BUFFER MANAGEMENT SCHEMES FOR ENHANCED TCP PERFORMANCE OVER SATELLITE LINKS

Sonia Jain and Eytan Modiano
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139

***Abstract*— We designed queue management policies to enhance the performance of TCP over satellite networks. Our queue management schemes, Smallest Window First (SWF) and Smallest Sequence Number First (SSF) give priority to sessions that have just opened or traveled across lossy and high delay channels. Thus, they combat the inherent unfairness of TCP. Our protocols help *slow* sessions grow their congestion windows more rapidly and effectively compete against other sessions for bottleneck resources. Although our protocols are useful in other contexts, they are particularly useful in satellite networks. With their high round trip times (RTTs), sessions can be severely penalized by the closure of the TCP congestion window. We show that our schedulers not only provide increased fairness, but they also substantially reduce the latency associated with transmitting short files (e.g., e-mails). SWF doubles the goodput of lossy sessions over FIFO and provides a 16% improvement over DRR. It also reduces the latency of short file transfers up to 40% over DRR.**

I. INTRODUCTION

The Transmission Control Protocol (TCP) is today's most ubiquitous Transport Layer protocols. In this paper we are interested in enhancing the performance of TCP over satellite networks. Satellite networks differ from terrestrial networks in a few significant ways that impact the performance of TCP. Satellite networks are characterized by high delay, high bit error rates (BER), and high bandwidth. High propagation delays slow the growth of the TCP congestion window thus reducing throughput. High BERs trigger the TCP congestion control apparatus causing window closures. As a result, TCP performance over satellite links is typically poor [12].

Many studies relating to TCP have suggested ways to improve performance in terrestrial and wireless networks. Some mechanisms by which TCP performance can be improved include: split connections [3] and [20], slow start modifications [2] and [16], and error handling [4] and [5]. Other researchers have also considered TCP modification for improving TCP performance over satellite links; see [1], [7], [11], [12], and [15]. However, TCP modifications require a long standardization process and are often slow to be adopted.

In contrast, rather than modify the TCP protocol, we attempt to enhance TCP performance by better engineering the protocols with which TCP interacts. In particular, we focus on the design of queue management schemes for improving TCP performance over shared satellite links (i.e., at the satellite gateway). Our queue management policies ensure that all sessions are treated fairly regardless of their channel characteristics, despite the fact that TCP is inherently biased against long delay and lossy sessions.

Queue management is responsible for allocating bandwidth, bounding delay, and controlling access to the buffer. We view queue management as being composed of two independent components: the scheduler and the packet admission strategy. The scheduler selects packets from the queue and passes them to the physical layer for transmission. The packet admission strategy controls the admission of packets into the buffer.

Total latency is comprised of the propagation delay, the transmission delay, and the queuing delay. The propagation and the transmission delay characteristics are largely fixed for a given static network. However, by controlling which sessions gain access to the channel, we can control the queuing delay, and thus the latency, experienced by users. Queue management policies also play a crucial role in the overall *fairness* provided to each session.

By releasing packets to the physical layer, schedulers determine the order in which packets are trans-

mitted. The simplest scheduler is FIFO. It transmits packets in the order that they are received. Connections are served in proportion to the share of the buffer they occupy. Typically, the greediest source, the source with the lowest RTT and the highest data rate, receive the lion's share of the bandwidth. Therefore, FIFO gateways are not particularly effective at achieving fairness, unless they are paired with *good* packet admission policies. In Fair Queuing (FQ), the gateway attempts to guarantee each session an equal or fair share of a bottleneck link. Deficit Round Robin (DRR) is a particular implementation of a fair queuing scheme. In DRR, the buffer is sub-divided into buckets. When a packet arrives, it is hashed into a bucket based on its source address. These buckets are then serviced in a round robin fashion [17].

Packet admission policies are vital to the overall *fairness* of queue management policies. If packets from certain sessions are refused entry into the buffer, these packets cannot be scheduled for transmission. The simplest packet entry policy, DropTail, admits packets into the buffer until the buffer is full and drops those packets that arrive at a full buffer. In heterogeneous networks, the DropTail policy penalizes connections that have high RTTs. These sessions experience a greater number of packet drops [8].

Random Early Detection (RED) is a popular packet admission policy designed to reduce congestion within networks [9]. RED drops packets probabilistically based on the average number of packets in the buffer. RED uses the same drop probability on all flows. When heterogeneous traffic shares the same link, using the same drop probability on all flows can lead to unfair bandwidth allocation. Flow Random Early Detection (FRED) is a modified version of RED designed to be fair [14]. FRED is similar to RED except the drop probability is determined on a per-flow basis. Flows with a lower average occupancy have a smaller drop probability than flows with higher average occupancy rates. Longest Queue Drop (LQD) is another packet dropping policy [19]. In the case of LQD, if a packet encounters a full buffer, it is not necessarily dropped. Instead, a packet from the flow occupying the greatest share of the buffer is dropped. LQD policies effectively favor flows with long propagation delays and penalize those with short propagation delays.

In this paper we consider queue management schemes for improving TCP performance over satellite links. Our queue management schemes attempt to give priority to sessions that have either just started transmission or have a small window size. In doing so, our queue management schemes allow these sessions to open-up their windows more rapidly. In the next section we describe our queue management schemes, and in subsequent sections we evaluate their performance through NS-2 simulations.

II. The SWF and SSF Schedulers

Our queue management policies called Shortest Window First (SWF) and Smallest Sequence Number First (SSF) are designed to give priority to packets from *slow* sessions. SWF schedules and admits packets into the buffer based on the source window size of the packets. Packets arriving from sessions with smaller source window sizes are considered to be slower. A session may have a small window for a variety of reasons, the connection only just opened, is lossy, or has an especially high RTT. To assist *slow* sessions in gaining a fair share of the bottleneck bandwidth, SWF prioritizes packets from slow sessions. These packets are sorted to the front of the queue. Thus, packets from slow sessions are transmitted without lengthy queuing delays. Scheduling packets based on window size ensures SWF is sensitive to sessions that suffer from timeout events. It also allows sessions to initially rapidly open their congestion windows.

The notion of preemption alone is not enough to ensure fairness. The buffer's packet admission policy must be considered as well. The packet admission strategy we use is a close approximation of Longest Queue Drop (LQD). If the buffer is full, the packet at the end of the queue is dropped to make room for the newly arriving packet. Since the router's queue is fully sorted, the packet that is dropped is the packet that comes from the source with the largest window size. It is likely that the source with the largest window size occupies the largest share of the buffer, hence, the LQD-like policy. Using window size, as an indicator of *slowness*, is useful, especially in light of studies that maintain TCP connections with large windows are much more robust to packet losses than those with small windows [14]. All references to the SWF scheduler, unless otherwise specified, are implemented with an LQD-type drop policy.

The SSF scheduler is very similar to the SWF scheduler, except that with SSF scheduling, packet sequence numbers are used to determine priority. The sequence number can be used as an indicator of *slowness*. Small sequence numbers suggest a connection has just opened, is traversing a lossy link, or has a large propagation delay. In an environment with many non-homogenous sessions, *slow* sessions simply cannot transmit packets as fast or as successfully as their competitors. Like the SWF scheduler, the SSF scheduler assists slow sessions by aggressively growing their TCP congestion window. This is accomplished by sorting packets with small sequence numbers to the front of the queue. As before, we use an approximate LQD policy. When a packet drop is necessary, the packet with the highest sequence number is dropped.

There are implementation issues associated with both SWF and SSF that warrant explanation. The use of the SWF scheduler necessitates an adjustment to the TCP header. The TCP header contains a field for the window size that stores the advertised window size of the receiver, for flow control purposes. However, SWF requires a field for the current window size of the transmitter. Since the size of the advertised window is only needed when the connection is initiated, one possibility is to overwrite the window size field in the header. The other possibility is to create a new field within the header for maintaining the current sequence number.

With SSF sequence numbers for persistent sessions can exceed the length allotted to them in the TCP header field, at which point, they wrap around. In addition, in practice, sequence numbers never start at zero as a security precaution. One possible solution is for the scheduler to keep track of the number of packets it has seen from each session and use this information for scheduling and admissions.

III. Throughput and Fairness Performance

We perform simulations using the Network Simulator, $NS-2$. In our simulations, we assume that all sessions use TCP Reno. We consider the performance of our schedulers over a variety of different traffic patterns. The performance of SWF and SSF is compared to that of DRR with LQD and FIFO with DropTail. We are most interested in understanding the behavior of our scheduler in terms of bandwidth fairness and link utilization. If link utilization is low, providing a high level of fairness is inconsequential. Link utilization is simply the system throughput over the bottleneck bandwidth.

A. Performance with lossy sessions

Here we study the performance of our schedulers in the presence of a mix of lossy (i.e., high error rate) sessions and lossless sessions. We consider ten sessions attempting a persistent file transfer over a shared 1Mbps bottleneck link. At the end of a 2000 second interval, we measure both aggregate and individual session throughput. With ten sessions, ideally, each session would obtain 10% of the bottleneck bandwidth. While neither the SWF nor the SSF scheduler can guarantee a completely equitable division of scarce resources, they perform much better than either FIFO or DRR. See Tables I and II for results when one of the ten sessions has a packet loss rate of 10% and 1% respectively. (Our packet size is 1000 bytes which implies the packet error rates correspond to bit error rates of at least 10^{-5} and 10^{-6}, respectively.)

Notice from Tables I and II that the highest aggregate throughput is obtained by the FIFO scheduler. However, the throughput of the lossy session using SWF and SSF is more than double that of FIFO and 5% - 20% greater than that of DRR. This increased performance for lossy sessions comes at a minimal decrease to overall throughput.

Lossy sessions perform poorly when FIFO is used due to the DropTail policy. Packets from lossy sessions are unable to gain entry into the buffer. However, the reason for FIFO's unfairness is also the reason for its higher throughput. The DropTail policy disregards lossy sessions, which can only transmit a limited number of packets successfully, and gives priority to lossless sessions instead, which allows them to continue increasing their window size and push data through the network. DRR is relatively more fair to lossy sessions. If packets from lossy sessions are awaiting transmission, they will be transmitted in round robin fashion. Part of the reason DRR performs well is due to its use of the LQD policy. Under this policy, lossy sessions always have access to the buffer.

There is a slight decrease in overall throughput when using the SWF scheduler. SWF gives priority to lossy sessions that continue to lose packets due to physical channel characteristics. The losses due to the

Scheduler	Total Goodput (bps)	Goodput of Lossy Session (bps)	Avg Goodput of Lossless Session (bps)
SWF	926216	31094	99458
SSF	947261	31331	101770
DRR	951041	26534	102723
FIFO	953099	13453	104405

TABLE I

TEN SESSIONS. ONE SESSION HAS A PACKET LOSS RATE OF 10%.

Scheduler	Total Goodput (bps)	Goodput of Lossy Session (bps)	Avg Goodput of Lossless Session (bps)
SWF	924900	94092	92312
SSF	959576	94766	96090
DRR	951845	89960	95915
FIFO	954804	46443	100929

TABLE II

TEN SESSIONS. ONE SESSION HAS PACKET LOSS RATE OF 1%.

physical channel, which the scheduler cannot prevent, limit the number of packets that can be successfully transmitted. Furthermore, prioritizing lossy sessions at the expense of lossless sessions, limits the number of packets that lossless sessions can transmit. Interestingly, when the packet loss rate is 1%, SWF actually favors the lossy session over the lossless sessions. At high loss rates, packets are lost frequently leading the congestion window to close. Thus, the throughput of the lossy sessions remains constrained and less than that of lossless sessions. However, when the loss rate is low, congestion control is not invoked as often. The lossy session can grow the TCP congestion window large enough so that there are packets backlogged for transmission at the gateway, while their congestion window is still small enough so that packets belonging to the lossy session receive priority. We also find that as the number of lossy sessions increases, the performance of our schedulers decreases as lossy sessions compete with each other for priority [13].

B. Variable Propagation Delays

In this section, we consider the performance of our schedulers in the presence of a mix of sessions with high and low propagation delays. Our setup is the same as that in the lossy channel case. However, we now examine the effect satellite traffic with different propagation delays has on the performance of our schedulers. In one scenario, we consider sessions with a round trip propagation delay of one second or half a second. Table III has the results for the case of one high delay session and nine low delay sessions. We also consider a case where a mixture of satellite and terrestrial traffic share the same bottleneck link. Sessions have round trip propagation delays on the order of half a second (the satellite case) and 20ms (the terrestrial case). See Table IV for the case where half of the ten sessions are terrestrial.

Notice from Tables III and IV that once again FIFO achieves the highest overall throughput and SSF and SWF achieve a slightly lower overall throughput. However, SSF achieves by far the fairest allocation of bandwidth. Notice that SSF achieves nearly four times the throughput of FIFO for the high delay session and about a 10% improvement over DRR. Although SWF provides a marked improvement over FIFO in terms of fairness, its performance is not appreciably better than that of DRR. A high propagation delay does not directly imply that the congestion window will close, rather it suggests that the congestion window grows more slowly. In the case of SWF, when the buffer is full, packets from sessions with low propagation delays will be dropped and their windows will close. Since SWF schedules packets based on the window size of the source, it is possible for packets from sessions with low propagation delays to be sorted to the front of the queue. Without the external pressure lossy channels place on the congestion window, low and high delay sessions will alternate priority leading to a reduction in overall performance.

Although a session with a high RTT can grow a congestion window comparable to competing sessions, a session with a high RTT will not have a large sequence number compared to competing sessions. Therefore, scheduling based on packet sequence numbers expedites the flow of traffic from sessions that face high propagation delays. SSF is able to distribute bottleneck bandwidth more equitably than DRR, with a minimal decrease in overall goodput.

IV. DELAY PERFORMANCE

In this section, we consider the delay performance of our queue management schemes when used for short file transfers. Some of the queue management schemes in use today attempt to combat the unfair effects of TCP, among them DRR. Earlier, we considered persistent file transfers. While queue management schemes like DRR provide a certain degree

Scheduler	Total Goodput (bps)	Goodput of High Delay Session (bps)	Avg Goodput of Low Delay Session (bps)
SWF	925262	63057	95801
SSF	948823	95443	94820
DRR	949814	87167	95915
FIFO	953440	14905	104282

TABLE III

TEN SESSIONS. ONE SESSION HAS A RTT OF 1.0S, AND THE OTHERS HAVE A RTT OF 0.5S.

Scheduler	Total Goodput (bps)	Avg Goodput of High Delay Session (bps)	Avg Goodput of Low Delay Session (bps)
SWF	916980	64662	118730
SSF	930190	93851	92187
DRR	956950	87613	103780
FIFO	957650	19822	171710

TABLE IV

TEN SESSIONS. FIVE SESSIONS HAVE RTTS OF 0.01S, AND THE OTHERS HAVE A RTT OF 0.5S.

of fairness in the long run, they are not equipped to deal with short-term file transfers. The benefits of fair queuing are not achieved because the session ends too soon. Short file transfers require only a few RTTs to complete transmission. However, the RTT is a function of the queuing delay. Therefore, the queue management policy has a large effect on the over all time to transmit a file.

In our simulations, files arrive according to a Poisson random process with interarrival times that are exponentially distributed and have a mean of two seconds. We model a GEO satellite network; the round trip propagation delay for each session is half a second. The performance of our schedulers is evaluated for both different loads as well as different file sizes by measuring the amount of time to transmit each file.

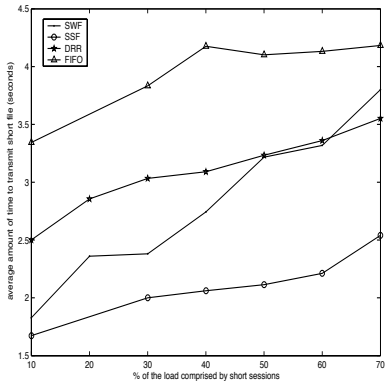
Today's Internet is dominated by numerous short-lived flows (e.g., e-mail) even though long-term flows (e.g., MPEG files) actually account for the bulk of data traffic being exchanged [18]. We simulate the performance of short and long file transfers using a mix of 10KB and 100KB files, see Figure 1, as well as a mix of 1MB and 10MB files [13]. We compute the transmission time for files as the percentage of short files that comprise the load varies (keeping the total load constant).

In this environment, SSF achieves the smallest delays for short files but large delays when transmitting long files. This is because with long files, SSF drives all sessions to have the same sequence number. At which point, active sessions transmit packets in a round robin fashion. Under this scheme long sessions all complete their transmissions at approximately the same time regardless of when their session began. In contrast, SWF achieves small delays for short files at minimal additional delays for the long files.

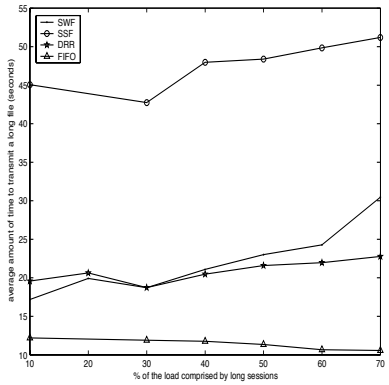
The time to transmit short files increases for all schedulers as the percent of the load comprised by short files increases. In the case of SWF, the time to transmit long files also increases. Short sessions compete with each other for priority. Additionally, each new short session decreases the relative priority of long file transfers. Therefore, as the number of short sessions increases, the delay in transmitting both long and short files increases. For DRR, the transmission time stays relatively constant over all combinations of the load. However, for FIFO, the transmission time associated with long files actually decreases. FIFO prioritizes long sessions by default since they typically have larger windows and occupy the bulk of the buffer. Therefore, as the fraction of short files increases, there are fewer competing long file transfers and FIFO's performance for long file transfers improves.

SWF performs substantially better than DRR when the majority of the load is comprised of large files, which is in practice true. It provides up to a 40% improvement over DRR for short file transfers. In some cases, SWF also has an advantage over DRR in the transmission of long files. Although, SWF has relatively poor performance compared to FIFO in long file transfers, FIFO requires almost 80% more time to transmit short files. When the portion of short files in the load is under 50%, SWF's superior performance for short files compensates for its performance with long files.

We performed simulations where short sessions comprised 5% of the system load. However, we varied the length of small sessions anywhere from 10KB to 1MB. Large files were 10MB in length. As expected, improvement in latency is the highest when message lengths are small (between 10-30KB). As can be seen from Figure IV, SWF provides as much as 20% reduction in delay over DRR and 70% reduction over FIFO. Of course, as the size of short files increases,



(a) Time to transmit short 10KB files



(b) Time to transmit long 100KB files

Fig. 1. Performance of 10KB and 100KB files as the load comprised by 10KB files increases.

this improvement vanishes.

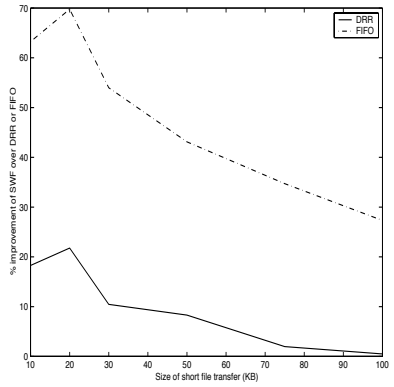


Fig. 2. Improvement of SWF over DRR and FIFO

V. Conclusion

We designed two new queue management schemes, SWF and SSF. Our scheduling schemes use novel metrics for packet scheduling and admission. Unlike existing queue management schemes, which rely on the number of packets a session has buffered, we utilize the congestion window size and packet sequence number in making scheduling decisions. While the SSF shows marked improvement over schedulers in certain aspects, it is the SWF that shows a general increase in the level of fairness provided. SWF treats lossy sessions fairly and allocates them an equitable share of bottleneck bandwidth. It doubles the throughput of lossy sessions in comparison to FIFO and provides a significant improvement over DRR. SWF is also fair to short-term files transfers that compete against persistent traffic. It sharply reduces the latency experienced by short file transfers in comparison to existing queue management schemes. We observed as much as a 40% reduction in delay as compared to DRR and an 80% reduction as compared to FIFO. By aggressively growing the congestion windows of *slow* session, our queue management policies are able to reverse some of the bias inherent to TCP.

References

[1] M. Allman, ed. "Ongoing Research Related to TCP over Satellites," *Internet RFC 2760*, 2000.

[2] M. Allman, S. Floyd, and C. Partridge. "Increasing TCP's Initial Window Size," *Internet RFC 2414*, 1998.

[3] A. Bakre and B.R. Badrinath. "I-TCP: Indirect TCP for Mobile Hosts," *Proceedings of the 15th International Conference on Distributed Computing Systems*, 1995.

[4] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless," *IEEE/ACM Transactions on Networking*, December 1997.

[5] H.M. Chaskar, T.V. Lakshman, and U. Madhow. "TCP over Wireless with Link Level Error Control: Analysis and Design Methodology," *IEEE/ACM Transactions on Networking*, 1999.

[6] D. Chiu and R. Jain. "Analysis of the Increase and Decrease Algoritmns for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, 1989.

[7] R.C. Durst, G.J. Miller, and E.J. Travis. "TCP Extensions for Space Communications Protocols," *Proceedings of MOBICOM*, 1996.

[8] S. Floyd and V. Jacobson. "On Traffic Phase Effects in Packet Switched Gateways," *Computer Communication Review*, 1991.

[9] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, August 1993.

[10] L. Guo and I. Matta. “The War Between Mice and Elephants,” *Proceedings of IEEE International Conference on Network Protocols*, 2001.

[11] T.R. Henderson. “Networking over Next-Generation Satellite Systems,” University of California Berkeley, 1999.

[12] T.R. Henderson and R.H. Katz. “Transport Protocols for Internet-Compatible Satellite Networks,” *IEEE Journal on Selected Areas of Communications*, 1999.

[13] S. Jain. “Enhancing the Performance of TCP over Satellite Links,” Massachusetts Institute of Technology. June 2003.

[14] D. Lin and R. Morris. “Dynamics of Random Early Detection,” *Proceedings of SigComm*, 1997.

[15] V. Mhatre and C. Rosenberg. “Performance Improvement of TCP-based Applications in Multiple Access Satellite Systems,” *Proceedings of IEEE Vehicular Technology Conference*, 2002.

[16] K. Poduri and K. Nichols. “Simulation Studies of Increased Initial TCP Window Size,” *Internet RFC 2415*, 1998.

[17] M. Shreedhar and G. Varghese. “Efficient Fair Queueing using Deficit Round Robin,” *Proceedings of SigComm*, 1995.

[18] F.D. Smith and F.H. Campos and K. Jeffay and D. Ott. “What TCP/IP Headers Can Tell Us About the Internet,” *Proceedings of ACM SIGMETRICS*, 2001.

[19] B. Suter, T.V. Lakshman, D. Stiladis, and A.K. Choudury. “Design Considerations for Supporting TCP with Perflow Queueing,” *Proceedings of IEEE INFOCOMM*, 1998.

[20] R. Yavatkar and N. Bhagwat. “Improving End-to-End Performance of TCP over Mobile Internetworks,” *Proceedings of Mobile ’94 Workshop on Mobile Computing Systems and Applications*, 1994.