Mild Aggression : A new approach for improving TCP Performance in Asymmetric Networks

Vijay T. Raisinghani **Abhishek Patil** **Sridhar Iyer**

K.R. School of Information Technology
IIT Bombay, Mumbai, INDIA
{rvijay, abhi, sri}@it.iitb.ernet.in

Abstract

An asymmetric network is one in which there is a mismatch between the characteristics of the forward and reverse channel. Typically, the forward channel is capable of high transmission rates and the reverse channel has lesser bandwidth. Cable modem networks and satellite networks are some technologies that exhibit network asymmetry. TCP, due to its widespread acceptance, is also used in asymmetric networks. However, the reverse channel being a bottleneck causes many problems in the behavior of traditional TCP protocols. The TCP protocol is heavily dependent on ACKs. The variation in the ACK arrival rate, caused due to network asymmetry, results in slower initial window growth and burstiness on the forward channel. Existing approaches to improve the performance of TCP in asymmetric networks use techniques such as reduction in number of ACKs. However, these approaches do not take into consideration the high bandwidth availability in the forward direction for early ramp up of TCP window. In this paper, we present a new approach, *Mild Aggression* which exploits the high forward channel bandwidth, by modifying the RTT estimate and incrementing the *cwnd* by *Mild Aggression factors,* to improve the performance of TCP in an asymmetric environment.

1. Introduction

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. The asymmetry may be due to lower bandwidth up−link on the reverse channel such as dial−up lines or latency and media−access asymmetry, such as in packet radio networks. The typical asymmetric technologies today are cable modem networks, direct broadcast satellite and asymmetric digital subscriber loop [6]. Asymmetric networks provide efficient and economical solutions for applications that do not require high up−link bandwidth. A typical application is Web access, where the data flow towards the client (the forward direction) is larger than the data flow towards the server (the reverse direction). For example, the forward channel may be a 2 Mbps wireless link and the reverse channel may be a low bandwidth 9.6 Kbps dial up link. This provides a cheap and efficient mechanism for web surfers who mostly download data from the web [6].

TCP, due to its existing wide−spread deployment, is also used as the transport−layer protocol in asymmetric networks. Network asymmetry hampers the performance of normal TCP since the TCP protocol assumes a reasonably high bandwidth in the reverse direction. For example: TCP−Reno uses congestion control algorithms such as *Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery* [5] . The TCP protocol maintains a congestion window *cwnd*. In *Slow Start* phase the window size *cwnd* is effectively doubled per RTT (RTT is the Round Trip Time, i.e. the time elapsed between sending a data packet and receiving an ACK for that data packet, at the sender) whereas in *Congestion Avoidance* phase the *cwnd* is incremented by 1 per RTT. TCP depends on feedback in the form of cumulative ACKs from the receiver to ensure reliability. It is *ACK−clocked* i.e. it relies on the timely arrival of ACKs, to increase *cwnd* and fully utilize the available bandwidth. Thus, any disruption in the ACK arrival rate impairs the forward data transfer [6].

The bottleneck on the reverse channel impairs the flow of the ACKs. The ACK flow is slower than the flow of the data packets in the forward direction. Also, ACKs may get dropped due to congestion. If some ACKs get dropped and an ACK arrives later, cumulatively acknowledging the data packets, it leads to sudden growth in *cwnd.* This results in the sender pumping in a burst of data on the forward channel. Also, this disrupted ACK flow results in slower growth in *cwnd*. In Packet Radio Networks (half−duplex i.e. the wireless units cannot simultaneously send and receive data), due to wireless communication, the latencies are variable. The ACKs flowing in a direction opposite to the data packets cause interference. Thus variability in RTT estimates leads to poor performance of TCP in case of noisy and congested channels [6].

Our *Mild Aggression* approach reduces the impact of RTT variation by promoting appropriate window growth in the *Slow Start* phase. It attempts to utilize the high bandwidth forward channel efficiently, thereby improving the performance of TCP. *Mild Aggression* uses *mild aggression factors* for improving the RTT estimate and incrementing *cwnd.* This helps in improving *cwnd* growth, in an asymmetric environment, during *Slow Start*.

The organization of the paper is as follows: Section 2 provides the necessary background. Section 3 discusses the Mild Aggression approach and section 4 gives the simulation details. section 5 mentions the related work . Section 6 concludes with a mention of on−going work.

2. Background

For ensuring end−to−end reliability and performance the TCP protocol heavily depends on the timely arrival of ACKs. Network asymmetry results in variable ACK arrival times and hence in varying RTT estimates. The degree of asymmetry can be quantified in terms of the forward channel bandwidth, data packet size and ACK size [10]. For a particular connection, let the forward channel bandwidth be b_f , reverse channel bandwidth be b_r , the data packet size be p_f and the ACK size be p_r. Thus for TCP to utilize the forward channel bandwidth effectively and no dropping of ACKs on the reverse channel, for this connection, the following condition needs to be satisfied: $(b_f / b_r) \leq (p_f / p_r)$.

If the above condition is not satisfied the TCP ACK−clock can break−down. For example, consider a 2 Mbps forward channel, a 10 kbps reverse channel, data packet size of 1600 bytes and ACK size of 40 bytes. In this case, the bandwidth ratio (b_f / b_r) is 5 times greater than the packet size ratio (p_f / p_r). This means that if there is more than one ACK for every 5 data packets, the reverse link will get saturated before the forward link does, resulting in poor TCP performance .

The performance of TCP becomes even worse when the reverse channel is also transferring data packets in addition to ACKs. Now, there is contention between the data packets and the ACKs. This may lead to delayed or lost ACKs. As a result, even though the forward channel has sufficient bandwidth, the throughput (*cwnd / rtt*) is very low.

Balakrishnan et al[6] have suggested *Sender Adaptation (SA), ACK filtering (AF)* and *ACK congestion Control (ACC)* as some approaches for improving the performance of TCP in asymmetric networks. However, in these approaches, TCP is still dependent on the ACK arrival rate for *cwnd* growth. This results in under−utilization of the forward channel bandwidth.

3. Mild Aggression

TCP uses ACKs to ensure reliability, synchronization and flow control. However, if ACKs are delayed or dropped because of reverse channel characteristics, even though the data packets are transmitted correctly, the performance of the TCP is poor .

We propose a new approach, *Mild Aggression (MA),* to improve TCP performance, which takes cognizance of the fact that the forward channel bandwidth in asymmetric links is under−utilized. It assumes that that there are no congestion or transmission losses for the data packets. In *MA,* TCP is modified (on end hosts) to make it independent of the reverse channel characteristics. The modified TCP uses the *Mild Aggression factors* (MAFs) θ and ε, defined subsequently.

In *Sender Adaptation* [6] the sender sends data in sizes of *maxburst.* The bursts are spaced by a timer. This timer value is the ratio of *maxburst* to *cwnd/rtt* , where *rtt* is the round trip time. *rtt* is a function of both the forward delay and reverse channel delay. Ideally, the forward and reverse delay may be the same. Thus, if the actual forward delay is known to the TCP sender, it can increase the congestion window *cwnd* per (`2 * forward delay`). However, determining the forward delay is not possible because the sender and receiver may not be synchronized.

Mild Aggression attempts to approximate the ideal value of (`2 * forward delay`) by using a MAF. This is the basic idea in *Mild Aggression* i.e. using the MAFs, θ and ε for calculating a better RTT termed as *goodRTT (grtt)* and incrementing *cwnd* respectively. θ and ε are based on the network conditions. The *grtt,* which is a fraction of the actual RTT, attempts to utilize the high bandwidth in the forward direction as follows: The sender sends out data spaced out using this *grtt*. Also, the *cwnd* is incremented by a MAF, ε, every *grtt*. This results in faster *cwnd* growth and hence better utilization of the forward channel bandwidth specially in the *Slow Start* phase. If the network asymmetry is high then a higher degree of *Mild Aggression* may be useful.

The detailed MA algorithm is presented in the following sections.

3.1 Definitions

cwnd	Congestion Window at sender
rtt	Round Trip Time
cwnd_last	Congestion Window when last ACK was received
goodRTT(grtt)	Mild Aggression parameter for RTT
θ	Mild Aggression Factor for calculating goodRTT; $0 < \theta < 1$
ε	Mild Aggression Factor for incrementing *cwnd*; $\varepsilon > 0$
receiver_window	Receiver window advertised to the sender
data_acknowledged	Amount of data acknowledged by the ACK received

3.2 Algorithm

Initialization

```
       cwnd =  4  /* To take advantage of high forward channel bandwidth, [4]
                       recommends using a value of 4 for cwnd */
```

Action performed by TCP sender

```
      while (data to be sent) {
            if ACK received then
                  grtt = rtt * θ
                  if (slow−start phase)
                        cwnd = cwnd + data_acknowledged
                  else if (congestion avoidance phase)
                        cwnd = cwnd + data_acknowledged / cwnd
                  endif
                  cwnd_last = cwnd
            endif

            while(new ACK not received){
              start grtt timer
              on new ACK arrival {
                break; // break out of the while(new ACK not received)
              }
              on grtt time−out {
                if (slow−start phase)then
                  /* Increase cwnd by ε per grtt */
                  cwnd = min(min(cwnd + ε, cwnd_last*2),receiver_window)
                else if (congestion avoidance phase)
                  /* Break and use normal TCP congestion avoidance*/
                  break;
                endif
              } // on grtt time−out
              if (cwnd == 2*cwnd_last) then
                 /* Behavior equivalent to that on a time−out
                    i.e at least one ACK expected per window doubling*/
                 break;
            } //  while(new ACK not received)
      } //  while (data to be sent)
```

Action performed by TCP receiver

```
      while (receiving data){
            on data packet receipt {
                  send ACK
            }
            on (out−of−order packet OR packet loss detection){
                  send NACK
            }
      }
```

3.3 Discussion

Unlike in traditional TCP, the initial window size *cwnd* is taken to be larger than 1 because the forward channel bandwidth is usually high. This helps in initial jump in the throughput. Since the initial *cwnd* is 4, wastage of 2 RTTs to reach a *cwnd* of 4 is avoided during *Slow Start* phase. Also it has been shown that using an initial cwnd of 4 segments does not negatively impact overall performance [11].

Ideally, the forward and reverse delay may be the same and the actual forward delay is known to the TCP sender. TCP can increase the congestion window *cwnd* per `( 2 * forward delay)`. However, since determining the forward delay is not possible, *Mild aggression* approximates the ideal value of `(2 * forward delay)` by using the factor θ to calculate *grtt*. This *grtt* tries to utilize the large bandwidth on the forward channel by considering the reverse channel to have sufficient bandwidth.

To avoid burstiness the sender sends out data using the *grtt*. However, *grtt* cannot be used throughout for *cwnd* growth or sending out data. *grtt* should be used only in the initial stages (*Slow Start*) to improve the sluggish *cwnd* growth, as once the *cwnd* reaches a sufficiently high value the bandwidth is optimally used.

The *grtt* timer is started after the processing of normal TCP portion of code is over. If the *grtt* timer times out then in *Slow Start* phase the window is incremented according to the MAF ε and parameter *cwnd_last.* In *Congestion Avoidance* phase then window size is incremented similar to that in traditional TCP implementations.

The parameter *cwnd_last* helps in restricting *cwnd* growth so that it doesn't exceed the limit (*cwnd_last* `* 2`) when the TCP sender does not get ACKs for a long time (which may be because of forward or reverse congestion). This avoids pumping of data packets on forward link and thus prevents congestion collapse on the forward channel and TCP performance degradation.

The parameter ε helps in gradual increase of the window size *cwnd* at least up to *cwnd_last * 2,* even if the sender does not receive ACKs for a long time. Here ε takes care of ACK loss due to congestion on the reverse link. However, if at least one ACK is not received by the time *cwnd* reaches *cwnd_last * 2*, then time−out occurs and TCP shows normal congestion control behaviour.

In congestion avoidance phase, *Mild Aggression* behaves as a traditional TCP protocol, to ensure that the forward channel doesn't get congested due to large number of packets.

4. Mild Aggression Simulation

4.1 Simulation Setup

We have considered two types of traffic generation i.e. one−way and two−way transfers. In one−way transfers there is no competing data traffic through the reverse channel. For example: downloading of data by user from a server and the reverse link not being used for any other data transfer except ACKs. In two−way transfers there are

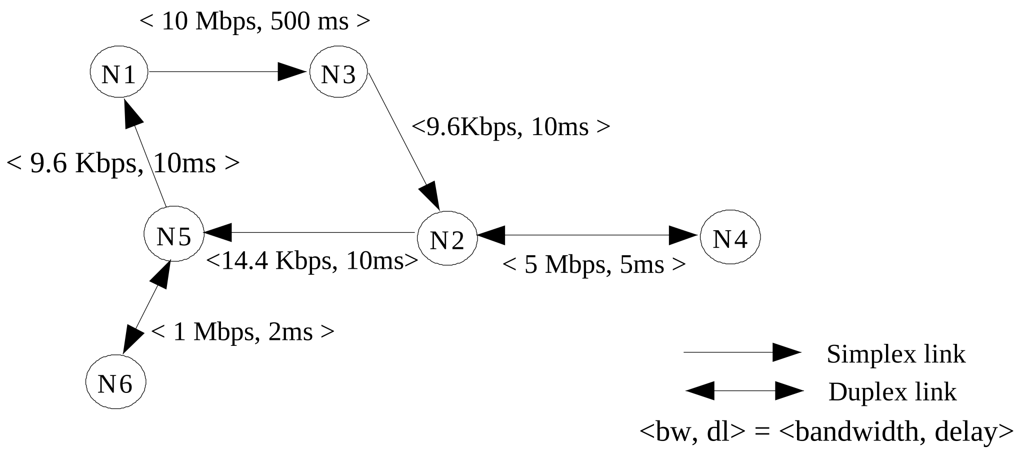

Figure 1: *Mild Aggression simulation setup*

two TCP transfers simultaneously going on on the reverse channel i.e. the ACK transfer for one connection (connection under study) and data transfer for another connection. For example: A user uploading his page and other related data to a web server while simultaneously receiving other data or downloading a file.

We compared *Mild Aggression* with Asymmetric TCP [6] for one−way and two−way transfers. We used *ns* [12] (network simulator) for our simulations, including modules added by [6] for Asymmetric TCP. The topology set−up for our experiments is shown in Figure 1. Both one−way and two−way transfers use the simulation topology of Figure 1.

In Figure 1, nodes N1, N4 are constant bit−rate sources and nodes N3, N6 are sinks. Nodes N2, N5 are intermediate nodes. The nodes N4 and N6 are activated only for two−way transfers.

One−Way Transfers

In this case the <N1,N3> is high bandwidth forward channel. <N3,N2,N5,N1> is low bandwidth reverse channel. The nodes N4, N6 are not participating in any transfer in one−way transfer .

Two−Way Transfers

In two−way transfers, the first transfer is from N1 to N3. <N1, N3> is the high bandwidth forward channel. <N3,N2,N5,N1> is low bandwidth reverse channel. Simultaneously, a second transfer is going on from node N4 to N6 passing through nodes N2, N5. N4 sends data packets on the path N4−N2−N5−N6 at 2, 6 and 10 seconds of sizes 2048 bytes, 4096 bytes, 2048 bytes respectively. This causes ACK flow disruption due to contention on the link N2−N5 between the data packets and ACKs.

The simulations for comparing performance of Asymmetric TCP versus *Mild Aggression* were conducted at N1. The experiments were conducted with initial window = 1 and with $\theta = 0.8$, $\theta = 0.6$ and $\theta = 0.5$. *cwnd* was incremented by ε, taken as 1+1/*cwnd,* during *slow start.* This was done so to reduce the effect of ε as *cwnd* increased to a sufficiently large value. The results of the simulations are shown in the figures 2,3,4 for 1−way transfer and figure 5,6,7 for 2−way transfers.

4.2 Observations

One−Way Transfers

Figure 2,3,4 shows the variation of congestion window (*cwnd*), round trip time (*rtt)* and sequence number v/s time. *Mild Aggression,* as expected, shows better performance as compared to Asymmetric TCP. This is because the use of *Mild Aggression Factors* improves the *rtt* estimate and *cwnd growth.*

Two−Way Transfers

Figure 5. shows the *cwnd* growth v/s time. In the case of two−way transfers, for $\theta \leq 0.7$ the reduction in *rtt* forces a early time−out resulting in *cwnd* collapse. The cross−over at ~12s i.e. where Asymmetric TCP shows better *cwnd* growth (slope) as compared to *Mild Aggression,* shall be investigated further in our future work.

Figure 6. shows the variation of *rtt* v/s time. The *rtt* variation is high for Asymmetric TCP since the reverse channel is affected by the intermittent data transfer from source N4 (see figure 1). *Mild Aggression* performance is affected in case of high aggression i.e. for low θ values, such as 0.5, the performance is affected since *grtt* makes a highly optimistic approximation of the forward channel characteristics and pumps data in form of heavy bursts.

Figure 7. shows the changes in sequence number v/s time. The trend is similar to that of congestion window growth.

As can be seen, for for low values of θ i.e. high aggression, the performance deteriorates. The optimum value of θ is expected to be 0.8 in practice. However, the exact value of θ will depend on the forward and reverse channel characteristics, the number of TCP senders and the their traffic.

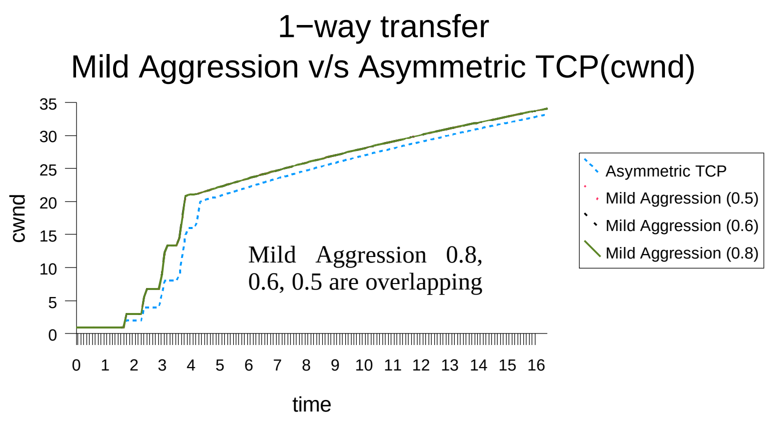

Figure 2*: Mild Aggression v/s Asymmetric TCP (cwnd growth for 1−way transfer)*

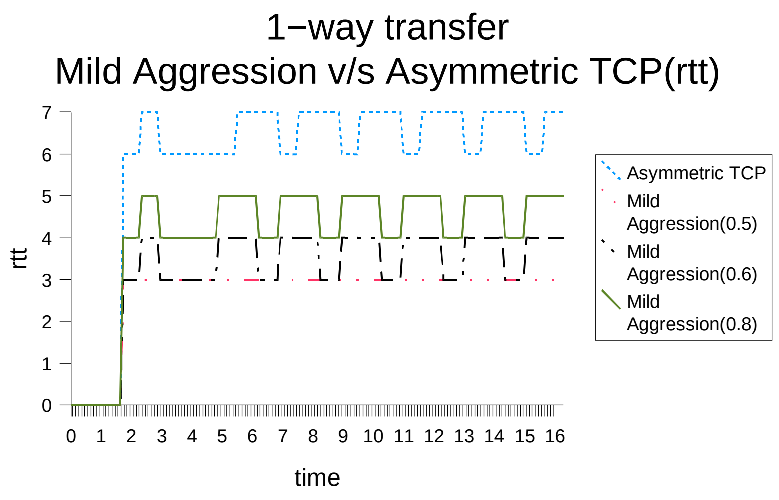

Figure 3*: Mild Aggression v/s Asymmetric TCP (rtt for 1−way transfer)*

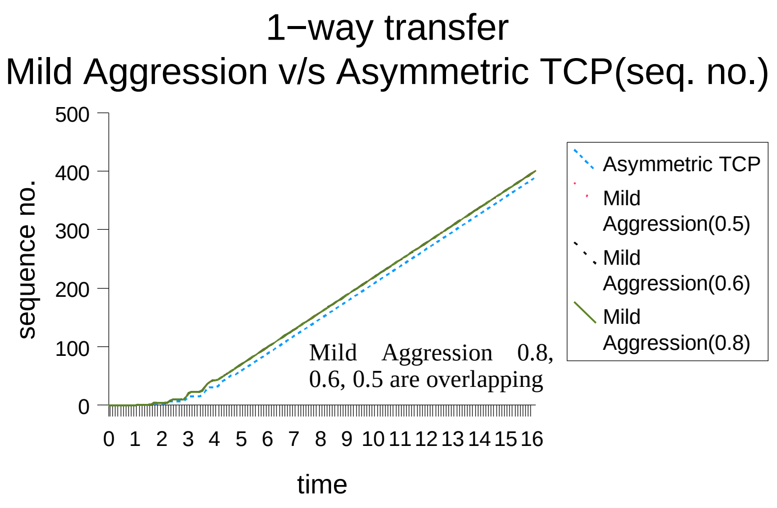

Figure 4*: Mild Aggression v/s Asymmetric TCP (sequence number for 1−way transfer)*

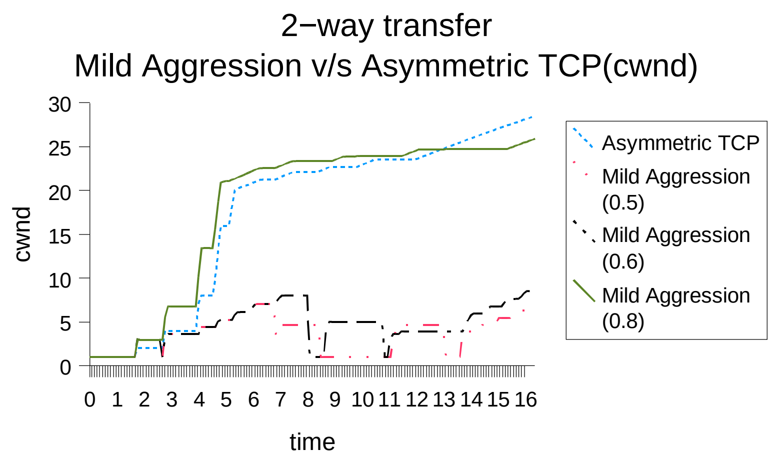

Figure 5*: Mild Aggression v/s Asymmetric TCP (cwnd growth for 2−way−transfer)*

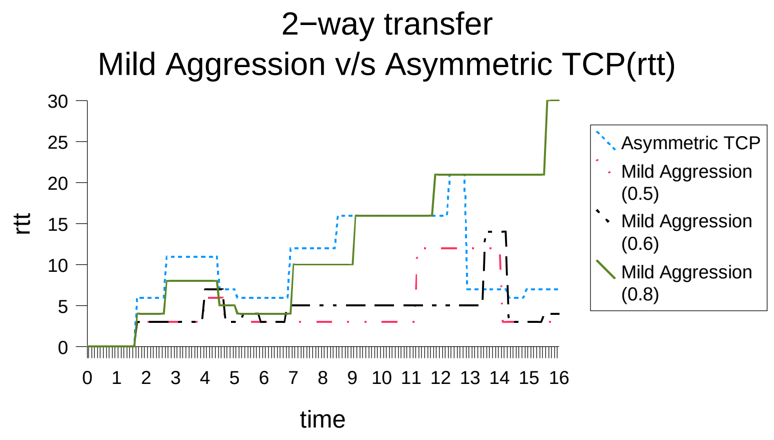

Figure 6*: Mild Aggression v/s Asymmetric TCP rtt for 2−way−transfer)*

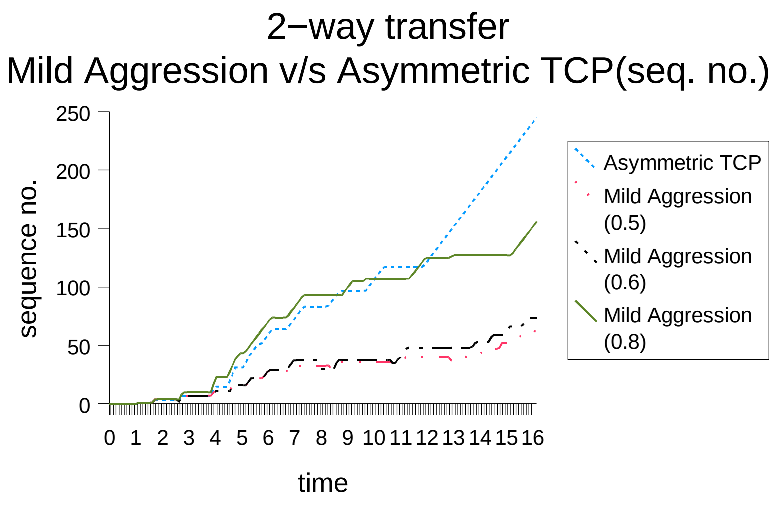

Figure 7*: Mild Aggression v/s Asymmetric TCP (sequence number for 2−way−transfer)*

5. Related Work

Attempting to reduce the number of ACKs as far as possible, is a natural solution to improve the throughput and performance of TCP in asymmetric networks. Other solutions include techniques to try to reduce the effect of ACK losses and prioritize the flow of ACKs over data. For a detailed discussion see [6] which describes methods like ACK Congestion Control(ACC), ACK Filtering (AF), ACK Reconstruction, ACK First Scheduling and Sender Adaptation. [11] proposes techniques similar to that in [6]. [1] proposes an end−to−end mechanism using sender window prediction to adapt the delayed−ack factor.

Satellite networks show asymmetry since the satellite broadcast channel has high bandwidth. Satellite Transport Protocol (STP) is presented in [8]. [8] also touches upon a number of other proposals for improving TCP performance in satellite networks.

Some observations regarding these techniques are as follows:

- ACK Congestion control (ACC)

ACC requires the setting of the ECN bit whenever the average queue size, at the router, exceeds some threshold. Whenever the packets (data as well as ACKs) are modified the old checksum is useless and hence a new checksum needs to be calculated at the router. ACC can result in slow down in the window growth, since the number of ACKs are decreased.

- ACK Filtering (AF)

In AF the router eliminates redundant ACKs, for a TCP connection, from its queue. This leads to overheads at the router. Also, the router needs to be TCP aware. AF results in slow down in the window growth, since the ACKs are eliminated.

- ACK Reconstruction (AR)

In AR the router maintains soft−state of a TCP connection, and determines whether any ACKs have been eliminated by AF/ACC or have been dropped. This router then inserts *new* ACKs to smooth out the ACK flow and thus avoid burstiness at the TCP sender. The topology of the reverse channel should be known beforehand.

- Scheduling data and ACKs

In this scheme the ACKs are given higher priority, compared to data packets, at the router and thus ACK delay is avoided.

- Satellite Transport Protocol (STP)

In this scheme the receiver sends a STAT (STATus) packet indicating the packets received, only when requested by the sender by a POLL packet. The sender adjusts its congestion window only after receiving a STAT or a USTAT (unsolicited STAT).

- Window Prediction Mechanism (WPM) [1]

WPM is an end−to−end mechanism wherein the receiver adapts the delayed−ack factor, based on the estimation of the sender congestion window and the allowed window against the actual number of packets received, to take into account the packet losses. The receiver reconstructs the congestion control behaviour of the TCP sender.

The schemes ACC,AF,AF and Scheduling data & ACKs result in an overhead at the router and also require the router to be TCP aware. Also, these schemes are not effective if the traffic is encrypted or the end host is mobile having a changing IP address. All the above mechanisms, including STP and WPM, do not utilize the high bandwidth of the forward channel for an early ramp−up of *cwnd.*

Mild Aggression(MA) has the following advantages:

1. There is an early ramp−up of the *cwnd*, as the *cwnd* will be updated per *grtt*, which is smaller value than normal RTT.
2. Only the end hosts are required to be modified. The intermediate routers need not be modified. Thus *Mild Aggression* is totally transparent to routers.
3. *Mild Aggression* will perform well even if the packets are encrypted because the scheme is on an end−to−end basis.
4. The scheme will work well even if the host is mobile and it's IP address changes frequently. Since *MA* is implemented only at the source and destination host, irrespective of the changes in IP addresses of the host, the data packets and ACKs will be correctly delivered to the hosts and *MA* will perform correctly.

6. Conclusions

We have presented a *Mild Aggression* approach for improving the performance of TCP in asymmetric networks. The *mild aggression factors* can be used to improve the initial sluggish growth of the *cwnd* in an asymmetric environment. The *mild aggression factors* are used to estimate an optimistic RTT which is used for *cwnd* growth and sending out data packets. Our approach does not require modifications at intermediate routers and is applicable even if the packets are encrypted.

Simulations for comparing *Mild Aggression (*using *goodRTT)* with respect to Asymmetric TCP indicate positive results. The one−way transfer simulation demonstrates the validity of the *Mild Aggression* approach. We have shown that the performance of TCP can be improved significantly by choosing the *MAF*s appropriately. The two−way transfers simulation further demonstrates how *Mild Aggression* will be useful in real world scenarios, where multiple simultaneous transfers are common. Thus *Mild Aggression* is likely to perform well in asymmetric environments.

As θ tends to 1 *Mild Aggression* tends to Asymmetric TCP. For $\theta = 0$, $grtt = 0$ resulting in repetitive time−outs prohibiting *cwnd* growth. In actual asymmetric environments, the value of θ would typically be around 0.8. Note that the aggression factors θ and ε are chosen by inspecting the network conditions and overly optimistic values of θ and ε can lead to packet losses.

Simulations to study the effect of the Mild Aggression factor ε on *cwnd* growth and efforts to dynamically adapt the aggression factors θ and ε to the changes in network conditions are underway .

Acknowledgment

We thank Suresh Pendhap, Graduate student, for his contribution in the preliminary discussions on Mild Aggression.

References

[1] Anna Calveras Augé, Jaume Linares Magnet, Josep Paradells Aspas; *Window Prediction Mechanism for Improving TCP in Wireless Asymmetric Links;* Proceedings of Globecom'98; November 1998

[2] M. Allman, D. Glover, L. Sanchez; *RFC 2488: Enhancing TCP Over Satellite Channels using Standard Mechanisms;* Jan 1999.

[3] M. Allman, et al; *RFC 2760: Ongoing TCP Research Related to Satellites*; Feb 2000.

[4] M. Allman, S. Floyd, C. Partridge; *RFC 2414 Increasing TCP's Initial Window September 1998*; Sep 1998.

[5] M. Allman, V. Paxson, W. Stevens; *RFC 2581: TCP Congestion Control*; April 1999.

[6] Hari Balakrishnan, Venkata N. Padmanabhan, Randy H. Katz; *The Effects of Asymmetry on TCP Performance*; ACM Mobile Networks and Application s (MONET) 1998.

[7] S. Biaz and N. H. Vaidya; *Using End−to−End Statistics to Distinguish Congestion and Corruption Losses: A Negative Result*; Technical Report 97− 009, Computer Science, Texas A&M Univ., August 1997.

[8] Thomas R. Henderson, Randy Katz; *Transport Protocols for Internet−Compatible Satellite Networks*; IEEE Journal on Selected Areas in Communication, Special Issue on "Direct−to−user satellite system & technologies at ka−band and beyond"; Feb 1999, Vol. 17, No.2, 326−44.

[9] S. Keshav.; *An Engineering Approach to Computer Networking: ATM Networks, The Internet and the Telephone Network*; 1997, Addison Wesley.

[10] T.V. Lakshman, U. Madhow, B.Suter, *Window Based Error Recovery and Flow Control with a Slow Acknowledgment Channel: A Study of TCP/IP Performance,* Proceedings of Infocom '97, April 1997.

[11] Nikhil K. G. Samaraveera. *Return Link Optimization for Internet Service Provision Using DVB−S Networks*; ACM SIGCOMM Computer Communication Review, Volume 29, No.3, July 1999.

[12] UCB/LBNL/VINT Network Simulator; *ns(version 2.1b6);* http://www−mash.cs.berkeley.edu/ns/