

How Network Asymmetry Affects TCP

Hari Balakrishnan, MIT

Venkata N. Padmanabhan, Microsoft Research

ABSTRACT

Several emerging wireline and wireless access network technologies exhibit asymmetry in their network characteristics. For instance, cable modem networks exhibit significant bandwidth asymmetry, while packet radio networks exhibit media access asymmetry. A high degree of asymmetry can have an adverse effect on the performance of feedback-based transport protocols. In this article we study the effects of bandwidth and media access asymmetry on the performance of the TCP protocol. We identify the fundamental reasons for the mismatch between TCP and asymmetric networks, and then present several techniques to address this problem.

INTRODUCTION

The ever-increasing desire of users for high-speed and ubiquitous connectivity has led to the deployment of many new network access technologies. Some of these, such as cable modem, digital subscriber line (DSL), and satellite-based networks, are aimed at alleviating the “last mile” bottleneck, while others, such as wireless packet radio networks, are motivated by the need to provide users with tetherless access to the Internet, especially to their mobile devices.

These networking technologies often exhibit *asymmetry* in their network characteristics — the network characteristics in one direction may be quite different than those in the opposite direction. For instance, much of the existing cable plant was designed for unidirectional (broadcast) transmission from the headend out to customer premises. As such, the *upstream* bandwidth of the cable plant, from the customer premises out to the Internet, is often limited compared to its *downstream* bandwidth toward the customer premises. In some cases, upstream communication on the same technology may simply be impossible; old cable plants with unidirectional amplifiers and direct broadcast satellite systems such as DirectPC [5] are examples of this. This necessitates the use of a different (and often slower) network technology, such as a dialup modem line, for upstream connectivity. Figure 1 depicts such a setup in the context of a “wire-

less” cable modem network (also known as *multichannel multipoint distribution service*, MMDS).

Network asymmetry can adversely impact the performance of feedback-based transport protocols such as TCP. The reason for this is that even if the network path in the direction of data flow is uncongested, congestion in the opposite direction can disrupt the flow of feedback. This disruption can lead to poor performance.

In the case of TCP, the feedback takes the form of acknowledgment packets (ACKs). A TCP sender depends on the receipt of a steady stream of ACKs to clock out new data packets (*ack clocking* [6]). If the ACKs get bunched up, the sender may burst out data, which could lead to packet loss on the forward path. This is not the only reason for degraded performance — a TCP sender depends on the receipt of ACKs to grow its *congestion window*, which governs its transmission rate. Again, disruption of the ACK stream can disrupt window growth and degrade performance to a fraction of the available bandwidth.

In this article we discuss the performance problems caused by network asymmetry in the context of TCP. We present several techniques that we have developed to address these problems. Our research results show that with these techniques, it is possible to obtain a TCP capable of near-optimal performance under a variety of asymmetric conditions.

CLASSIFICATION OF ASYMMETRY

Network asymmetry can take several forms. We present a brief classification to help structure our discussion.

- **Bandwidth asymmetry:** Typically, the downstream bandwidth is 10–1000 times the upstream bandwidth. Examples include cable modem, asynchronous DSL (ADSL), and satellite-based networks, especially in configurations that depend on a dialup link for upstream connectivity.
- **Media access asymmetry:** This manifests itself in several ways. In a cellular wireless network, a centralized base station incurs a lower medium access control (MAC) overhead in transmitting to distributed mobile hosts than the other way around. In a pack-

Much of the material in this article is derived from the authors' Ph.D. dissertations [1, 2], and from [3, 4].

et radio network (e.g., the Metricom Ricochet™ network), the MAC overhead makes it more expensive to constantly switch the direction of transmission than to transmit steadily in one direction.

- **Loss rate asymmetry:** The network may inherently be more lossy in one direction than in the other.

Next, we analyze the problems caused by bandwidth and media access asymmetry in detail.

BANDWIDTH ASYMMETRY

We first discuss the case where TCP transfers happen only in the downstream direction and then turn to the bidirectional case.

Unidirectional Data Transfer — A common example of this is a user downloading data from a server. For simplicity, we restrict ourselves to the case of a single TCP connection in the downstream direction.

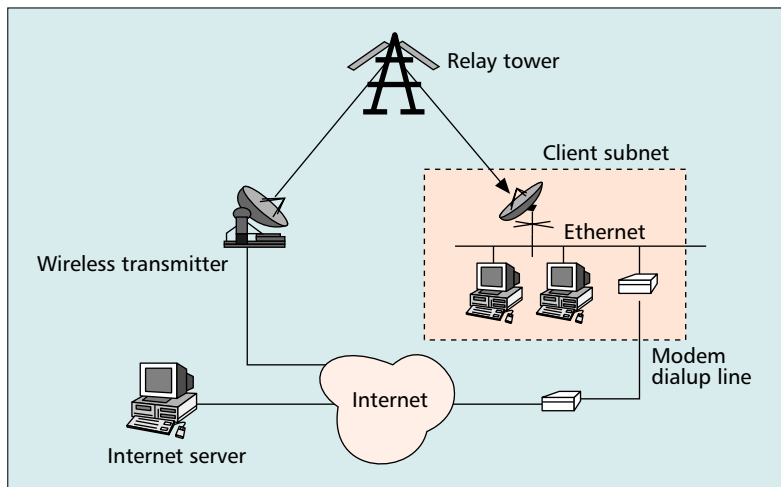
As defined in [7], the *normalized bandwidth ratio*, k , between the downstream and upstream paths is the ratio of the raw bandwidths divided by the ratio of the packet sizes used in the two directions. For example, for a 10 Mb/s downstream channel and a 100 kb/s upstream channel, the raw bandwidth ratio is 100. With 1000-byte data packets and 40-byte ACKs, the ratio of the packet sizes is 25, so k is $100/25 = 4$.

The ratio, k , holds the key to the behavior of TCP in an asymmetric network setting. If the receiver transmits more than one ACK every k data packets, the upstream bottleneck link will get saturated before the downstream one does. This will force the sender to clock out data more slowly than optimal, thus decreasing throughput.

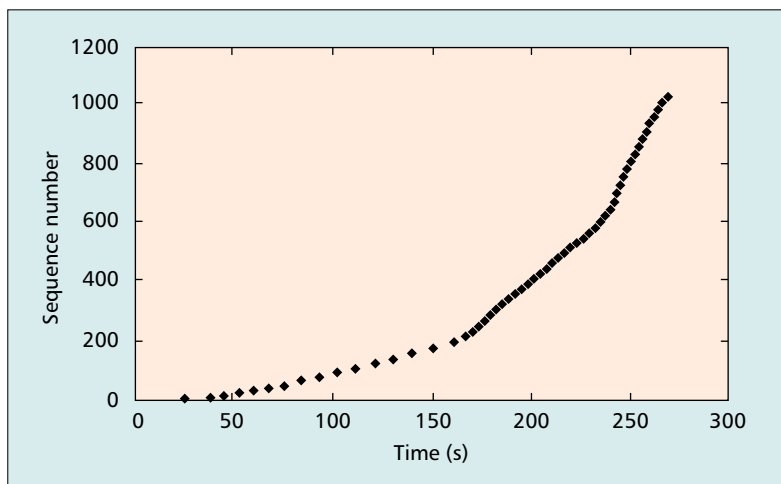
If the upstream bottleneck remains congested for a sustained length of time, the corresponding buffer will fill up, causing ACKs to be dropped. On average, only one ACK will get through for every k data packets transmitted by the sender, which can degrade performance in several ways. First, the sender could burst out k packets at a time, which increases the chance of data packet loss, especially when k is large. Second, since conventional TCP senders base congestion window growth on *counting* the number of ACKs and not on how much data is actually acknowledged, infrequent ACKs result in slower growth of the congestion window. Finally, the loss of (the now infrequent) ACKs elsewhere in the network could cause long idle periods while the sender waits for subsequent ACKs to arrive.

Bidirectional Data Transfers — We now consider the case when both downstream and upstream TCP transfers occur simultaneously. An example of this is a user sending out data (e.g., an e-mail message) while simultaneously receiving other data (e.g., Web pages).

The presence of bidirectional traffic effectively increases the degree of bandwidth asymmetry for the downstream transfer, thereby exacerbating the problems discussed in the above section. In addition, there are other effects that arise due to the interaction between data packets of the upstream transfer and ACKs of the downstream transfer. Essentially, the former can quickly fill up the upstream buffer, causing a large delay



■ **Figure 1.** The network topology of the wireless cable modem network, which illustrates bandwidth asymmetry. Hosts on the client subnet receive data from the Internet via the 10 Mb/s wireless cable upstream link and send out data via a low-bandwidth (e.g., dialup) downstream link.



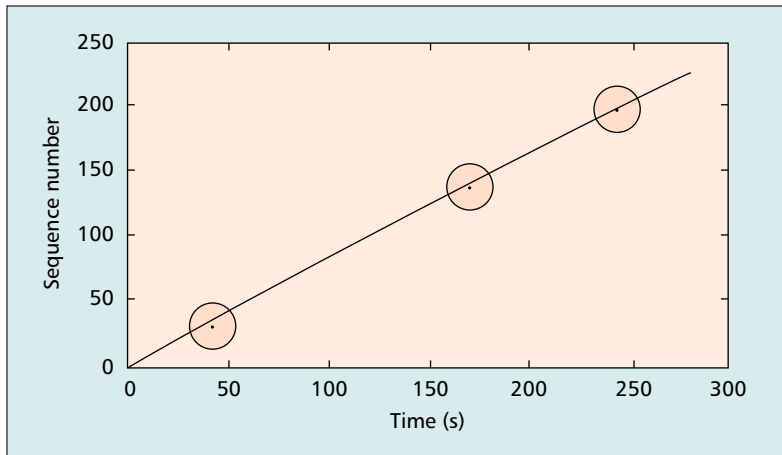
■ **Figure 2.** Measurements of the downstream connection operating over a 10 Mb/s wireless cable modem network. The sharp upswings in its data rate occur whenever the upstream connection suffers a loss and slows down (Fig. 3).

and a high loss rate for the latter. For instance, a single 1 kbyte data packet of the upstream transfer can add a whopping 1 s of queuing delay for ACKs of the downstream transfer. The resulting performance problem is illustrated in Figs. 2 and 3, which show that the downstream transfer makes significant progress only when the upstream transfer suffers a hiccup.

In summary, the presence of bidirectional traffic exacerbates the problems due to bandwidth asymmetry because of the adverse interaction between data packets of an upstream connection and ACKs of a downstream connection.

MEDIA ACCESS ASYMMETRY

As discussed earlier, media access asymmetry manifests itself in several ways. However, the fundamental cause is the same: uneven access to a shared medium by a distributed set of nodes. In a hub-and-spokes model, a central base station has complete knowledge of and control over the downstream traffic. Hence, it suffers a lower



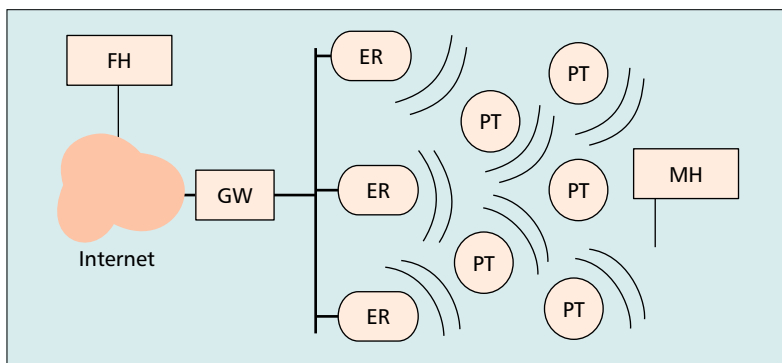
■ **Figure 3.** Measurements of the upstream connection operating over a 9.6 kb/s dialup line. The circles indicate times when the upstream connection retransmits packets.

MAC overhead than the distributed nodes that contend for the uplink. On the other hand, in a packet radio network the peer nodes are equals. Nevertheless, the MAC overhead makes it expensive to transmit packets in one direction when there is an ongoing data transfer in the opposite direction. In the remainder of this section, we investigate this issue in detail.

Although we use a network modeled after Metricom's Ricochet network, our general results and conclusions are applicable to packet radio networks in general. We start by describing the underlying packet radio network technology used in our study.

Topology — The network topology of the packet radio network is shown in Fig. 4. The maximum link speed between two nodes in the wireless cloud is 100 kb/s. Packets from a fixed host (FH) on the Internet are routed via a gateway (GW) and through poletop radios (PTs) to the mobile host (MH). There are typically between one and three wireless hops before the packet reaches the MH.

Half-Duplex Radios — Because the transmission power drowns incoming receptions in the same frequency band, the radio units in the network are *half-duplex*. This means that they can-



■ **Figure 4.** Topology of a packet radio network. The MH has a modem attached to it, which communicates with the FH on the Internet through the PTs and Ethernet radios (ERs). The GW routes packets between the packet radio network and the Internet.

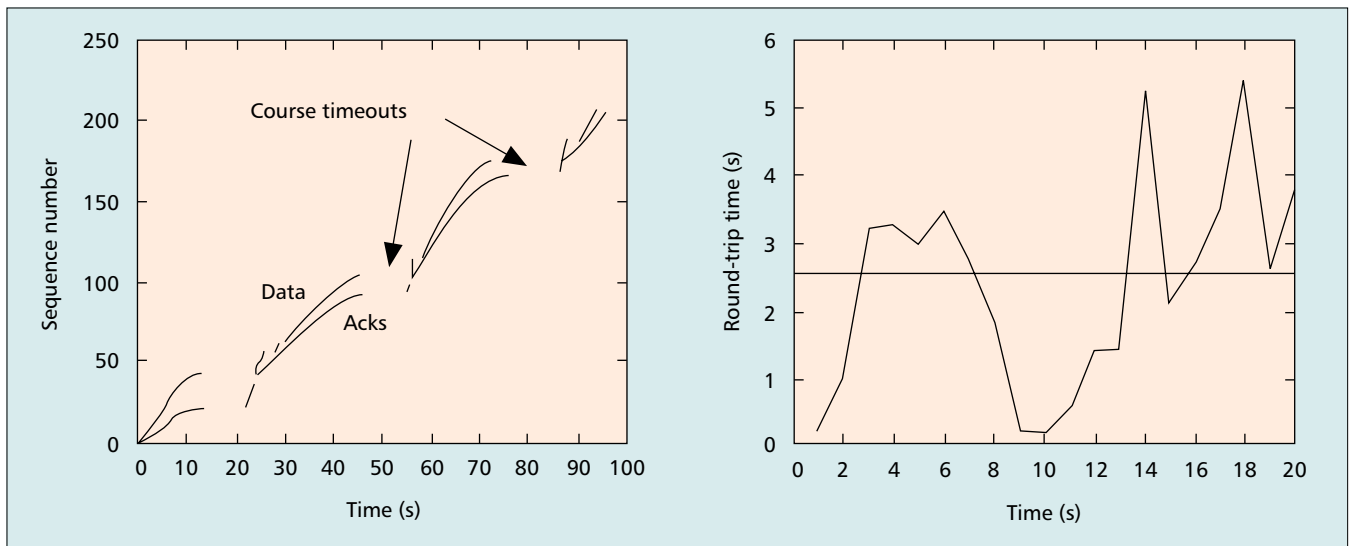
not simultaneously transmit and receive data. After transmitting (receiving) a packet, a sender (receiver) wanting to receive (transmit) needs to turn around and change modes. Moving from transmitting to receiving mode takes a nontrivial amount of time, called the *transmit-to-receive turnaround time*, T_{TR} . Similarly, going from receiving to transmitting mode takes a time equal to the *receive-to-transmit turnaround time*, T_{RT} .

Media Access — The radios in the network are frequency-hopping spread-spectrum units operating in the 915 MHz industrial, scientific, medical (ISM) band. The details of the frequency-hopping protocol are not relevant to transport performance, since the predominant reason for poor performance is the interactions between the MAC protocol and TCP. The MAC protocol for contention resolution is based on a polling scheme, similar (but not identical) to the Request-To-Send/Clear-To-Send (RTS/CTS) protocol used in the IEEE 802.11 standard. A station wishing to communicate with another (called the *peer*) first sends it an RTS message. If the peer is not currently communicating with any other station and is willing to receive this transmission, it sends a CTS message acknowledging the RTS. When this is received by the initiator, the data communication link is established. A data frame can then be sent to the peer. If the peer cannot currently communicate with the sender because it is communicating with another peer, it does not send a CTS, which causes the sender to back off for a random amount of time and schedule the transmission for later. It could also send a negative ACK (NACK)-CTS to the sender, which achieves the same effect. In all of this, care is taken by both stations to ensure that messages and data frames are not lost because the peer was in the wrong mode, by waiting enough time for the peer to change modes. To do this, each station maintains the value of the turnaround times of its peers in the network.

Error-control — The reliable link layer protocol used in this network for error control is a simple frame-by-frame protocol with a window size of 1. When a frame is successfully received, the receiver sends a link-level ACK to the sender. If the frame is not received successfully, the sender retransmits it after a timeout period. Such simple link layer protocols are the norm in several packet radio networks.

The need for the communicating peers to first synchronize via the RTS/CTS protocol and the significant turnaround time for the radios result in a high per-packet overhead. Furthermore, since the RTS/CTS exchange needs to back off when the polled radio is otherwise busy (for example, engaged in a conversation with a different peer), this overhead is variable. This leads to large and variable communication latencies in such networks. In addition, with an asymmetric workload with most data flowing in one direction to clients, ACKs tend to get queued in certain radio units, such as the client modems.

These variable latencies and queuing of ACKs adversely affect smooth data flow. In particular, TCP ACK traffic interferes with the flow of data



■ **Figure 5.** a) Packet and ACK sequence trace of a 200 kbyte TCP bulk transfer measured over one wireless hop in the Ricochet network: the three pauses are sender timeouts, lasting between 9 and 12 s each because large round-trip time variations cause the retransmission timeout estimate to be very long; b) 20 round-trip time samples collected during this connection: the samples have a mean of about 2.5 s and a standard deviation of about 1.5 s.

and increases the traffic load on the system. Figure 5a shows the packet sequence trace of a measured 200 kbyte TCP transfer over an unloaded wired path and one wireless hop in the packet radio network. This clearly shows the effect of the radio turnarounds and increased variability affecting performance. The connection progresses well for the most part, except for the three timeouts that occur during the transfer. These timeouts last between 9 and 12 s each, keeping the connection idle for a total of 35 percent of the total transfer time! These timeouts clearly underutilize the available bandwidth for the connection.

Why are these timeouts so long in duration? Ideally, the round-trip time estimate ($srtt$) of a TCP data transfer will be relatively constant (i.e., have a low linear deviation, $mdev$). Then the TCP retransmission timeout, set to $srtt + 4 * mdev$, will track the smoothed round-trip time estimate and respond well when multiple losses occur in a window. Unfortunately, this is not true for connections in this network, as shown in Fig. 5b. This figure plots the individual round-trip time estimates for 20 successive samples during the same connection over the Ricochet network. The mean value of these samples is about 2.5 s, and the standard deviation is large, about 1.5 s. Because of the high variation in the individual samples, the retransmission timer is on the order of 10 s, leading to the long idle timeout periods.

In general, it is correct for the retransmission timer to trigger a segment retransmission only after an amount of time that is dependent on both the round-trip time and the linear (or standard) deviation. If only the mean or median round-trip estimates are taken into account, there is a significant potential for spurious retransmissions of segments still in transit. Thus, our challenge is to devise solutions to this problem where the increased ACK flow and ACK queuing lead to variable latencies and long timeout periods.

An optimization to the link layer error control protocol that piggybacks link layer ACKs

with data reduces TCP round-trip variations to some extent. This scheme is motivated by the observation that the radios turnaround for both data frames as well as link layer ACKs. The presence of traffic in both directions, even when caused by TCP ACKs, already causes turnarounds to happen. So, if link layer ACKs were piggybacked with data frames, some extra radio turnarounds could be eliminated.

Despite this optimization, the fundamental problem of additional traffic and underlying protocols affecting round-trip time estimates and causing variabilities in performance still persists. Connections traversing multiple hops of the wireless network are more vulnerable to this effect, because it is now more likely that the radio units may already be engaged in conversation with other peers.

SOLUTIONS

Our discussion thus far makes it clear that there are two key issues that need to be addressed in order to improve TCP performance over asymmetric networks. The first issue is managing bandwidth usage on the uplink, used by ACKs (and possibly other traffic). Many of the techniques to address this issue work by reducing the number of ACKs that flow over the upstream channel, which has the potential of destroying the (desirable) self-clocking property of the TCP sender where new data transmissions are triggered by incoming ACKs. Thus, the second issue is to avoid any adverse impact of infrequent ACKs.

Each of these issues can be handled by local link layer solutions and/or end-to-end techniques. In this section we discuss solutions of both kinds.

UPLINK BANDWIDTH MANAGEMENT

Uplink bandwidth management may be performed by controlling the degree of compression, frequency, and scheduling of upstream ACKs.

It is important to note that with ACC, both data packets and TCP ACKs are candidates for being marked with an ECN bit. Therefore, upon receiving an ACK packet with the ECN bit set, the TCP receiver reduces the rate at which it sends ACKs.

TCP Header Compression — TCP header compression [8] describes TCP header compression for use over low-bandwidth links running SLIP or Point-to-Point Protocol (PPP). Because it greatly reduces the size of ACKs on the uplink when losses are infrequent (a situation which ensures that the state of the compressor and decompressor are synchronized), we recommend its use over low-bandwidth uplinks where possible. However, this alone does not address all of the problems:

- As discussed later, in certain networks there is a significant per-packet MAC overhead that is independent of packet size.
- A reduction in the size of ACKs does not prevent adverse interaction with large upstream data packets in the presence of bidirectional traffic.

Hence, in order to effectively address the performance problems caused by asymmetry, there is a need for techniques over and beyond TCP header compression.

ACK Filtering — *ACK filtering (AF)* is a TCP-aware link layer technique that reduces the number of TCP ACKs sent on the upstream channel. The challenge is to ensure that the sender does not stall waiting for ACKs, which can happen if ACKs are removed indiscriminately on the upstream path. AF removes only certain ACKs without starving the sender by taking advantage of the fact that TCP ACKs are cumulative. As far as the sender's error control mechanism is concerned, the information contained in an ACK with a later sequence number subsumes the information contained in any earlier ACK.¹ When an ACK from the receiver is about to be queued at the upstream bottleneck router, the router or the end host's link layer (if the host is directly connected to the constrained link) checks its queues for any older ACKs belonging to the same connection. If any are found, it removes them from the queue, thereby reducing the number of ACKs that go back to the sender. The removal of these "redundant" ACKs frees up buffer space for other data and ACK packets. AF does not remove duplicate or selective ACKs from the queue to avoid causing problems to TCP's data-driven loss recovery mechanisms.

The policy that the filter uses to drop packets is configurable and can be either deterministic or random (similar to a random-drop GW, but taking the semantics of the items in the queue into consideration). State needs to be maintained only for connections with at least one packet in the queue. However, this state is soft, and if necessary can easily be reconstructed from the contents of the queue.

ACK Congestion Control — *ACK congestion control (ACC)* is an alternative to ACK filtering that operates end-to-end rather than at the upstream bottleneck router. The key idea in ACC is to extend congestion control to TCP ACKs, since they do make nonnegligible demands on resources at the bandwidth-constrained upstream link. ACKs occupy slots in the

upstream channel buffer, whose capacity is often limited to a certain number of packets (rather than bytes).

ACC has two parts:

- A mechanism for the network to indicate to the receiver that the ACK path is congested
- The receiver's response to such an indication

One possibility for the former is the Random Early Detection (RED) algorithm [9] at the upstream bottleneck router. The router detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds a threshold, the router selects a packet at random and marks it, i.e., it sets an explicit congestion notification (ECN) bit in the packet header. This notification is reflected back to the upstream TCP end host by its downstream peer.

It is important to note that with ACC, both data packets and TCP ACKs are candidates for being marked with an ECN bit. Therefore, upon receiving an ACK packet with the ECN bit set, the TCP receiver reduces the rate at which it sends ACKs. The TCP receiver maintains a dynamically varying delayed-ACK factor, d , and sends one ACK for every d data packets received. When it receives a packet with the ECN bit set, it increases d multiplicatively, thereby also decreasing the frequency of ACKs multiplicatively. Then for each subsequent round-trip time (determined using the TCP timestamp option) during which it does not receive an ECN, it linearly decreases the factor d , thereby increasing the frequency of ACKs. Thus, the receiver mimics the standard congestion control behavior of TCP senders in the manner in which it sends ACKs.

There are bounds on the delayed-ACK factor d . Obviously, the minimum value of d is 1, since at most one ACK should be sent per data packet. The maximum value of d is determined by the sender's window size, which is conveyed to the receiver in a new TCP option. The receiver should send at least one ACK (preferably more) for each window of data from the sender. Otherwise, it could cause the sender to stall until the receiver's delayed-ack timer (usually set at 200 ms) kicks in and forces an ACK to be sent.

Despite RED+ECN, there may be times when the upstream router queue fills up, and it needs to drop a packet. The router can pick a packet to drop in various ways. For instance, it can drop from the tail, or it can drop a packet already in the queue at random.

ACKs-First Scheduling — In the case of bidirectional transfers, data as well as ACK packets compete for resources in the upstream direction. In this case, a single first-in first-out (FIFO) queue for both data packets and ACKs could cause problems. For example, if the upstream channel is a 28.8 kb/s dialup line, the transmission of a 1-kbyte-sized data packet would take about 280 ms. So even if just two such data packets get queued ahead of ACKs (not an uncommon occurrence since data packets are sent out in pairs during slow start), they would shut out ACKs for well over half a second. And if more than two data packets are queued up ahead of

¹ We are only considering cumulative, not selective, ACKs.

an ACK, the ACKs would be delayed even more.

A possible approach to alleviating this problem is to schedule data and ACKs differently from FIFO. One algorithm, in particular, is *acks-first scheduling*, which always accords a higher priority to ACKs over data packets. The motivation for such scheduling is that it minimizes the idle time for the downstream connection by minimizing the amount of time its ACKs spend queued behind upstream data packets. At the same time, with techniques such as header compression [11], the transmission time of ACKs becomes small enough that its impact on subsequent data packets is minimal. (Networks in which the per-packet overhead of the upstream channel is large, e.g., packet radio networks, are an exception.)

Note that as with ACC, this scheduling scheme does not require the gateway to explicitly identify or maintain state for individual TCP connections.

Acks-first scheduling does not help avoid the delay due to a data packet in transmission. On a slow uplink, such a delay could be large if the data packet is large in size. One way to reduce the delay is to fragment the data packet into small pieces before transmission [10, 11].

HANDLING INFREQUENT ACKS

This can be done either end-to-end or locally at the constrained uplink.

TCP Sender Adaptation — ACC and AF alleviate the problem of congestion on the upstream bottleneck link by decreasing the frequency of ACKs, with each ACK potentially acknowledging several data packets. As discussed earlier, this can cause problems such as sender burstiness and a slowdown in congestion window growth.

TCP sender adaptation is an end-to-end technique for alleviating this problem. A bound is placed on the maximum number of packets the sender can transmit back to back, even if the window allows the transmission of more data. If necessary, more bursts of data are scheduled for later points in time computed based on the connection's data rate. The data rate is estimated as the ratio $cwnd/srtt$, where $cwnd$ is the TCP congestion window size and $srtt$ is the smoothed RTT estimate. Thus, large bursts of data get broken up into smaller bursts spread out over time.

The sender can avoid a slowdown in congestion window growth by simply taking into account the amount of data acknowledged by each ACK, rather than the number of ACKs. Thus, if an ACK acknowledges s segments, the window is grown as if s separate ACKs were received. (One could treat the single ACK as being equivalent to $s/2$ instead of s ACKs to mimic the effect of the TCP delayed-ACK algorithm.) This policy works because window growth is only tied to the available bandwidth in the downstream direction, so the number of ACKs is immaterial.

ACK Reconstruction — *ACK reconstruction (AR)* is a technique to reconstruct the ACK stream after it has traversed the upstream direction bottleneck link. AR is a local technique designed to prevent the reduced ACK frequency from adversely affecting the performance of standard TCP sender implementations (i.e., those

that do not implement sender adaptation). This enables us to use schemes such as AF or ACC without requiring TCP senders to be modified to perform sender adaptation. This solution can be easily deployed by Internet service providers (ISPs) of asymmetric access technologies in conjunction with AF to achieve good performance.

AR deploys a soft-state agent called the *ACK reconstructor* at the upstream end of the constrained ACK bottleneck. The reconstructor does not need to be on the downstream data path. It carefully fills in the gaps in the ACK sequence and introduces ACKs to smooth out the ACK stream seen by the sender. However, it does so without violating the end-to-end semantics of TCP ACKs, as explained below.

Suppose two ACKs, a_1 and a_2 , arrive at the reconstructor after traversing the constrained upstream link at times t_1 and t_2 respectively. Let $a_2 - a_1 = \delta_a > 1$. If a_2 were to reach the sender soon after a_1 with no intervening ACKs, at least δ_a segments are burst out by the sender (if the flow control window is large enough), and the congestion window increases by at most 1, independent of δ_a . ACK reconstruction remedies this problematic situation by interspersing ACKs to provide the sender with a larger number of ACKs at a consistent rate, which reduces the degree of burstiness and causes the congestion window to increase at a rate governed by the downstream bottleneck.

How is this done? One of the configurable parameters of the reconstructor is `ack_thresh`, the ACK threshold, which determines the spacing between interspersed ACKs at the output. Typically, `ack_thresh` is set to 2, which follows TCP's standard delayed-ACK policy. Thus, if successive ACKs arrive at the reconstructor separated by δ_a , it interposes $\text{ceil}(\delta_a / \text{ack_thresh}) - 2$ ACKs, where `ceil()` is the ceiling operator. The other parameter used by the reconstructor is `ack_interval`, which determines the temporal spacing between the reconstructed ACKs. To do this, it measures the rate at which ACKs arrive at the input to the reconstructor. This rate depends on the output rate from the constrained upstream channel and on the presence of other traffic on that link. The reconstructor uses an exponentially weighted moving average estimator to monitor this rate; the output of the estimator is δ_r , the average temporal spacing at which ACKs are arriving at the reconstructor (and the average rate at which ACKs would reach the sender if there were no further losses or delays). If the reconstructor sets `ack_interval` equal to δ_r , we would essentially operate at a rate governed by the upstream bottleneck link, and the resulting performance would be determined by the rate at which unfiltered ACKs arrive out of the upstream bottleneck link. If sender adaptation is done, the sender behaves as if the rate at which ACKs arrive is δ_a / δ_r . Therefore, a good method of deciding the temporal spacing of reconstructed ACKs, `ack_interval`, is to equate the rates at which increments in the ACK sequence happen in the two cases. That is, the reconstructor sets `ack_interval` such that $\delta_a / \delta_r = \text{ack_thresh} / \text{ack_interval}$, which implies that $\text{ack_interval} = (\text{ack_thresh} / \delta_a) * \delta_r$. Therefore, the latest ACK in the current sequence, a_2 , is

Acks-first scheduling does not help avoid the delay due to a data packet in transmission. On a slow uplink, such a delay could be large if the data packet is large in size. One way of reducing the delay is to fragment the data packet into small pieces before transmission.

Metric	Reno	ACC/SA	AF/SA	AF/AR	AF alone
Throughput (Mb/s)	6.71	6.95	7.82	8.57	5.16
Average cwnd (pkts)	66.7	62	65.3	104.6	43.8
Average rtt (ms)	79	70	65	97	65

Table 1. Performance of different protocols in the presence of losses in the downstream direction. The highlighted fields show the benefits of SA and AR, and demonstrate that AF alone is not enough.

held back for a time roughly equal to δ_t , and $\text{ceil}(\delta_a/\text{ack_thresh}) - 2$ ACKs are evenly interposed in this time.

Thus, by carefully controlling the number of and spacing between ACKs, unmodified TCP senders can be made to increase their congestion window at the right rate and avoid bursty behavior. ACK reconstruction can be implemented by maintaining only soft state at the reconstructor that can easily be regenerated if lost. Note that the reconstructor generates no spurious ACKs, and the end-to-end semantics of the connection are completely preserved. The trade-off in AR is between obtaining less bursty performance, a better rate of congestion window increase, and a reduction in the round-trip variation, vs. a modest increase in the round-trip time estimate at the sender. We believe it is a good trade-off in the asymmetric environments we are concerned with.

EXPERIMENTAL RESULTS

We briefly discuss experimental results. For details refer to [1, 2]. TCP Reno (Reno) is the base protocol. Reno is enhanced with ACC, AF, sender adaptation (SA), and ACK reconstruction (AR), in various combinations.

BANDWIDTH ASYMMETRY

In this setting we have a 10 Mb/s downlink and a 28.8 kb/s uplink with TCP header compression enabled. The maximum window size was set to 120 packets, and all queue sizes were set to 10 packets. Table 1 shows the performance of a sin-

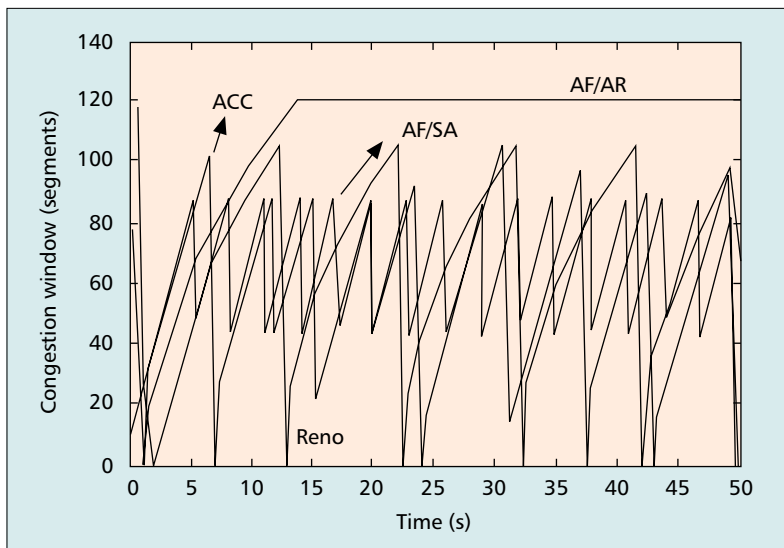


Figure 6. Congestion window evolution for the different schemes. ACC, AF/SA, and AF/AR do not fluctuate as much as Reno, achieving better performance.

gle downstream transfer.

AF/AR and AF/SA perform the best, achieving throughputs between 15 and 21 percent better than Reno. ACC/SA performs about 5 percent better than Reno for this configuration. The important point to note is that the degree of burstiness is reduced significantly, while the upstream router queue is no longer perpetually full because of AF or ACC. This can be seen from Fig. 6, which shows the time evolution of congestion windows for the different protocols. Table 1 shows the time-averaged TCP congestion window and round-trip times for the different protocols. It is clear from the table that reducing the frequency of ACKs alone is insufficient, and that techniques like SA or AR need to be used as well.

We note that AR results in a larger round-trip time than the other protocols, but this leads to the best performance for this setting of parameters because it reduces the number of losses (since packet transmissions from the source are now spread over a longer duration). For ACC/SA we use a RED GW to mark packets (ACKs) and drop packets when the queue is full. We found that using a random drop policy is superior to dropping from the tail when an ACK has to be dropped. This is because tail drops sometimes lead to long, consecutive sequences of ACKs being dropped, and thus increased sender burstiness.

In summary, our results show that SA or AR is important to overcome the burstiness that results from a lossy ACK stream, and that a random drop policy at the RED gateway was better for performance.

MEDIA ACCESS ASYMMETRY

We used a model of the Metricom Ricochet network. The workload in these experiments consists of a 100-s TCP transfer, with no other competing traffic and a maximum receiver window size of 32 kbytes. Congestion losses occur as a result of buffer overflow, and lead to sender timeouts if multiple packets are lost in a transmission window. The protocols investigated include unmodified TCP Reno, Reno with ACC/SA (i.e., ACC and SA at the sender), and Reno with AF/SA (i.e., AF with SA at the sender).

Figure 7 shows the results of these experiments as a function of the number of wireless hops. The performance of AF and ACC with SA are better than Reno, and AF/SA is better than ACC/SA. The degree of improvement in throughput varies from 25 percent (one wireless hop) to 41 percent (three wireless hops).

RELATED WORK

Lakshman, Madhow, and Suter [7] analyzed the problems due to bandwidth asymmetry, and showed that performance degraded if the normalized asymmetry ratio exceeded one. They also propose the use of the drop-from-front strategy on ACKs in the constrained upstream channel buffer as a way to alleviate the problems that arise.

Kalampoukas, Varma, and Ramakrishnan [12] study the problems that arise when bidirectional traffic is present in a bandwidth-asymmetric network, using a combination of analysis and simulation. They show that ACK compression sometimes occurs in these environments, causing

throughput reduction. To alleviate the effects on throughput, they investigate three approaches:

- Prioritizing ACKs over data on the upstream channel, which results in starvation
- Using a connection-level backpressure mechanism to limit the maximum amount of data buffered in the upstream channel router, which results in unfair bandwidth utilization
- Using connection-level bandwidth allocation, which provides flow isolation

Cohen and Ramanathan [13] study TCP performance over a hybrid coaxial cable distribution system using simulations of that network. They conclude that it is important for receivers to tune their socket buffer sizes, use an initial window of two segments as opposed to one, use smaller segment sizes to overcome problems with delayed TCP ACKs and ACK losses, use a finer granularity for TCP timers, and use fast retransmission triggered by a single duplicate ACK for better loss recovery. In general, while some of these techniques improve performance in their network, they are not universally applicable and lead to worse performance in other networks. For example, retransmitting a segment upon the arrival of the first duplicate ACK is not correct in general because of the significant amount of packet reordering in the Internet. The schemes in this article, in contrast, improve TCP performance in heterogeneous wireless networks without compromising performance in other situations.

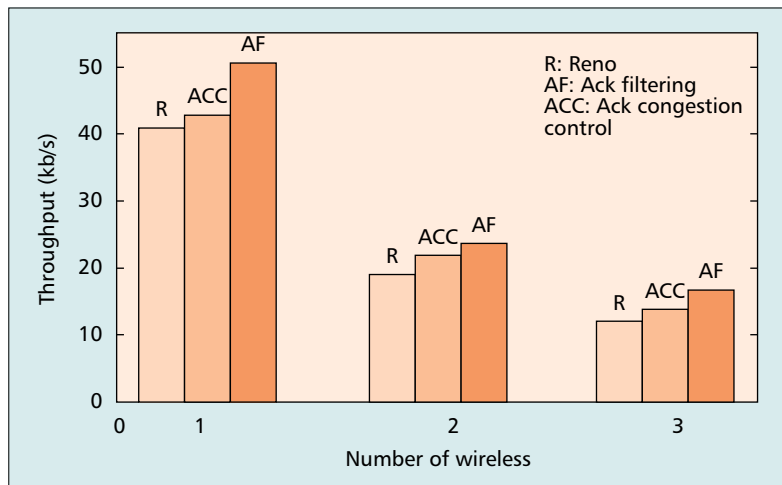
Samaraweera presents ACK compaction and expansion that are similar in vein to, but more complicated than, AF and AR [14]. The compacter discards older ACKs in the upstream queue while retaining newer ACKs (just as in AF), but in addition conveys the number of discarded ACKs and the total number of bytes they acknowledge to its peer, the expander. The expander can then regenerate the discarded ACKs without having to guess how many ACKs were discarded. This is an advantage Metric Reno ACC compared to AF/AR. However, it comes at the cost of new protocol machinery to convey the information about discarded ACKs from the compacter to the expander. AF/AR does not require any new protocol machinery.

CONCLUSION

In this article we describe how network asymmetry in terms of bandwidth and media access can affect the performance of a feedback-based transport protocol such as TCP. After analyzing these problems, we present and evaluate several solutions to alleviate the situation. Our results show that these techniques provide network designers with solutions for efficient TCP in a variety of asymmetric conditions.

REFERENCES

- [1] H. Balakrishnan, "Challenges to Reliable Data Transport over Heterogeneous Wireless Networks," Ph.D. thesis, UC Berkeley, Aug. 1998.
- [2] V. N. Padmanabhan, "Addressing the Challenges of Web Data Transport," Ph.D. thesis, UC Berkeley, Sept. 1998.
- [3] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The Effects of Asymmetry on TCP Performance," *Proc. ACM MOBICOM '97*, Sept. 1997.
- [4] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The Effects of Asymmetry on TCP Performance," *ACM MONET*, 1999.



■ **Figure 7.** TCP throughputs from simulations of Reno, ACC, and AF, as a function of the number of wireless hops.

- [5] DirecPC Web page: <http://www.direcpc.com>
- [6] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM '88*, Aug. 1988.
- [7] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A Study of TCP/IP Performance," *Proc. IEEE INFOCOM*, Kobe, Japan, Apr. 1997.
- [8] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, Feb. 1990.
- [9] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Net.*, vol. 1, no. 4, Aug. 1993, pp. 397-413.
- [10] K. Sklower et al., "The PPP Multilink Protocol (MP)," RFC-1990, Aug. 1996.
- [11] C. Bormann, "The Multi-Class Extension to Multi-Link PPP," RFC 2686, Sept. 1999.
- [12] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions," *Proc. ACM SIGMETRICS*, June 1998.
- [13] R. Cohen, S. Ramanathan, "TCP for High Performance in Hybrid Fiber Coaxial BroadBand Access Networks," *IEEE/ACM Trans. Net.*, Feb. 1998.
- [14] N. K. G. Samaraweera, "Return Link Optimization for Internet Service Provision Using DVB-S Networks," *ACM SIGCOMM CCR*, July 1999.

BIOGRAPHIES

HARI BALAKRISHNAN (hari@lcs.mit.edu) is the KDD Career Development assistant professor of communications technology in the EECS Department and the Laboratory for Computer Science (LCS) at MIT. He leads the Networks and Mobile Systems group at LCS, which conducts research in wireless and mobile systems, network protocols and architecture, and pervasive computing. He received his Ph.D. from the University of California at Berkeley (UC Berkeley) in 1998, for a dissertation on wireless transport that won an ACM doctoral dissertation award; <http://nms.lcs.mit.edu>

VENKATA N. PADMANABHAN (padmanab@microsoft.com) [ACM'94, IEEE'94] is a researcher at Microsoft Research where his work focuses on location-aware computing, Web dynamics, and Internet performance. He has also conducted research on Web data transport and emerging access network technologies. His work on persistent-connection HTTP was adopted by the HTTP/1.1 standard and resulted in influential papers. He holds a B.Tech. from IIT Delhi (1993) and an M.S. and a Ph.D. from UC Berkeley (1995 and 1998); <http://www.research.microsoft.com/~padmanab>