Performance Optimization of TCP/IP over Asymmetric Wired and Wireless Links

Dibyendu Shekhar, Hua Qin, Shivkumar Kalyanaraman, Kalyan Kidambi
Department of Electrical, Computer and System Engineering
Rensselaer Polytechnic Institute
Tory NY 12180-3590, U.S.A.
shekhd@rpi.edu, qinh@rpi.edu, shivkuma@ecse.rpi.edu

Abstract

In this paper, we analyze TCP performance in asymmetric networks, where the throughput significantly depends on the reverse direction and packet loss. We introduce a new operational model called the "AMP model" which, with an understanding of TCP dynamics and buffer management techniques explains the performance effects seen. We apply this model to guide design improvements in buffer management (ack-regulation) and scheduling schemes to achieve performance improvements of an order of magnitude or more. The schemes have been tested through simulation and implementation, and the module driven improvements observed are better than those proposed in earlier literature.

1. Introduction

Transport Control Protocol (TCP) is widely used in Internet communications. However, the TCP/IP performance over wired and wireless networks is significantly diminished due to the effects of asymmetry. Figure 1 shows a network topology with wired and wireless asymmetric channels: the forward channel from broadband networks or wireless networks ranges in speeds is much higher than the reverse channel does. This bandwidth asymmetry has a profound impact on TCP performance because the TCP throughput is regulated (and limited) by the flow of acknowledgements (called "ack clock")[1]. The more asymmetric the bandwidth, the more severe impact will have on TCP performance.

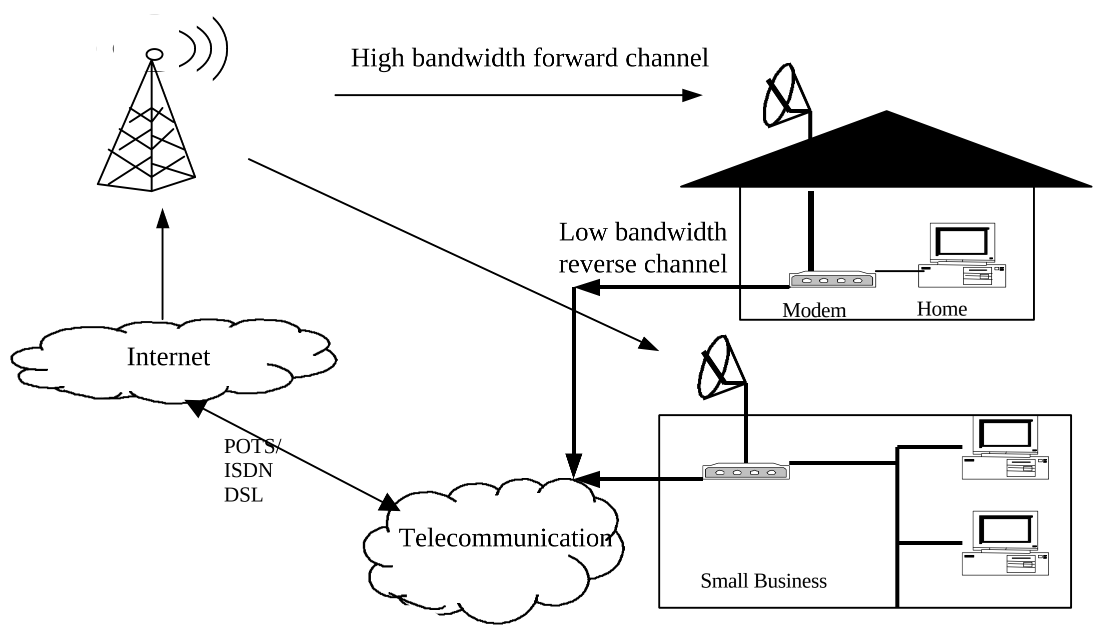

Figure 1: Network Topology with Asymmetric Links

The impact of asymmetry on TCP/IP performance has been characterized by Lakshman [2]using an index called ***normalized asymmetry ratio (k)*** which is the *ratio of raw bandwidths on both directions to the ratio of packet sizes in both directions*. When k is greater than one, the TCP throughput on the forward channel is restricted to a maximum of (**forward channel bandwidth) / k**. Other factors like bi-directional traffic (e.g.: downloading a web page while sending an email) or protocol overhead (e.g.: PPPoE, PPTP, L2TP, ATM etc) will increase k and further aggravate the asymmetry problem.

Balakrishnan [3], Keshav [4] and the PILC group [5] independently show that performance can be substantially increased by making two key changes: simply suppressing acknowledgements on the reverse channel (***ack-filtering***), and regenerating them after the reverse link has been traversed (***ack-reconstruction***). Balakrishnan also apply the same techniques to address types of asymmetry other than bandwidth asymmetry including asymmetry in delay, loss rates etc. We classify these techniques, as "***ack-regulation techniques***" since they are very different from mechanisms like Random Early Drop (RED) because the latter are designed to drop packets, not acks, and are used to signal congestion to TCP, not alleviate asymmetry problems. Our work focuses on generalizing these studies into a model called the ***AMP model***, and making specific design improvements (Smart Ack Dropper: SAD) for performance optimization based upon this model.

2. The AMP model

We extend the notion of normalized bandwidth ratio (k) [2] to form a simple model called the AMP model. The model is described as follows:

- Assume that, over an observation period, the ***fraction of the reverse link allocated to ack traffic is saturated at A acks/second***, and
- ***Each ack generates M packets on the average (M = multiplicative factor),*** due to the effects of TCP dynamics and ack-regulation schemes working together, and
- The ***average size of packets is P bits/packet,***

The ***forward link capacity allocated to packets corresponding to these acks is F bits/s,*** then the ***forward link throughput over the observation period is limited to Min (F, A*M*P) bits/second***.

Observe that Lakshman's normalized bandwidth ratio "k" characterizes an absolute upper bound on the forward throughput based upon link bandwidths and packet vs. ack sizes. The AMP expression, on the other hand, is an *operational bound* achievable with the asymmetric channel augmented with ack-regulation schemes. In particular, if the ack-regulation components can achieve a maximum average multiplicative factor of M, then the AMP expression is tighter bound on the achievable forward throughput. Moreover, it also accounts for scheduling allocations to acks and corresponding packets and vice versa (in the definition of F and A), and can hence be useful in understanding effects of bi-directional traffic.

For example, if the reverse link speed is 64kbps, the forward link speed is 8Mbps, packet size is 1000 bytes, ack size = 40 bytes, TCP is in its slow start phase (generating two packets per ack) and no ack-regulation schemes are used. Then ack rate (A) would now be 200 acks/second , the multiplicative factor M=2, and P = 8000 bits, the maximum throughput limit on the forward link = Min (8 Mbps, 3.2 Mbps) = 3.2 Mbps. Note that *the number of TCP flows sharing the asymmetric link is immaterial in this model* (it is captured in M). In particular, in this case if the TCP sources are all in congestion avoidance (and not in slow start), then M is closer to 1, and limit is even lower (about 1.6 Mbps)!

The AMP model is also useful in analyzing the ***bi-directional*** traffic case, especially when link-sharing schemes like CBQ [6] are deployed. Specifically, assume that in each direction, there are two classes (queues) served by CBQ: one for packets and one for acks. Further assume that packets on the forward link get a fraction f ($1 > f > 0$) of the forward link capacity (C_f), and on the reverse link get a fraction g ($1 > g > 0$) of the reverse link capacity (C_r). Acks, therefore get fractions 1-f and 1-g of capacity forward and reverse links, respectively. Assume that P_{ack} is the size of acks in bits/ack and *that the reverse channel (both packets and acks) is saturated at their respective scheduling shares.* The bounds on the maximum rates of packets and acks in both directions are given in the following table:

	Max Packet Rate (pkts/s)	**Max Ack Rate (acks/s)**
Reverse Channel	$G^* C_r /P$ (saturated)	$(1-g)^* C_r$ /Pack (saturated)
Forward Channel	**$\mathbf{Min[(1\text{-}g)\ C_r\ M/Pack,\ f^*C_f /P]}$**	$Min[\ f^*C_f /P_{ack}\ ,\ g^*\ C_r /P\ ]$

Observe that f (the link fraction allocated to packets on the forward link) should be chosen to be large, and can be as large as 99% for pure TCP/IP bi-directional traffic. The choice of g (the link fraction allocated to packets on the reverse link) is a tradeoff between performance and policy considerations. We will use this model to design ack regulation and scheduling policies.

3. The Smart-Ack Dropping (SAD) and Ack Regeneration Policy

The Smart-ack dropping technique (SAD) is a simple extension of concepts developed by Balakrishnan [3] and Srinivasan [4], i.e. to suppress as many acks in the reverse channel as possible because acks are cumulative. Balakrishnan's "*ack-filtering*" technique involves checking the entire queue upon the arrival of a new ack to remove earlier acks for the same connection. The goal is partially to free some space in the queue for other data packets and acks, and partially to compress the ack information. Srinivasan [4] independently proposes an "*ack-collapsing*" technique where all acks are queued, but at the transmission opportunity, the latest ack is sent and all others are dropped.

SAD is close to Srinivasan's *"ack-collapsing"* in concept, but uses minimal per-flow state to avoid enqueuing acks, which are going to be dropped anyway. This way, we require only a buffer of size N (acks) where N is the number of active flows (assuming separate packet buffers). In particular, SAD works as follows (Figure 2):

We assume a FIFO queue for acks (and optionally packets). The per-flow information (stored as a hash table) includes a *single bit, which indicates if an ack of that flow exists in the queue,* and the *latest ack number seen from that flow*. When an ack arrives and the per-flow bit is zero, then we enqueue the ack, set the per-flow bit and copy the ack number into the table. If the per-flow bit is already set upon arrival of a non-duplicate ack, then we update the per-flow ack number information to this value, and drop the ack. When an ack is dequeued for transmission, the latest value of ack number from the table is copied to the header (header checksum adjusted), and the bit in the table is cleared. Ack processing is O(1), since we search the hash table and not the FIFO queue. This scheme can be extended to account for duplicate acks and SACKs [7].

A detailed analysis of the scheme applying the AMP model is also described in [7]. Specifically, if TCP is in congestion avoidance, the throughput is almost constant at W/RTT. The impact of SAD is to use the reverse channel capacity of A acks/s to support a maximum forward rate of Min[m*A, W/RTT] packets/s. When TCP is in slow start, each ack reaching the TCP source results in (m + 1) segments being sent where m is the number of acks suppressed by SAD. In fact, if the TCP source remains in slow start phase, the AMP model predicts that SAD can compensate for arbitrary degrees of asymmetry (i.e. any finite k). However, since most TCPs reach congestion avoidance within a few RTTs, the multiplicative factor saturates at the value of (near-constant) m seen during the congestion avoidance.

SAD also reduces the probability of negative interactions with RTT estimation (which occurs in Balakrishnan et al's scheme) because acks are sent out in a timely manner, and queueing delay is bounded. The suppression of acks leads to burstiness in the forward direction. This burstiness can be alleviated by using an ***acknowledgement regeneration/reconstruction (AR)*** technique at the end of the reverse link as suggested by Keshav [4]and Balakrishnan [3]. The regeneration scheme would regenerate (and optionally smooth out) acks suppressed by SAD. In fact, the regenerator could regenerate more than one ack per MSS (which we quantify as the "***regeneration factor***", ***R***), all the way upto one ack for every byte acked. The regeneration factor, R directly affects the rate of TCP window increase especially during slow start.

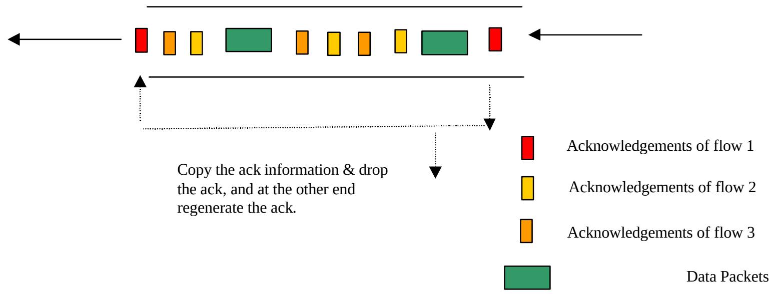

Figure 2:Illustration of SmartAckDropper(SAD)

4. Performance Evaluation

In this section, we use this actual model to design our experimentation architecture on "ns" simulation and VxWorks implementation. Finally, we present the results of simulation and implementation.

Figure 1 shows a general network topology, which illustrates bandwidth asymmetry. In our experiment, more specially, we study downstream channel ranges in speed from 6Mpbs-8Mbps, and upstream channel ranges in speed from is 64Kpbs-768Kpbs.

4.1. Network Simulation

We use network simulation ns (version 2) as our experiment tool. We first present packet loseless simulation both in unidirectional and bi-directional traffic. Then, we present packet loss and delay simulation in unidirectional traffic.

4.1.1. Lossless simulations

- **Lossless undirectional TCP flow**

Talbe 1 shows the throughputs obtaned for two configuration — TCPReno and Reno with SAD but without any ack regulation schemes in undirectional single flow — with different types of return channels. The simulation yielded the following results, which can be used as a baseline to evaluate the effectiveness of SAD. As shown, the asymmetry problem grows increasingly worse as the upstream channel becomes more constrained to a worst case of 19% utilization. However, these problems can be overcome almost entirely by using SAD. For instance, in the worst-case scenario (8 Mbps/64 kbps), ***throughput rose four-fold to 6.2 Mbps***.

Downstream link/ Upstream link (Mbps/Kbps)	TCP Reno Throughput (Mbps)/Link Utilization	Reno with SAD Throughput (Mbps)/Link Utilization	K
8/64	1.52 / 19%	6.20 / 77.5%	5[1]
8/128	3.01 / 37%	6.79 / 84.88%	2.5
8/256	5.98 / 74.75%	7.28 / 91%	1.25
8/640	7.46 / 93.25%	7.45 / 93.13%	0.5
8/784	7.46 / 93.25%	7.46 /93.25%	0.41

Table 1: Uni-directional Single Flow Simulation with and without SmartAckDropper

[1] (8 Mbps/64 kbps)/(1000 bytes/40 bytes) = 5

Additional throughput can be gained using AR techniques (Table 2**).** A regeneration factor (R) of R generates R acks for every ack suppressed by SAD. With a regeneration factor of 4, a maximum throughput of 7.4 Mbps can be achieved even in the worst-case scenario.

Downstream/ Upstream link (Mbps/kbps)	**Throughput (Mbps)**	**Regeneration factor**
8/64	7.51	4
8/64	7.40	3
8/64	6.98	2
8/64	6.65	1

Table 2: SmartAckDropper and Ack Regenerator

Although this performance is the result of a trivial uni-directional flow, in reality one will find multiple flows, both uni-directional and bi-directional. We focus on the latter here for brevity. Please refer our detailed technical report for a full suite of simulations [7].

- **Lossless bi-directional TCP flow**

The problems associated with bi-directional transfers are unique and require a slightly different approach. Here, link utilization on *both* channels is key performance indicator. Balakrishnan [3] proposed that an acks-first scheduling policy be used. However, we find that if acks are given priority, the rate of packets on the reverse (low-speed) channel drops to unacceptably low values. Yet if packets are given priority, the effects of asymmetry lead to low packet throughput on the forward (high-speed) channel and beyond a point, this cannot be compensated by ack-regulation schemes (since the behavior is also dependent on TCP window dynamics).

Table 3 shows performance of a single FTP flow in each direction (baseline), with FIFO queuing at the upstream link. Here, the presence of bi-directional traffic reduces downstream throughput (and increases asymmetry) by almost a factor of 10. Downstream throughput worst case is 18.8 kbps, compared to 1.5 Mbps in the uni-directional model (Table 1). Even in the case of 640 kbps upstream, downstream throughput is reduced to 7.4% available capacity. This drop can be attributed almost entirely to increased asymmetry and bidirectional issues, as there were no lost packets observed.

Downstream/ Upstream link (Mbps/Kbps)	**Downstream/ Upstream link Throughput (Kbps)**	**Downstream/ Upstream link Link Utilization**	**K**
8/64	18.8/58.8	0.235% / 91.875%	5
8/256	255.2/232	3.19%/90.63%	1.25
8/640	588/579.6	7.35%/90.56%	0.5

Table 3: Bi-directional Single Flow Simulation without SmartAckDropper

The cleanest way to improve performance in such cases is to set up one queue for acks and one for packets in both directions, serviced with a link-sharing scheduler (CBQ: Class-Based Queuing) rather than a priority scheduler. On the ack queue on the reverse direction, both SAD and AR are applied. CBQ assigns fractions, F and G respectively, to the packet queues on the reverse and forward links respectively. Based on analysis in section 2, we set F large — as large as 99.7%. The setting of G is a policy/performance issue which we investigate here.

Table 4 shows performance using a combination of CBQ, SAD and AR for different values of G and degrees of asymmetry. Aggregate (packet + ack) throughput for both downstream and upstream directions is shown. Note that the performance in all cases has increased ***by a factor of 10-20*** compared to corresponding values in Table 3. Furthermore, as G increases from 0.1 to 0.9, the upstream aggregate throughput increases (and in some cases nearly doubles or triples) at the expense of downstream throughput (20-50% throughput loss), hence leading to a policy/performance tradeoff.

Downstream/Upstream link (Mbps/kbps)	**Aggregate Throughput (Mbps/kbps)**	**G**
8/64	3.39/34.4	0.1
8/64	3.73/44.4	0.5
8/64	6.43/54	0.9
Downstream/Upstream link	**Aggregate Throughput**	**G**
8/256	5.46/84.4	0.1
8/256	5.59/149.2	0.5
8/256	6.55/217.2	0.9
Downstream/Upstream link	**Aggregate Throughput**	**G**
8/640	6.7/333.6	0.1
8/640	6.65/336.8	0.5
8/640	7.02/542.8	0.9

Table 5: Effect of SAD+AR+CBQ

4.1.2. Loss and delay simulations

In most wireless network lossy links and delay links can cause throughput degradation due to bit errors and handoffs. In this section, we demonstrate that SAD results in significant performance improvements in these two environments.

- Downstream data loss simulation

In this simulation, we build a network loss module in downstream links, and set random loss level rate from 10^{-4} to 10^{-1}.

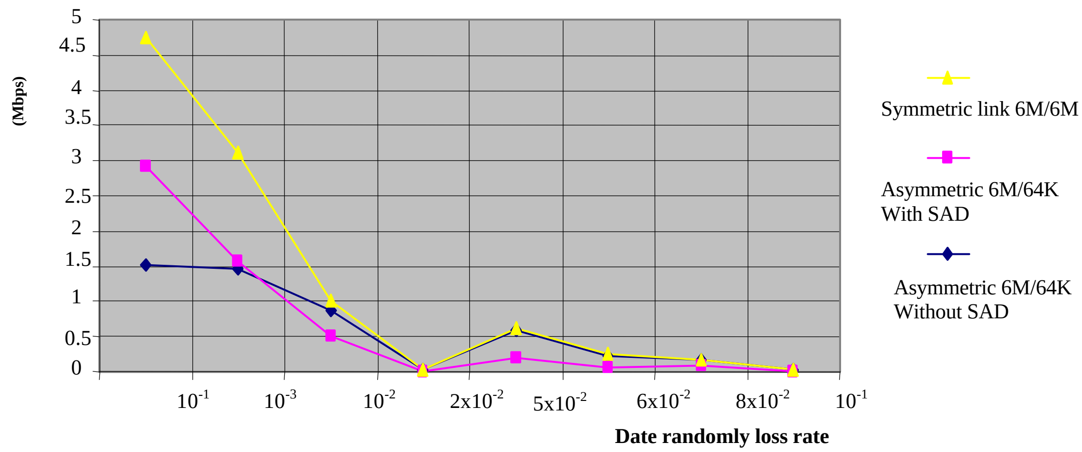

Chart 2: Uni-directional Single Flow Simulation on Loss Link

- Link delay simulation

We change link delay time in this simulation. Chart 3 shows that SAD still can improve performance on delay link ranging from 1ms to 100ms.

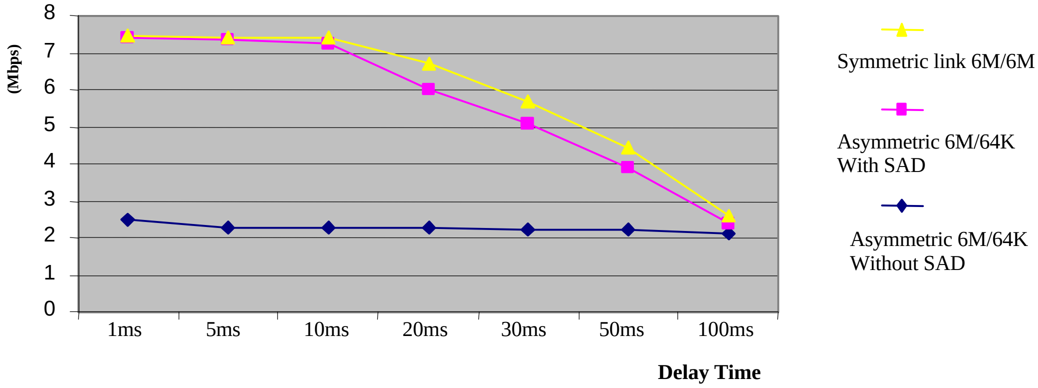

Chart 3: Uni-directional Single Flow Simulation on Delay Link

4.2. VxWorks implementaion

This implementation is based on VxWorks: a real-time operating system running on Pentium platform. We first emulate asymmetric bandwidth and then implement SmartAckDropper on VxWorks testbed. Finally, we evaluate implementation results. The implementation source code is available in [9].

4.2.1. Link emulation

Asymmetric link is emulated by restricting the packets sending rate at the egress port. This requiring bandwidth measured at the egress port. The algorithm used is taken from Clark and Feng’s allocated capacity paper [8]. Bandwidth updating is based on running average. This calculation is only valid in 1millisecond intervals since that is the granularity of our system clock. The average is:

```
bitsInWin = oldAvgRate * ((float) winLen / (float) oneSec);
 updatedBitsInWin = bitsInWin + len * 8;
 newAvgRate =(float)updatedBitsInWin / ((float) ((now - *prevTime) + winLen)/(float) oneSec);
*prevTime = now;
```

This function returns the new average rate and modifies the prevTime parameter.
We can adjust new average rate from 64Kbps to 8Mbps, and asymmetric link is built.

4.2.2. VxWorks box design for SmartAckDropper

Figure 3 shows the SmartAckDropper design on VxWorks box.

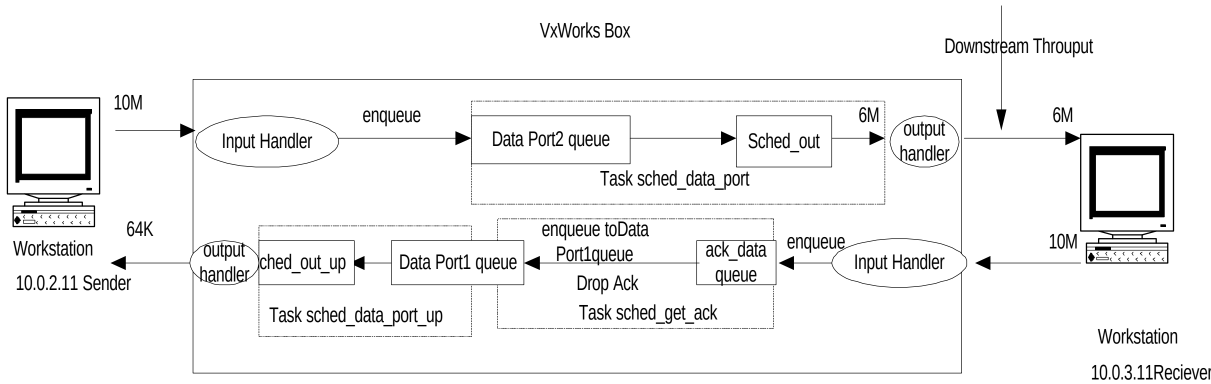

Figure 3: SmartAckDropper Algorithm Implemented on VxWorks

Figure 3 has the following explanations:

- End to end, one is sender and the other is receiver
- Asymmetric link is changed from 64Kbps to 8Mbps
- Task agent to deal with the packets from Sender and Receiver
- 3 tasks are built, "sched_out" for downstream; and "sched_get_pack" and "sched_data_port_up"for upstream.

4.2.3. Implementation results

TCP packets generated by Netperf from the sender, and downstream throughput can be obtained at the receiver end. Table 6.1 and Table 6.2 show the downstream throughputs in single flow and multi-flows (10 flows). The test results prove AMP model, and confirmed the performance trends observed in the simulations.

Downstream /Upstream	**Without SAD**	**With SAD**
	Throughput	**Throughput**
6M/64K	2.78M	5.78M
6M/6M	5.77M	
8M/64K	2.78M	6.61M
8M/128K	5.66M	6.79M
8M/8M	6.6M	

Table 6.1 Uni-directional Single Flow Implementation with SmartAckDropper

Downstream /Upstream	**Without SAD**	**With SAD**
	Throughput	**Throughput**
6M/64K	2.63M	5.59M
6M/6M	5.88M	
8M/64K	2.79M	6.3M
8M/128K	5.34M	7.37M
8M/8M	7.2M	

Table 6.2 Uni-directional Multi-Flow Implementation with SmartAckDropper

5. Summary

We presented a simple AMP model and an improved ack-regulation scheme called SAD to explain and improve the performance of TCP/IP over wireless networks. We also propose the use of link-sharing schedulers with just two queues (ack and packet queues, with SAD implemented on the ack queues) at the ATU-R to effectively support bidirectional Internet traffic. The percentage gains in performance can range from 100%-2000% . However arbitrary degrees of asymmetries cannot be solved by ack regulation schemes because performance is ultimately dictated TCP dynamics as well.

References

[1] M. Allman, V. Paxson and W. Stevens. TCP Congestion Control. April 1999, RFC 2581.

[2] T. V. Lakshman, U. Madhow, B. Suter. Window-based Error Recovery and Flow control with a Slow Acknowledgement Channel: a Study of TCP/IP Performance. Proceedings of INFOCOM '97, April 1997.

[3] H. Balakrishnan, V.N. Padmanabhan, and R. H. Katz. The Effects of Asymmetry on TCP Performance. ACM MobiCom, Spember 1997.

[4] K. Srinivasan. Method and Apparatus for collapsing TCP Acks on Asymmetric Conditions. US patent number 5,793,768, Issued Aug. 11, 1998, Filed Aug. 13, 1996

[5] G Montenegro, J. Griner, J. Border and M. Kojo. Performance Enhancing Proxies. draft-ietf-pilc-pep-00.txt. July 1999.

[6] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. IEEE/ACM transactions on Networking, Vol 3 No. 4, August 1995.

[7] D. Shekhar, S. Kalyanaraman and K. Kidambi, "TCP/IP over ADSL," Technical report, Available from http://www.ecse.rpi.edu/Homepages/shivkuma

[8] D.Clark, W.J. Feng Explicit Allocation of Best Effort Packet Delivery Service.

[9] http://networks.ecse.rpi.edu/~qinhua/research/thesis/vimpl/vimpl.html