCP-friendly high-speed TCP variant," *Proceedings of PFLDNet'05*, February 2005

[6] L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," *Proceedings of IEEE INFOCOM 2004*, March 2004

[7] C. Jin, D. X. Wei and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *Proceedings of IEEE INFOCOM 2004*, March 2004

[8] R. Shorten and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks," *Second International Workshop on Protocols for Fast Long-Distance Networks*, February 16-17, 2004, Argonne, Illinois USA

[9] Lai, C., Leung, K.-C., and Li, V. O. K., "TCP-NCL: A Unified Solution for TCP

Packet Reordering and Random Loss," *Proceedings of the 20th Personal, Indoor and Mobile Radio Communications Symposium 2009 (IEEE PIMRC 2009)*, pp. 1093-1097, Tokyo, Japan, 13-16 September 2009.

[10] M. Fisk and W.-C. Feng, "Dynamic Right-Sizing in TCP," *Proceedings of the Los Alamos Computer Science Institute Symposium*, October 2001

[11] S. Floyd, T. Henderson, "RFC 2582: New Reno Modification to TCP's Fast Recovery," April 1999

[12] J. Davies, "The Cable Guy: TCP Receive Window Auto-Tuning," *TechNet Magazine*, January 2007

[13] S. Vangala and M. A. Labrador, "The TCP SACK-Aware Snoop Protocol for TCP over Wireless Networks," IEEE Vehicular Technology Conference, October 2003

[14] J.H. Hu and K. L. Yeung, "FDA: A Novel Base Station Flow Control Scheme for TCP over Heterogeneous Networks," In Proceedings of IEEE INFOCOM 2001

[15] The Linux Foundation, "Net:Netem," Available: http://www.linuxfoundation.org/en/Net:Netem

[16] V. Paxson, M. Allman, " RFC 2988: Computing TCP's Retransmission Timer," November 2000

[17] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang: "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *Proceedings of the ACM Mobicom 2001*

[18] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, Apr. 1999

[19] B. S. Bakshi, N. Vaidya, and D. K. Pradhan, "Improving the performance of TCP over Wireless Networks," In *Proceedings of 17th, International Conference on DCS*, July 1997

[20] M. Mathis, J. Mandavi, S. Floyd, and A. Romanov., "TCP Selective Acknowledgement Options," *IETF RFC 2018*, 1996

[21] A. Bakre and B. R. Badrinath. "I-TCP: Indirect TCP for Mobile Hosts," In *Proceedings of ICDCS*, pages 126-143, 1995

[22] R. Sierra, "Fair Queuing in Data Networks," *Internetworking 2002*, pages 1- 6

[23] R.Jain, D-M. Chiu and W. Hawe. "A Quantitative Measure of Fairness and Discrimination For Resource Allocation in Shared Conputer Systems," Technical Report TR-301, DEC Research Report, September, 1984

[24] P. Frenger, S. Parkvall, and E. Dahlman, "Performance comparison of HARQ with chase combining and incremental redundancy for HSDPA," *In Proceeding. IEEE 54th Vehicular Technology Conference, vol. 3*, October 7–11, 2001, page 1829–1833