9. Acknowledgments

We are grateful to several people for their help in setting up and debugging our various experimental testbeds: Mike Ritter, Mike Cunningham, Bob Luxemburg, Sheela Rayala, and Will SanFilippo (Metricom, Inc.); Ed Moura (Hybrid Networks, Inc.); Andrew Nestor (MetroNet, Inc.); and Ken Lutz, Steve Hawes, and Fred Archibald (UC Berkeley). We appreciate their time and assistance.

We thank Mary Baker, Sally Floyd, Tom Henderson, Mike Ritter, Srinivasan Seshan, and the anonymous Mobicom referees for several comments and suggestions that helped improve the quality of this paper. Our special thanks to Giao Nguyen for his contributions to ns, which greatly facilitated our work.

This work was supported by DARPA contract DAAB07-95-C-D154, by the State of California under the MICRO program, and by the Hughes Aircraft Corporation, Metricom, Fuji Xerox, Daimler-Benz, Hybrid Networks, and IBM. Hari is partially supported by a research grant from the Okawa Foundation.

10. References

[1] A. Bakre and B. R. Badrinath. Handoff and System Support for Indirect TCP/IP. In *Proc. Second Usenix Symp. on Mobile and Location-Independent Computing*, April 1995.

[2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *Proc. ACM SIGCOMM '96*, August 1996.

[3] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), December 1995.

[4] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[5] S. Cheshire and M. Baker. A Wireless Network in MosquitoNet. *IEEE Micro*, Feb 1996.

[6] D-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[7] R. Durst, G. Miller, and E. Travis. TCP Extensions for Space Communications. In *Proc. ACM Mobicom Conference*, November 1996.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC, Jan 1997. RFC-2068.

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM 88*, August 1988.

[11] V. Jacobson. *Compressing TCP/IP Headers for Low-speed Serial Links*. RFC-1144, Feb 1990.

[12] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991.

[13] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Performance of Two-Way TCP Traffic over Asymmetric Access Links. In *Proc. Interop '97 Engineers' Conference*, May 1997.

[14] P. Karn. MACA – A New Channel Access Method for Packet Radio. In *Proc. 9th ARRL Computer Networking Conference*, 1990.

[15] P. Karn. Dropping TCP acks. Mail to the end-to-end mailing list, February 1996.

[16] 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. In *Proc. Infocom 97*, April 1997.

[17] ns – Network Simulator. http://www-mash.cs.berkeley.edu/ns/, 1996.

[18] V. N. Padmanabhan, H. Balakrishnan, K. Sklower, E. Amir, and R. Katz. Networking Using Direct Broadcast Satellite. In *Proc. Workshop on Satellite-Based Information Systems*, November 1996.

[19] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

[20] L. Zhang, S. Shenker, and D. D. Clark. Observations and Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proc. ACM SIGCOMM '91*, pages 133–147, 1991.

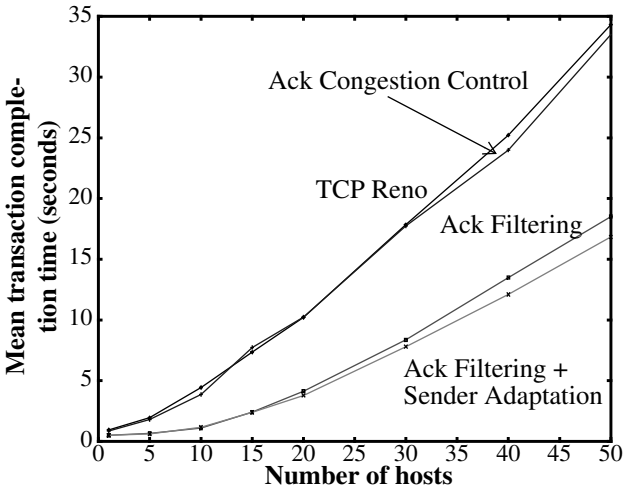

Figure 14. Simulated scaling behavior of *TCP Reno*, *AF*, and *ACC* as a function of the number of connections for a Web-like microbenchmark over a packet radio return path. The graph shows average completion time vs. *n*, the number of hosts (4 concurrent connections per host). The ack filtering protocol performs the best as *n* increases.

rather to understand the effects of small and concurrent connections, as well as competing and interacting users, on performance. Since latency is a critical factor that impacts performance in the packet radio network, we implicitly assume that the Web requests use pipelining [19] and that one 500 byte request results in 4 downloads.

We vary the number of hosts from 1 to 50 in the simulations. Hosts make requests independent of each other at a time uniformly distributed in [0,5] seconds. We measure the mean and standard deviation of the completion time for the entire Web transaction (i.e., all 4 connections).

.Figure 14 shows the mean completion time for a transaction as the number of hosts varies, for TCP Reno, Reno with ACC (Section 4.3.1) and Reno with AF (Section 4.3.2). Two curves are shown for the AF case — with sender adaptation enabled based on the round-trip time and the congestion window, and without (Section 4.3.3). These results indicate that AF is very beneficial in reducing the response time and improving the throughput of the network as the system scales. The reason ACC is not as beneficial as in the cases investigated in Section 4 is the shorter transfer lengths in this benchmark. No connection exceeds 10 packets, so the sender's window is never very large[3]. This limits the extent to which ACC can be performed. The other reason is the larger number of acks traversing the network with ACC, compared to AF. One critical factor in this network is the latency and variability associated with each packet on the wireless network. AF results in significant gains because it purges all redundant acks from the queue independent of the state of congestion, thereby reducing the number of packets in the wireless cloud. Finally, we note that the lack of sender adaptation does not significantly hurt performance, since each connection is rather small. The maximum possible burst in the network is automatically limited by this short length and the slow start process starting from 1 segment.

3. With persistent-connection HTTP [19], recommended by HTTP/1.1 [8], connections will tend to be longer. This should help ACC.

7. Conclusions

In this paper, we investigated the effects on network asymmetry on TCP performance in the context of wide-area wireless networks. We studied the impact of the reverse path, used primarily for acknowledgments and data requests, on end-to-end performance in the forward direction. We distinguished between bandwidth asymmetry, latency and media-access asymmetry, and loss asymmetry, and focused on the first two types.

The following are our main results:

- SLIP header compression [11] alleviates some of the problems due to bandwidth asymmetry, but does not completely eliminate all problems, especially those that arise in the presence of bidirectional traffic.
- Connections traversing packet radio networks suffer from large variations in round-trip times caused by the half-duplex nature of the radios and asymmetries in the media-access protocol, which lead to variable latencies. This adversely affects TCP's loss recovery mechanism and results in degraded performance.
- The various end-to-end and router-based techniques that we propose help improve performance significantly in many asymmetric situations. These include decreasing the frequency of acknowledgments (acks) on the constrained reverse channel (*ack congestion control* and *ack filtering*), reducing source burstiness when acknowledgments are infrequent (*TCP sender adaptation),* and scheduling data and acks intelligently at the reverse bottleneck router.
- In addition to improving throughput for individual connections, our proposed modifications also help improve the fairness and scaling properties when several connections contend for scarce resources in the network. We demonstrate this via simulations of bulk and Web-like transfers.

8. Future Work

There are several areas of future work that we plan to investigate.

- We plan to investigate *ack reconstruction*, as a complement to ack filtering and alternative to sender adaptation, to shield the end-hosts from disruptions in the ack stream. The main idea is for the reconstructor to smooth out the filtered ack stream by inserting new acks to fill in the gaps, after it has traversed the constrained reverse channel. The ack reconstructor would be located at the other end of the constrained reverse channel.
- The asymmetry in loss and error rates, especially in the context of wireless return channels, poses new challenges. Current packet radio networks use link-layer protocols for local error recovery, but this results in increased latency and variability in latency. We plan to extend Explicit Loss Notification (ELN) schemes proposed in the context of single-hop cellular networks [2] to multi-hop wireless networks, with the goal of reducing variability without sacrificing local error recovery.
- Cellular Digital Packet Data (CDPD) networks exhibit media-access asymmetry. Communication from the base station to the end stations is unimpeded, but end stations contend with each other for channel access in the reverse direction. It will be informative and useful to study the impact of this asymmetry on reliable transport performance.
- Finally, we intend to implement and measure the promising techniques over the asymmetric, wide-area wireless networks in our testbed and measure their performance.

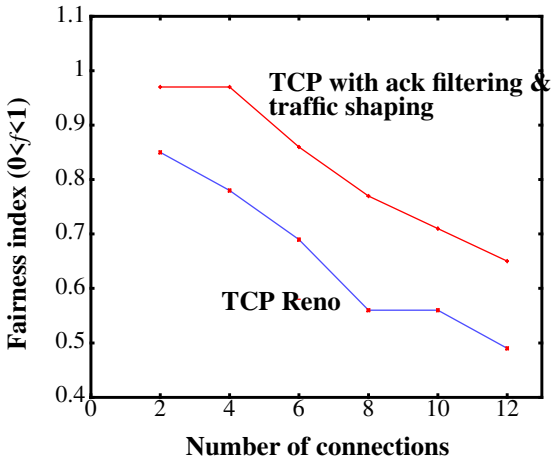

Figure 12. Simulation results for the fairness index of TCP Reno and TCP with AF, as a function of the number of connections traversing one hop of the packet radio network.

ing for resources. For simplicity, we consider connections over one wireless hop in this section.

We consider two important metrics while studying the impact of increasing the number of connections in the network: *utilization* and *fairness*. Network utilization is defined as the ratio of the aggregate throughput of all connections to the maximum achievable throughput of the network. Fairness is quantified using the fairness index defined in Section 3.2. We are interested in having large values of both the network utilization as well as the fairness index for any configuration.

Table 5 shows the simulated aggregate throughput achieved by all the connections in the network, as a function of the number of competing connections for Reno and AF. The table also shows the standard deviation of the resulting performance, which shows the degree of variation in the throughputs seen by the different concurrent connections. The aggregate performance varies between 44

Connections	Reno throughput (Kbps) [std-dev]	Reno+AF (Kbps) [std-dev]
1	44.69 [-]	52.17 [-]
2	42.8 [13.1]	51.8 [10.0]
4	45.2 [24.9]	49.3 [8.9]
6	47.1 [32.5]	49.3 [20.2]
8	45.0 [37.6]	49.6 [28.0]
10	45.6 [42.2]	48.4 [32.4]
12	45.8 [48.0]	48.8 [36.4]

Table 5. Throughputs of TCP Reno and Reno with AF as a function of the number of connections (1 wireless hop).

and 47 Kbps for TCP Reno and between 48 and 52 Kbps with AF, as the number of connections varies, implying that there is little change in utilization as a function of the number of connections. However, the standard deviation of the performance, calculated over concurrent connections, is much higher for Reno than for ack-filtered Reno. This suggests that the distribution of throughputs is more spread out and less equal across any set of connections, especially as their number increases.

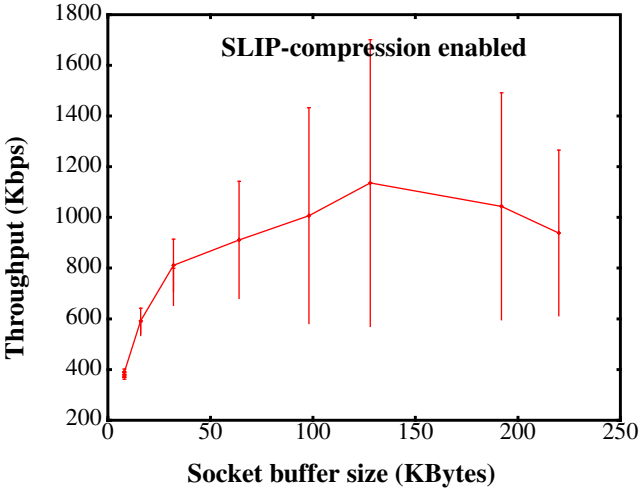

Figure 13. Measured performance of a 1 MByte TCP transfer across a Hybrid wireless cable downlink and Ricochet return channel. The large errorbars are a consequence of the inherent variability of the Ricochet return channel.

We quantify this effect further in Figure 12, where the fairness index of throughput is plotted as a function of the number of connections. TCP Reno is often grossly unfair in the distribution of throughput, reaching a value as low as 0.49 (n=12) and not exceeding 0.85 (n=2). In contrast, AF substantially improves the fairness index of the throughput distribution, while also improving the overall utilization of the network.

Thus, we see that our enhancements to TCP and the reverse bottleneck router help improve overall utilization and fairness as the number of connections increases.

6. Combining Wireless Technologies: Wireless Cable and Packet Radio

In this section, we investigate the effects of combining different types of asymmetry on TCP performance. We focus on a network topology with a high-bandwidth forward path modeled after Hybrid's wireless cable channel, and a low-bandwidth, packet radio reverse path modeled after the Ricochet network. Such composite network topologies are relevant in several application scenarios. For example, a disaster relief vehicle or ambulance with a unidirectional high-bandwidth link would use a wide-area wireless network as its reverse channel.

Figure 13 shows the measured performance of 1MB TCP transfers as a function of the receiver socket buffer size using a Hybrid Networks' wireless cable forward path and a one-hop wireless Ricochet return path. The error-bars show the standard deviation of measured performance (20 runs each). These controlled measurements were performed in the absence of any cross-traffic and the inherent variability of the return path manifests itself in the significant error-bars on the graph.

We focus on a Web-like benchmark in the following simulations and study the performance of this network as the number of hosts and connections increases. We investigate the various modifications to the transport and router protocols to help reduce the average completion time of a Web request in such networks.

We model the Web microbenchmark as a 500 byte Web request followed by set of 4 concurrent TCP transfers of 10 KB each to the client. This is not intended to be an accurate model of reality, but

and added support for arbitrary MAC and link-layer protocols. The simulation parameters used to obtain the results described in Section 5.3 are shown in Table 4. We do not consider the impact of wireless bit errors in these simulations, to isolate the impact of variability due to the MAC protocol on performance. In practice, link-layer retransmissions of corrupted packets will only add to the variability of the network.

Parameter	Value
Link bandwidth	100 Kbps
Fixed link latency	10 ms
T_{TR}	11.125 ms
T_{RT}	13.25 ms
Radio queue size	10 packets
Receiver buffer size	32 KBytes

Table 4. Simulation parameters of the multi-hop packet radio network. The number of wireless hops varies between 1 and 3.

5.3 Analysis and Solutions

In this section, we perform a detailed analysis of the problems caused by this type of asymmetry and present some solutions that alleviate the adverse effects of increased variability.

5.3.1 Piggybacking Link-Layer Acks with Data

This scheme is motivated by the observation that the radios turn-around both for data frames as well as for link-layer acks. The presence of traffic in both directions, even when caused by TCP acknowledgments, already causes turnarounds to happen. Thus, link-layer acks can be piggybacked with data frames, thereby avoiding some extra radio turnarounds.

The basic reliable link-layer protocols in several systems do not piggyback acks with data. However, recent releases of the radio software in the Ricochet network attempt to do this whenever possible. Our simulations of the multi-hop wireless network assume that the radio units piggyback link-layer acks with data.

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.

5.3.2 Ack Filtering and Ack Congestion Control

We now analyze this system in more detail and present the results of two improved protocols — one that performs AF (Section 4.3.2) at the entry (from the receiver's side) to the packet radio network, coupled with TCP sender adaptation (Section 4.3.3), and the other that performs ACC using RED (Section 4.3.1) — to reduce the effects of variability and improve performance. Figure 10 shows these results. The performance of AF and ACC are better than Reno, and AF is better than ACC. The reason for this are the reduced number of packets and reduced round-trip variability of the two improvements compared to Reno.

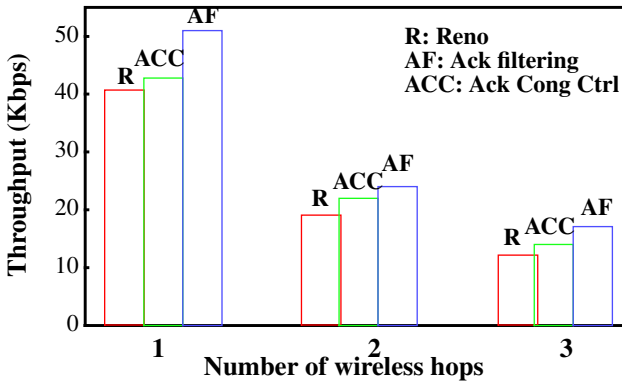

Figure 10. TCP throughputs from simulations of Reno, ACC and AF, as a function of the number of wireless hops.

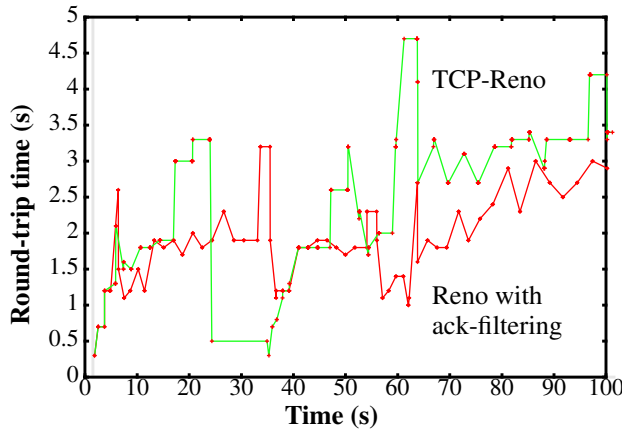

Figure 11. Round-trip times obtained from simulations of TCP Reno and TCP with AF and traffic shaping for a connection over two packet radio hops. The round-trip times for the Reno connection are much more variable, with a mean of 2.67 secs and a standard deviation of 1 sec, whereas AF reduces the mean to 1.85 secs and the standard deviation to 0.6 secs.

Figure 11 shows the round-trip times of simulations of a TCP Reno connection and a TCP connection with AF, over two wireless hops (in a chain-like topology between sender and receiver). For Reno, the mean round-trip time is about 2.67 seconds and the standard deviation is about 1 second. AF reduces the mean round-trip time to 1.85 seconds and the corresponding standard deviation to only 0.6 seconds. The number of packets traversing each node also drops, reducing the amount of contention. These factors result in a 25% improvement in end-to-end throughput, from 19 Kbps (Reno) to 24 Kbps over 2 wireless hops. Similar improvement in performance is seen for connections traversing three wireless hops — end-to-end throughput improves on average from 12.1 Kbps (Reno) to 17.0 Kbps (AF), an improvement of 40%. Finally, we note that AF outperforms ACC because the former completely eliminates all redundant acks and reduces the amount of "interfering" traffic to a greater extent.

5.3.3 Scaling and Fairness Issues

We now investigate the scaling properties of two protocols in the packet radio network — TCP Reno and Reno enhanced with AF. We are interested in this to understand performance as a function of the number of connections traversing the network and compet-

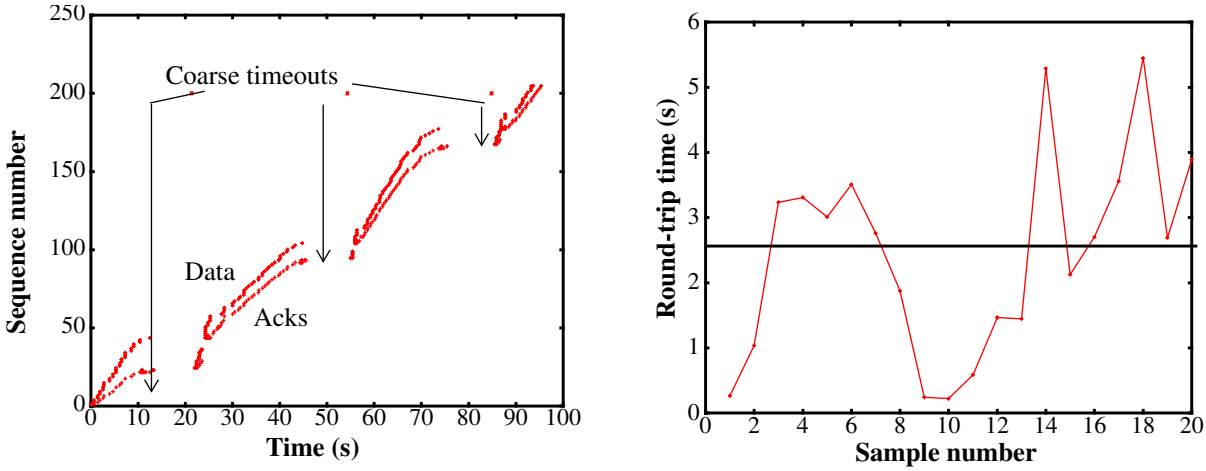

Figure 9. (a) Packet and ack sequence trace of a 200 KB TCP bulk transfer measured over one wireless hop in the Ricochet network. The three pauses are sender timeouts, lasting between 9 and 12 seconds each because large round-trip time variations cause the retransmission timeout estimate to be very long.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

less cloud is 100 Kbps. Packets from a fixed host (FH) on the Internet are routed via the Metricom Gateway (GW) and through the poletop radios (PT), to the end mobile host (MH). The number of wireless hops is typically between 1 and 3.

Radio Turnarounds: The radio units in the network are *half-duplex*, which means that they cannot simultaneously transmit and receive data. Moving from transmitting to receiving mode takes a non-trivial 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}.

MAC Protocol: The radios are frequency-hopping, spread-spectrum units operating in the 915 MHz ISM band. The details of the frequency-hopping protocol are not relevant to this paper, since the predominant reason for variability is the MAC protocol. The MAC protocol is based on a polling scheme, similar to (but not identical to) the RTS/CTS ("Request-To-Send/Clear-To-Send") 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, 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 backoff for a random amount of time and schedule the transmission for later. It could also send a NACK-CTS to the sender, which achieves the same effect. In all 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 neighbors in the network.

Link-Layer Protocol: The reliable link-layer protocol used in this network 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 after a timeout. Such simple link-layer protocols are the norm in several packet radio networks (see, e.g., [14]).

Variable Delays: 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. Further, 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), the overhead is variable. This is the main reason for large and variable latency in packet-radio networks. It is also clear why an increase in "interfering" traffic (like TCP acks) can significantly impact the flow of TCP data packets.

5.2 Measurements

We now discuss the results of several measurements and simulations under various network topologies and traffic workloads. We start with the simplest case of a bulk TCP transfer across one wireless hop running the MAC and link-layer protocols described above. Although these particular measurements were made using the Phase 1 Ricochet modems, we have observed similar effects in measurements made with the newer Phase 2 modems as well.

Figure 9 shows the packet sequence trace of a measured 200 KB TCP transfer across one wired and one wireless hop in the Ricochet network. This clearly shows the effect of the radio turnarounds and increased variability affecting performance. The connection is idle for 35% its duration, as a result of only three coarse timeouts (six other losses are recovered by TCP's fast retransmission mechanism). Ideally, the round-trip time of a data transfer will be relatively constant (i.e., have a low deviation). Unfortunately, this is not true for connections in this network, as shown in Figure 11. This figure plots the individual round-trip time estimate samples during a TCP connection over the actual Ricochet network. The mean value of these samples is about 2.5 seconds and the standard deviation is about 1.5 seconds. Because of the high variation in the individual samples, the retransmission timer, set to *srtt + 4*mdev,* is on the order of 10 seconds, causing long idle periods. In general, it is correct for the retransmission timer to trigger a segment retransmission only after an amount of time dependent on both the round-trip time and the linear (or standard) deviation, since this avoids spurious retransmissions. Thus, techniques are needed to alleviate the problems caused by large deviations in TCP round-trip times to the loss recovery process. These problems are exacerbated in the presence of two-way traffic as well as other competing traffic.

Based on several experimental measurements of the Ricochet network, we modeled the system in the ns simulator. We extended the point-to-point link abstraction of ns to a more general shared LAN

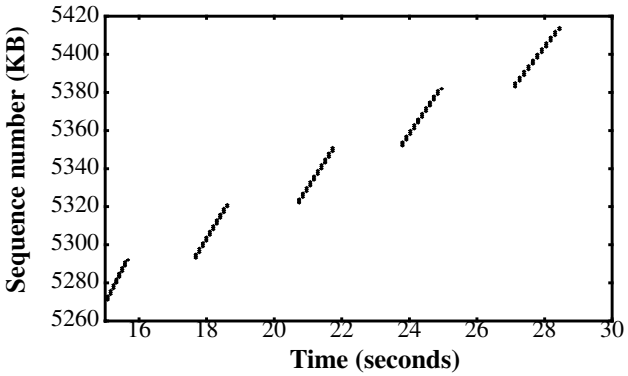

Figure 6. Simulation results showing a portion of the sequence number trace for the forward transfer after the reverse transfer has started up. The reverse channel router uses ack filtering. The multi-second idle times are caused by acks getting queued behind multiple 1 KB data packets belonging to the reverse transfer.

With ACC (and the reverse channel router employing the RED algorithm), the throughput of the reverse transfer is quite high (24.22 Kbps as against the maximum possible of 28.8 Kbps). At the same time, the throughput of the forward transfer (1.74 Mbps) is much better than before. The reason for the better performance is that feedback from the RED gateway prevents the reverse transfer from filling up the reverse gateway with its data packets. The reverse connection can sustain optimal throughput without having to grow its window to more than 1-2 packets. (Even assuming a rather large RTT of 500 ms for the reverse connection, the bandwidth-delay product is 28.8 Kbps * 500 ms = 1.8 KB which is less than two 1 KB packets.) Thus, the reverse connection can decrease the impact that its data packets have on ack packets of the forward transfer, while sustaining optimal throughput.

Even with the RED algorithm in operation, ack packets could get queued behind more than one data packet, which decreases forward throughput. The acks-first scheduling scheme (Section 4.3.4) avoids this by prioritizing acks over data. The assumption is that such scheduling will not add significantly to the queuing delay of data packets. With ACC (which decreases the frequency of acks) and header compression (which makes them small in size), data packets are indeed not affected significantly. As shown in Table 3, ACC with acks-first scheduling achieves a forward throughput of 2.67 Mbps while maintaining a close-to-optimal reverse throughput (27.17 Kbps).

A simple calculation shows that with the parameters we have chosen, we cannot do much better than 2.85 Mbps while maintaining optimal reverse throughput. While a data packet of the reverse connection is undergoing transmission on the 28.8 Kbps link (lasting 280 ms), the forward connection sender does not receive any new acks. Figure 7 illustrates this effect through simulation. So it can send at most one window's worth of data in 280 ms. With the socket buffer size of 100 KB that we have chosen, the maximum sender throughput works out to 100*8/280 = 2.85 Mbps.

In contrast to ACC, combining acks-first scheduling with AF leads to starvation of data packets of the reverse transfer. This is because ack packets arrive at the queue at a faster rate than they can be drained out, so there is always an ack waiting to be sent in the queue. Note that an ack undergoing transmission is no longer in the queue, and so is not considered by the ack filtering algorithm.

Finally, to point out the benefits of using RED feedback to do ACC, we consider the case where feedback from the RED gateway

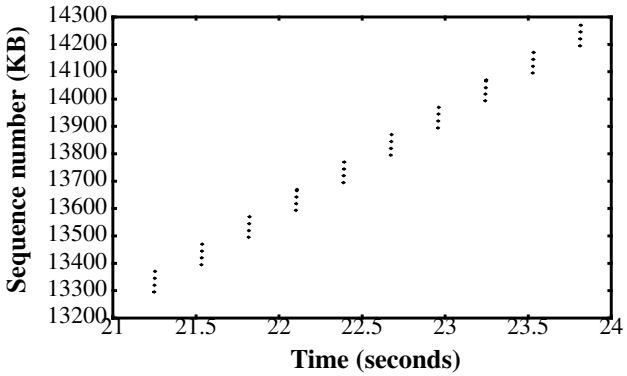

Figure 7. Simulation results showing a portion of the ack trace for the forward transfer after the reverse transfer has started up. ACC is used in conjunction with acks-first scheduling. There is an idle time of about 280 ms between bursts of acks because of the 1 KB data packets belonging to the reverse transfer.

The forward throughput (more than 3 Mbps) is higher than before, but the reverse throughput is only 17.8 Kbps. Since acks are not subject to congestion control like data, they cause the reverse connection to lose packets and time out periodically. During these idle periods of the reverse connection, the forward transfer makes rapid progress, resulting in a higher forward throughput than before.

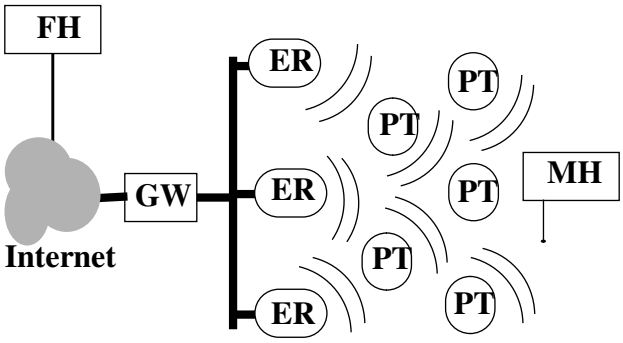

Figure 8. Topology of the Ricochet packet radio network. The Mobile Host (MH) has a modem attached to it, which communicates with a Fixed Host (FH) on the Internet through the Poletop Radios (PT) and Ethernet Radios (ER). The Gateway (GW) routes packets between the packet radio network and the Internet.

5. Latency and Media-Access Asymmetry

In this section, we discuss the effects of latency and media-access asymmetry on TCP performance. As before, we use a combination of measurements and simulations to obtain our results. We focus on TCP connections through a packet radio network as an example of a situation with this type of asymmetry. We start by describing the topology of the network and the media-access and link-layer protocols. We then discuss the results of our experiments and solutions to observed problems. Finally, we discuss some scaling and fairness issues in this network.

5.1 Network Topology and Underlying Protocols

Topology: The topology of the packet radio network is shown in Figure 8. The maximum link speed between two nodes in the wire-

Reverse Channel Bandwidth	Reno	Reno+ACC	Reno+AF
9.6 Kbps	1.78	3.64	6.28
9.6 Kbps C	6.67	7.69	7.93
28.8 Kbps	4.35	7.58	9.49
28.8 Kbps C	9.78	9.77	9.88

Table 1. Throughputs (in Mbps) from the simulation of a single one-way transfer in the forward direction. "C" indicates the use of SLIP header compression.

The factor k (the normalized asymmetry ratio) exceeds 10 for the cases of SLIP without header compression, which explains the poor throughput of TCP Reno in those cases (1.78 and 4.35 Mbps). The sender adaptation employed in conjunction with ACC and AF breaks up potential bursts, avoiding performance degradation in those cases.

For the 9.6 Kbps reverse channel with header compression, k is 6.25, which is less than 10. Still, the throughput obtained with TCP Reno (6.67 Mbps) is worse than that for the other schemes. This happens because the reverse channel buffer gets filled with acks (totalling 10*6 = 60 bytes), which adds a significant delay (60*8/9.6 = 50 ms) to the connections round-trip time (RTT). The same effect also explains why the performance with ACC is somewhat worse than that with AF for both the 9.6 Kbps and 28.8 Kbps cases. The former only tries to ensure that the reverse channel queue does not get completely filled up. The latter ensures that there is not more than one ack per connection in the queue, which minimizes the effect of queuing on the round-trip time.

To summarize, TCP Reno suffers performance degradation when k is large and there is significant queuing delay. ACC and AF alleviate these problems by decreasing the frequency of acks.

4.4.2 Two Simultaneous One-way Transfers

We now consider two simultaneous one-way transfers with the same topology as in Section 4.4.1 and the reverse channel fixed to be a 28.8 Kbps dialup line with header compression. The first transfer is initiated at time 0 and continues for 50 seconds. The second transfer starts at a randomly picked time between 5 and 10 seconds and ends at time equal to 50 seconds. Ten runs were conducted for each configuration. The goal here is to see how the two connections share the reverse channel bandwidth and buffer, which impacts the throughput of each.

Table 2 summarizes the results obtained in terms of the aggregate

Metric	Reno	Reno+ACC	Reno+AF
Total throughput	9.80	8.59	8.98
Fairness index	0.5	0.95	0.99

Table 2. The aggregate throughput (in Mbps) and the fairness index based on the simulation of two one-way transfers in the forward direction. The reverse channel is a 28.8 Kbps dialup line with header compression.

throughput for the two connections and the fairness index (as defined in Section 3.2) computed over the period during which both connections are active. We see that unmodified TCP Reno yields the best aggregate throughput but has a much worse fairness index value than the others.

The high degree of unfairness with TCP Reno arises because the acks of the first connection quickly fill up the reverse channel buffer. So, when the second connection starts up, it suffers ack losses early on, leading to timeouts and hence a lack of progress. Even if all acks of the second connection were not lost, the growth of its window during the slow start phase would be slowed down because of the large queuing delay that its acks would encounter.

By decreasing the frequency of acks, ACC and AF keep the reverse channel queue small, so that the new connection does not face problems such as the ones that happen with unmodified TCP Reno. Consequently, the fairness indices in these cases are close to the maximum value of 1.

4.4.3 Two-way Transfers

Next we consider two simultaneous transfers, one each in the forward and the reverse directions. Again we fix the reverse channel to be a 28.8 Kbps dialup line with header compression. The forward transfer is initiated at time 0. The reverse transfer is initiated at a randomly picked time between 5 and 10 seconds. Both transfers continue until time equal to 50 seconds. Table 3 summarizes

Protocol	Forward Throughput	Reverse Throughput
TCP Reno	9800.00	0.00
ACC	1740.00	24.22
ACC + acks-first	2670.00	27.17
AF	135.80	28.70
AF + acks-first	8000.06	0.00
RED only for data	3032.53	17.84

Table 3. The throughput (in Kbps) from the simulation of simultaneous forward and reverse transfers. The reverse channel is a header-compressed 28.8 Kbps dialup line.

the results.

We make several interesting observations. With unmodified TCP Reno, acks of the forward connection fill up the reverse channel buffer, thereby completely shutting out reverse transfer that starts later. However, if the reverse connection were to start before the forward connection, the situation is very different, with the reverse connection achieving close to optimal throughput at the cost of the forward connection (this data not shown in Table 3). The reason for this entirely different behavior will become clear in our discussion below of performance with ack filtering.

AF achieves very poor throughput for the forward transfer but close to optimal throughput for the reverse transfer. The reason this happens is that when the reverse transfer starts up, 1 KB sized data packets start entering the reverse channel queue. The transmission delay of each data packet over the 28.8 Kbps line is 280 ms. Because of FIFO scheduling, acks of the forward transfer get queued behind these data packets for this entire duration, causing the sender of the forward transfer to stall. Many acks are also lost during this period. These may cause the sender to time out while waiting for acks. But the reverse connection continues building up its window, so as time progresses, ack packets get queued behind not one but several data packets. The end result is that the forward connection makes progress in short bursts interspersed by multi-second idle times. Figure 6 illustrates this for a simulation experiment with AF.

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 is 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.

4.3.2 Ack Filtering (AF)

The ACC mechanism described above modifies the TCP stack at the receiver in order to decrease the frequency of acks on the constrained reverse link. Ack filtering, based on an idea suggested by Karn [15], is a gateway-based technique that decreases the number of TCP acks sent over the constrained channel by taking advantage of the fact that TCP acks are cumulative.

When an ack from the receiver is about to be enqueued, the router (or the end-host's routing layer, if the host is directly connected to the constrained link) traverses its queue to check if any previous acks belonging to the same connection are already in the queue. It then removes some fraction (possibly all) of them, depending on how full the queue is. The removal of these "redundant" acks frees up space for other data and ack packets. The policy that the filter uses to drop packets is configurable and can either be deterministic or random (similar to a random-drop gateway, but taking the semantics of the items in the queue into consideration). There is no need for any per-connection state to be maintained at the router — all the information necessary to implement the drop policy is already implicitly present in the packets in the queue.

In the experiments reported in this paper, AF deterministically clears out all preceding acks belonging to a connection whenever a new ack for the same connection with a larger cumulative ack value enters the queue.

4.3.3 TCP Sender Adaptation

ACC and AF alleviate the problem of congestion on the reverse bottleneck link by decreasing the frequency of acks, with each ack potentially acknowledging several data packets. As discussed in Section 4.2.1, this can cause problems such as sender burstiness, a slowdown in window growth, and a decrease in the effectiveness of the fast retransmission algorithm.

We combat sender burstiness by placing an upper bound on the 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 window growth by simply taking into account the amount of data acknowledged by each ack, rather than the number of acks. So, if an ack acknowledges s segments, the window is grown as if s separate acks had been received. This policy works because the window growth is only tied to the available bandwidth in the *forward* direction, so the number of acks is irrelevant.

Finally, we solve the fast retransmission problem by not requiring the sender to count the number of duplicate acks. Instead, with ACC when the receiver observers a threshold number of out-of-order packets, it marks all of the subsequent duplicate acks with a bit to indicate that a fast retransmission is requested. With AF, the reverse channel router takes similar action when it has filtered out a threshold number of duplicate acks. The receipt of even one such marked packet causes the sender to do a fast retransmission.

4.3.4 Scheduling Data and Acks

In the case of two-way transfers, data as well as ack packets compete for resources in the reverse direction (Section 4.2.2). In this case, a single FIFO queue for both data and acks could cause problems. For example, if the reverse channel is a 28.8 Kbps dialup line, the transmission of a 1 KB sized data packet would take about 280 ms. So if two such data packets get queued ahead of ack packets (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 an ack, the acks would be delayed by even more.

To alleviate this problem, we configure the router to schedule data and ack packets differently from FIFO. A particular scheduling algorithm we consider is one that always gives higher priority to acks over data packets (*acks-first* scheduling). The motivation for this is that with techniques such as header compression [8], the transmission time of acks becomes small enough that it affects subsequent data packets very little (unless the per-packet overhead of the reverse channel is large, as is the case in packet radio networks). At the same time, it minimizes the idle time for the forward connection by minimizing the amount of time acks remain queued behind data packets.

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

4.4 Simulation Results

In this section, we present the results of several simulations of one-way and two-way TCP transfers on a network that exhibits bandwidth asymmetry. The simulation topology, depicted in Figure 2, is modeled after the Hybrid wireless cable modem network.

4.4.1 Single One-way Transfer

We conducted a set of experiments, each involving a 50-second transfer in the forward direction. There was no traffic in the reverse direction other than the acks for the forward transfer. Table 1 summarizes the throughputs obtained for three protocol configurations — regular TCP Reno, Reno with ACC and Reno with AF — with different types of return channels. With both ACC and AF, we included the sender adaptation technique described in Section 4.3.3.

The socket buffer size at the sender and receiver was set to 100 KB and each data packet was 1 KB in size. The buffer size at each router was set to 10 packets. The ack size was set to 6 bytes and 40 bytes, respectively, with and without header compression.

The main observation here is that since the transfers are long, the reverse buffer fills up early on. Beyond that point, only one ack in k gets through on average, causing the sender to send out bursts of k packets. As long as k does not exceed the bottleneck buffer size in the forward direction (which is 10 packets in our topology), the increased burstiness of the sender does not lead to losses.

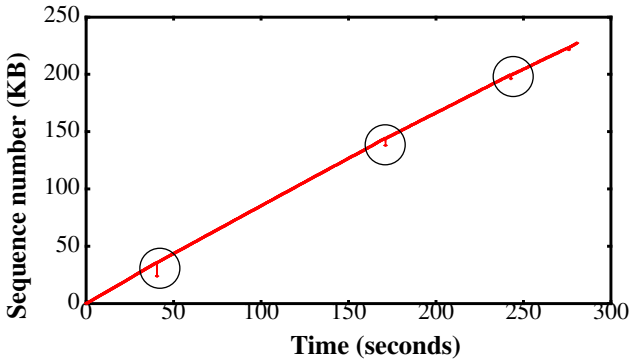

Figure 4. Measurements of the reverse connection operating over a 9.6 Kbps dialup line. The circles indicate times when the reverse connection retransmits packets.

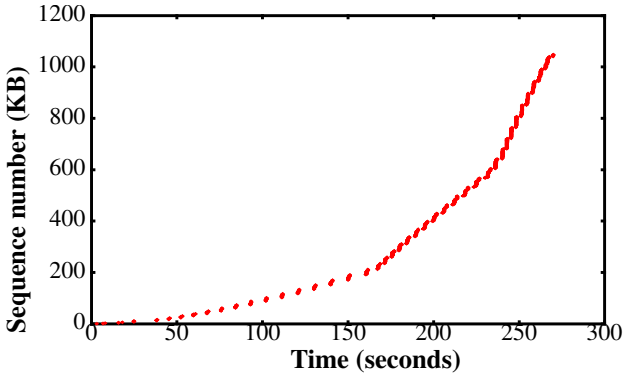

Figure 5. Measurements of the forward connection operating over the 10 Mbps Hybrid wireless cable network. The sharp upswings in its data rate occur whenever the reverse connection suffers a loss and slows down.

could disrupt the sender's fast retransmission algorithm when there is a data packet loss. The sender may not receive the threshold number of duplicate acks although the receiver may have sent out more than the required number. And finally, the loss of the (now infrequent) acks further down the path to the sender could cause long idle periods while the sender waits for subsequent acks to arrive.

4.2.2 Two-way Transfers

We now consider the case when TCP transfers simultaneously occur in the forward and reverse directions. An example of this is a user sending out data (for example, an e-mail message) while simultaneously receiving other data (for example, Web pages). We restrict our discussion to the case of one connection in each direction.

In this scenario, the effects discussed in Section 4.2.1 are more pronounced, because some of the reverse direction bandwidth is used by the reverse transfer. This increases the degree of bandwidth asymmetry for the forward transfer.

In addition, there are other effects that arise due to the interaction between data packets of the reverse transfer and acks of the forward transfer. Suppose the reverse connection is initiated first and that it has saturated the reverse channel and buffer with its data packets at the time the forward connection is initiated. There is then a high probability that many acks of the newly initiated forward connection will encounter a full reverse channel buffer and hence get dropped. Even after these initial problems, acks of the forward connection could often get queued up behind large data packets of the reverse connection, which could have long transmission times (e.g., it takes about 280 ms to transmit a 1 KB data packet over a 28.8 Kbps line). This causes the forward transfer to stall for long periods of time.

Figure 4 and Figure 5 show concurrent reverse and forward connections, measured in the wireless cable modem network. The reverse connection is initiated first. As discussed above, the forward connection starts off very slowly. Figure 5 clearly shows large idle times until about 160 seconds into the transfer. It is only at times when the reverse connection loses packets (due to a buffer overflow at an intermediate router) and slows down that the forward connection gets the opportunity to make rapid progress and quickly build up its window. This is evident from the sharp upswings in the forward connection's data rate just before the times at which the reverse connection retransmits packets, marked by circles in Figure 4.

4.3 Solutions

Most of the problems discussed in the preceding sections arise because of contention for the bottleneck resources in the reverse direction — link bandwidth and buffer space. This observation serves as the starting point for the solutions discussed below.

We first present two techniques — ack congestion control and ack filtering — for alleviating the effects of congestion of ack packets on the reverse channel. We then discuss changes at the TCP sender to enable it to adapt well to the situation where acks are received infrequently. Finally, we present a simple scheduling algorithm for data and ack packets at the reverse channel router to improve performance when there are two-way transfers.

4.3.1 Ack Congestion Control (ACC)

The idea here is to extend congestion control to TCP acks, since they do make non-negligible demands on resources at the low-bandwidth bottleneck link in the reverse direction. Acks occupy slots in the reverse channel buffer, whose capacity is often limited to a certain number of *packets* (rather than bytes), as is the case in our BSD/OS systems.

Our approach is to use the RED (*Random Early Detection*) algorithm [9] at the gateway of the reverse link to aid congestion control. The gateway detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds a threshold, the gateway selects a packet at random and marks it, i.e. sets an *Explicit Congestion Notification* (ECN) bit using the RED algorithm[2]. This notification is reflected to the sender of the packet by the receiver. Upon receiving a packet with ECN set, the sender reduces its sending rate.

The important point to note is that with ACC, *both* data packets and TCP acks are candidates for being marked. The TCP receiver maintains a dynamically varying delayed-ack factor, d, and sends one ack for every d data packets. When it receives a packet with the ECN bit set, it increases d multiplicatively, thereby decreasing the frequency of acks also 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.

2. The gateway can also be configured to drop the selected packet (Random Early Drop), but we chose to mark it instead.

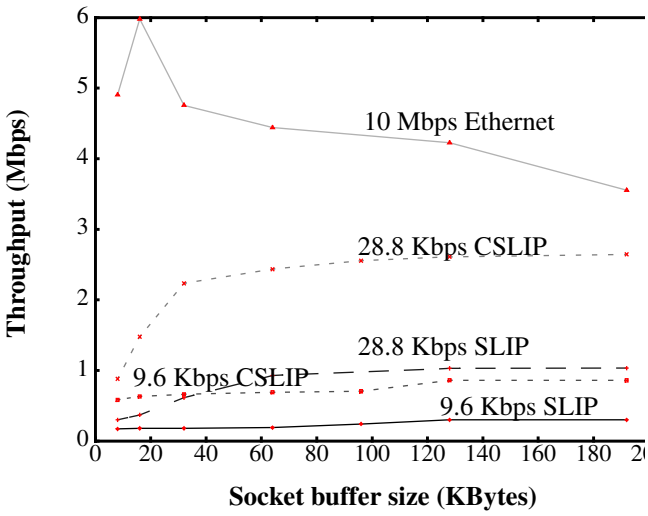

Figure 3. Measured performance of the Hybrid wireless cable network using different return channels, across a range of socket buffer sizes. Each run of the experiment involved the transfer of 1 MB of data in the forward direction between two BSD/OS hosts.

sidered an Ethernet reverse channel. While such a configuration is possibly unrealistic, it serves as a useful data point for comparison.

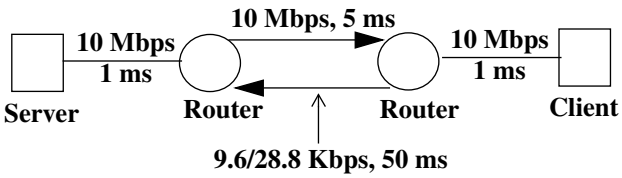

Figure 2. The simulation topology used to model a network with bandwidth asymmetry. The bandwidth and delay parameters have been chosen to closely model the Hybrid wireless cable modem network.

The simulation topology we used to investigate the effects of bandwidth asymmetry is shown in Figure 2. We considered reverse channels of different bandwidths. In practice, reverse channels ranging from slow to high speed dialup lines to ISDN have different delays. But keeping the delay constant (at 50 ms) in the simulation experiments helps us focus on bandwidth asymmetry.

In the following sub-sections, we discuss several performance problems that we observed based on experiments conducted in the real testbed as well as in the simulator. We then discuss a variety of solution techniques and evaluate their efficacy via simulations.

4.2 Analysis of performance problems

We now discuss the problems that arise due the limited bandwidth of the reverse channel in an asymmetric-bandwidth network.

4.2.1 One-way Transfers

We first discuss the case where TCP transfers happen only in the forward direction. A common example of this is a user downloading data from a server. For simplicity, we initially restrict ourselves to the case of a single TCP transfer in the forward direction.

We define the *normalized bandwidth ratio*, k, (as defined in [12]) between the forward and reverse paths as the ratio of the raw bandwidths divided by the ratio of the packet sizes used in the two directions. For example, for a 10 Mbps forward channel and a 100 Kbps reverse 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. This implies that if there is more than one ack for every $k = 4$ data packets, the reverse bottleneck link will get saturated before the forward bottleneck link does, possibly limiting the throughput that can be achieved in the forward direction.

The main effect of bandwidth asymmetry in this case is that TCP ack clocking can break down. Consider two data packets transmitted by the sender in quick succession. While in transit to the receiver, these packets get spaced apart according to the bottleneck link bandwidth in the forward direction. The principle of ack clocking is that the acks generated in response to these packets preserve this spacing (in time) all the way back to the sender, enabling it to clock out new data packets with the same spacing.

However, the limited reverse bandwidth and consequent queuing effects could alter the inter-ack spacing. When acks arrive at the bottleneck link in the reverse direction at a faster rate than the link can support (which happens when $k > 1$ assuming every data packet is acknowledged), they get queued behind one another. The spacing between them when they emerge from the link is dilated with respect to their original spacing. (This is in contrast to ack compression which happens when acks get queued at a fast link, i.e. $k < 1$). Thus the sender clocks out new data at a slower rate than if there had been no queuing of acks. One consequence of this is that the sender's window growth is slowed down.

This is part of the reason why the measured throughputs shown in Figure 3 for dialup reverse channels without SLIP header compression are so low. SLIP header compression (CSLIP) reduces the sizes of acks and decreases k, improving performance. For example, consider the case of a 10 Mbps forward and 28.8 Kbps reverse channel, with a data packet size of 1 KB. With the TCP timestamp option enabled, the ack size is 52 bytes with SLIP and 18 bytes with CSLIP. So k is 18.05 with SLIP and is 6.25 with CSLIP. With TCP delayed acks (one ack for every two data packets), throughput is limited to 10*2/18.05 = 1.1 Mbps and 10*2/6.25 = 3.2 Mbps, with SLIP and CSLIP respectively. These numbers closely match the measured throughputs shown in Figure 3.

In comparison, the performance with an Ethernet return channel is much better because of the absence of bandwidth asymmetry (k is 0.052) and a much smaller link delay than the dialup lines. As an aside, the throughput shows a dip beyond a socket buffer size of 16 KB because larger socket buffer sizes lead to overflow of some router queue along the forward path.

In practice, the reverse bottleneck link will also have a finite amount of buffer space. If the TCP transfer lasts for long enough, this buffer can fill up and cause acks to get dropped. If the receiver acknowledges every packet, on average $(k-1)$ out of every k acks get dropped at the reverse channel buffer. Since in effect only one ack traverses the reverse bottleneck link for every k data packets, acks may not directly limit forward throughput. However, this situation leads to several other problems because the sender now receives fewer acks than it would have otherwise.

First, the sender could become bursty. If the sender receives only one ack in k, it ends up sending out data in bursts of k packets. This increases the chance of data packet loss, especially when k is large. Second, since conventional TCP senders base their window increase on *counting* the number of acks and not on how much actual data is acknowledged, fewer acks imply a slower rate of growth of the congestion window. Third, the receipt of fewer acks

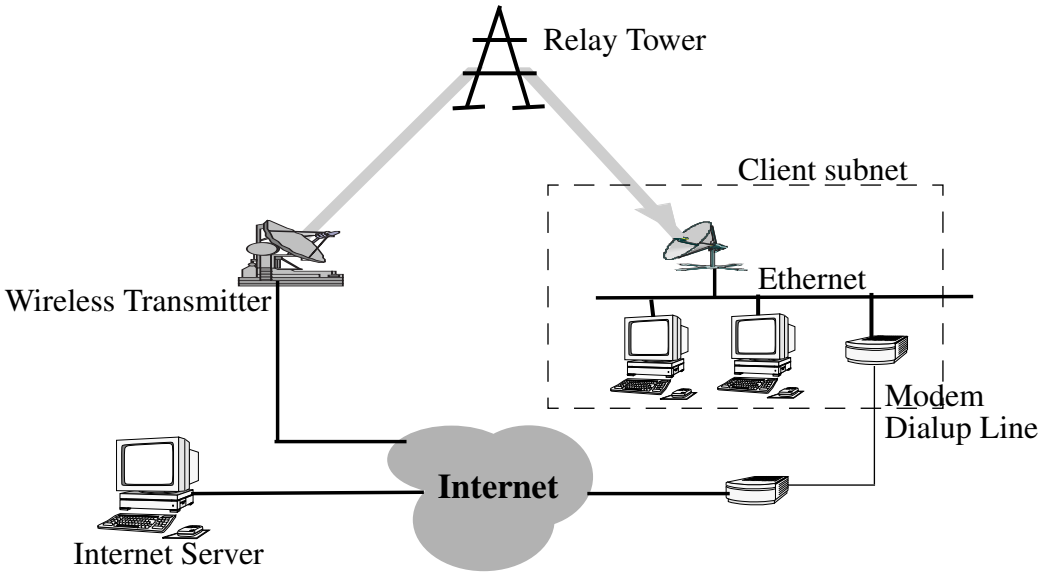

Figure 1. The network topology of the wireless cable modem network which illustrates bandwidth asymmetry. Hosts on the client subnet receive data from the Internet via the 10 Mbps wireless cable link (the *forward* channel) and send out data via a low bandwidth (e.g., dialup) link (the reverse channel).

direction. This arises because of the half-duplex nature of the radio units that cannot simultaneously send and receive packets, having to incur a significant overhead in turning around from one mode to the other.

After studying the impact of each type of asymmetry independently, we combine networks with these different asymmetric characteristics and study the performance characteristics of TCP connections through them. We now describe the details of our simulation setup, workloads, and performance metrics.

3.2 Simulation Setup and Performance Metrics

We used ns [17], an event-driven packet-level network simulator from Berkeley and LBNL for our work. We developed several extensions to this simulator to model the networks of interest to us. We added the notion of a shared link (LAN) to the simulator with the ability to incorporate arbitrary link-layer and media-access protocols. Our simulations of the packet radio network use a MAC protocol loosely based on Ricochet's protocol. The details of these additions are described in Section 5.1.

Our simulation parameters and topologies are closely tied to measurements of the real networks, such as link bandwidths, latencies, packet radio turn-around times, etc. We validated the simulated performance obtained by unmodified TCP and constant-rate UDP traffic with actual measurements in all the real networks. We experiment with two kinds of workloads — large bulk transfers and short Web-like transfers. We also consider simultaneous transfers in opposite directions.

Our main performance metrics are *throughput* measured at the receiver and a metric for fairness, called the *fairness index* [6]. The fairness index, f, is defined as follows: if there are n concurrent connections in the network and the throughput achieved by connection i is equal to x_i, $1 <= i <= n$, then

$$f = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

The fairness index always lies between 0 and 1 for non-negative throughputs, and as explained in [12], is equal to (k/n) if k of the n connections receive equal throughput and the remaining none. Thus, f cannot be less than $1/n$ in a network with n connections. We use the fairness index to understand and analyze the scaling properties of the network when multiple connections are simultaneously active.

4. Bandwidth Asymmetry

In this section, we discuss the performance problems that arise due to bandwidth asymmetry. These include the slowdown and increased burstiness of a TCP sender due to the disruption of ack clocking, and highly variable performance when there are simultaneous TCP transfers in both the forward and reverse directions. We then propose some solutions to these problems and evaluate the improvement in performance.

4.1 Network Topology

The network topology of the wireless cable system is shown in Figure 1. The bandwidth of the forward channel is 10 Mbps. The reverse channel is much slower, usually a dialup phone line of speed up to 28.8 Kbps. In addition, in our measurements, we con-

The following are our major results and conclusions:

- SLIP header compression [11] alleviates some of the performance problems due to bandwidth asymmetry, but does not completely eliminate all problems, especially those that arise in the presence of bidirectional traffic.
- Connections traversing packet radio networks suffer from large variations in round-trip time caused by the half-duplex nature of the radios and asymmetries in the media-access protocol. This adversely affects TCP's loss recovery mechanism and results in degraded performance.
- The various end-to-end and router-based techniques that we propose help improve performance significantly in many asymmetric situations. These include decreasing the frequency of acknowledgments (acks) on the constrained reverse channel (*ack congestion control* and *ack filtering*), reducing source burstiness when acknowledgments are infrequent (*TCP sender adaptation),* and scheduling data and acks intelligently at the reverse bottleneck router.
- In addition to improving throughput for individual connections, our proposed modifications also help improve the fairness and scaling properties when several connections contend for scarce resources in the network. We demonstrate this via simulations of bulk and Web-like transfers.

The rest of the paper is organized as follows. Section 2 describes some related work and Section 3 discusses the details of our experimental and simulation methodology. In Section 4, after analyzing the problems that arise due to bandwidth asymmetry, we propose and evaluate several solution techniques. In Section 5, we discuss the problems that arise due to asymmetry in latency and media-access in packet radio networks and how this leads to a large variation in round-trip time, and evaluate some solutions. In Section 6, we combine wireless cable and packet radio technologies and investigate the issues of scale and performance when bandwidth and latency asymmetries are present together. We present our conclusions in Section 7 and plans for future work in Section 8.

2. Related Work

Several researchers have identified and proposed solutions to transport protocol problems that arise in single-hop wireless networks [1, 2, 3, 4]. The main issue considered in these papers is the impact of packet losses due to reasons other than congestion (wireless error, handoff, etc.) on TCP performance. We view our work as being in the natural progression of such research, with the overall goal of understanding and improving the performance of reliable transport protocols like TCP in the face of ever-increasing heterogeneity in network technologies and characteristics. The specific measurements reported in this paper were taken over a wireless cable modem network and a packet radio network.

There has been some previous work on understanding the effects of two-way traffic on TCP performance. In [20], the authors demonstrate how two-way traffic can lead to *ack compression*, where closely-bunched acknowledgments disrupt the smooth ack-clocked transmission at the sender. More recently, there has been interest in how asymmetric-bandwidth networks exacerbate this problem. In [16], the authors model a network with bandwidth asymmetry and derive analytical expressions for throughput in terms of packet loss probability and the *normalized asymmetry ratio* under certain ideal assumptions. They also propose the use of a *drop-from-front* strategy for dropping acknowledgments at the bandwidth-constrained reverse link. In [13], the authors demonstrate how bidirectional traffic over asymmetric links leads to ack compression, and consequently, degraded performance. They investigate a backpressure mechanism to limit data flow in the reverse direction, but conclude that this alone is not enough for good performance.

There have also been studies of bandwidth asymmetry in the context of satellite networks [7, 18]. The main distinction between such a network and the wireless cable modem network we consider in this study is that the former has a much larger bandwidth-delay product, which could be the dominating factor in performance. Finally, some basic performance measurements of Metricom's Ricochet packet radio network are presented in [5], such as one-way transmission delays of unidirectional traffic.

While the results and analysis in [13] and [16] are very useful, the set of problems is far from being understood or solved. In addition to proposing and evaluating other schemes to alleviate the adverse effects of bandwidth asymmetry, we also characterize other types of asymmetry that occur in wide-area wireless networks. We evaluate our solutions for these networks in terms of connection throughput, fairness, and scaling behavior.

3. Experimental and Simulation Methodology

In this section, we describe our experimental testbed, simulation setup, and traffic workloads used in the study.

3.1 Experimental Testbed

We use a combination of simulation and actual experimentation on a real heterogeneous, wireless testbed to evaluate the performance of TCP, understand the reasons for observed performance, and design end-host and router-based techniques to improve performance. Our simulation topologies and parameters are derived from the following networks in our testbed:

- *Wireless cable modem network:* This is a wireless cable modem network using technology developed by Hybrid Networks, Inc. (*www.hybrid.com*). The aggregate bandwidth of the (unidirectional) forward channel is 10 Mbps[1] and the one-way link latency is about 5ms. The topology for this testbed is shown in Figure 1. The downstream channel for data operates in the 2.4 GHz range and is down-converted at the receiving end to standard television channel frequencies. The reverse channel could be a dialup line, an ISDN line, a wireless channel using a wide-area packet radio network, etc.
- *Packet radio network:* Our packet radio network is based on Metricom Inc.'s Ricochet network (*www.metricom.com*). The topology for this network is shown in Figure 8. The packet radios operate in the 915 MHz ISM band and have a raw link speed of 100 Kbps. The poletop units typically have a range of several hundred meters.

The wireless cable modem testbed is an example of a network with bandwidth asymmetry depending on the return path used, which could be a dialup phone line (e.g., 14.4 Kbps or 28.8 Kbps), or a wireless channel. In a good installation of the wireless cable modem network, the bit-error rate of the forward channel is negligible. We therefore do not focus on the effects of bit-errors on the forward channel in this paper.

The packet radio network we study is an example of a network that does not have explicit (bandwidth) asymmetry, but has the characteristic that the flow of traffic (e.g., TCP data) in one direction is affected by the flow of traffic (e.g., TCP acks) in the opposite

1. A 30 Mbps 2-way wireless cable system is currently under development; this system also exhibits bandwidth asymmetry.

To appear, Proc. 3rd ACM/IEEE Intl. Conference on Mobile Computing and Networking (MobiCom), Budapest, Hungary, Sept. 1997.

The Effects of Asymmetry on TCP Performance

Hari Balakrishnan, Venkata N. Padmanabhan, and Randy H. Katz
{hari,padmanab,randy}@cs.berkeley.edu
Computer Science Division, Department of EECS
University of California at Berkeley, Berkeley, CA 94720-1776.

Abstract

In this paper, we study the effects of network asymmetry on end-to-end TCP performance and suggest techniques to improve it. The networks investigated in this study include a wireless cable modem network and a packet radio network. In recent literature (e.g., [16]), asymmetry has been considered in terms of a mismatch in bandwidths in the two directions of a data transfer. We generalize this notion of *bandwidth asymmetry* to other aspects of asymmetry, such as *latency* and *media-access,* and *packet error rate*, which are common in wide-area wireless networks.

Using a combination of experiments on real networks and simulation, we analyze TCP performance in such networks where the throughput achieved is not solely a function of the link and traffic characteristics in the direction of data transfer (the *forward* direction), but depends significantly on the *reverse* direction as well. We focus on bandwidth and latency asymmetries, and propose and evaluate several schemes to improve end-to-end performance in these cases. These include techniques to decrease the rate of acknowledgments on the constrained reverse channel (*ack congestion control* and *ack filtering*), techniques to reduce source burstiness when acknowledgments are infrequent (*TCP sender adaptation),* and algorithms at the reverse bottleneck router to schedule data and acks differently from FIFO.

1. Introduction

The Transmission Control Protocol (TCP) is widely used in the Internet for reliable, unicast communications. The robustness of TCP in a wide variety of networking environments is the primary reason for its large-scale deployment. However, emerging networking technologies pose new challenges to TCP in terms of performance, requiring analysis and solutions. In this paper, we focus on the challenges to end-to-end TCP performance that arise due to network asymmetry, especially in the context of wide-area wireless networks. The increased interest in asymmetric networks is motivated by technological and economic considerations as well as by popular applications such as Web access, which involve a substantially larger flow of data towards the client (the *forward* direction) than from it (the *reverse* direction).

Examples of networks that exhibit asymmetry include *wireless cable modem* networks, *direct broadcast satellite* networks, and *Asymmetric Digital Subscriber Loop* (ADSL) networks, where bandwidth in the forward direction is often orders of magnitude larger than that in the reverse. Such asymmetry is accentuated when the channel is unidirectional, necessitating the use of a different, often low-bandwidth channel (e.g., a dialup line or a bandwidth-constrained and lossy wireless channel) for communication in the reverse direction.

Our study is not limited to networks where the asymmetry is explicit and obvious because of mismatched bandwidths — we also study TCP dynamics in *packet radio networks*, where traffic flowing simultaneously in different directions can adversely affect performance. This is because most packet radio networks use half-duplex radio units, which cannot transmit and receive data frames at the same time. In addition, we combine two wireless technologies — wireless cable and packet radio — in our study to understand the problems that arise in these situations when different types of asymmetry are tied together.

Asymmetry is inherent in several wide-area wireless networks, where it is often the case that a central transmitter (or base station) can transmit at high power to receiving portable/mobile units. However, to reduce power consumption, these units transmit to the base station at relatively low power. In addition, they often have to contend with other mobile units to gain access to the channel.

We generalize the various phenomena and examples described above to the following definition of asymmetry: *a network is said to exhibit network asymmetry with respect to TCP performance, if the throughput achieved is not solely a function of the link and traffic characteristics of the forward direction, but depends significantly on those of the reverse direction as well*. In addition to the *bandwidth asymmetry* described above, this definition extends to other types of asymmetry, such as *latency* and *media-access,* and *packet error rate*. In this paper, we study bandwidth, and latency and media-access asymmetries, both individually and in combination. We use measurements on a real testbed as well as simulations experiments with different choices of topology and workload, to identify the performance problems. Based on these results, we propose and evaluate several techniques to improve performance. The wireless networks that serve as the basis for our work include a wireless cable modem network and a packet radio network.

Fundamentally, network asymmetry affects the performance of reliable transport protocols such as TCP because these protocols rely on feedback in the form of cumulative acknowledgments from the receiver to ensure reliability. In addition, TCP is *ack-clocked*, relying on the *timely* arrival of acknowledgments, to make steady progress and fully utilize the available bandwidth of the path [10]. Thus, any disruption in the feedback process could potentially impair the performance of the forward data transfer. For example, a low bandwidth acknowledgment path could significantly slow down the growth of the TCP sender window during slow start, *independent* of the link bandwidth in the direction of data transfer. A second example is from packet radio networks, where variable latencies in the presence of bidirectional traffic (caused, for instance, by acknowledgements flowing in a direction opposite to data packets) causes the sender's round-trip time estimate to be highly variable. This inflates TCP's retransmission timeout value, thereby impairing loss recovery.