Microscopic Examination of TCP flows over transatlantic Links

Antony Antony [a], Johan Blom [b], Cees de Laat [b], Jason Lee [b], Wim Sjouw [b]

[a] *NIKHEF, 409 Kruislaan, 1098 SJ Amsterdam, The Netherlands*

[b] *Universteit van Amsterdam, 403 Kruislaan , 1098 SJ Amsterdam, The Netherlands*

.

Abstract

Much of the recent research and development in the area of high speed TCP is focused on the steady state behavior of TCP flows. However, our experience with the first research only transatlantic 2.5 Gbps Lambda link clearly demonstrates the need to focus on the initial stages of TCP. The work we present here examines the behavior of TCP flows at microscopic level over high bandwidth long delay networks. This examination has led us to study the influence of the minute properties of the underlying network on bursty protocols such as TCP at these very high speeds combined with high latency. From this study we briefly describe the requirements for such an extreme network environment to support high speed TCP flows. We also present results collected from using transatlantic links at iGrid2002 where we tuned various host parameters and used modified TCP stacks.

Key words: HSTCP, High Speed Long latency TCP, iGrid2002, Long Fat Networks

1 Introduction

Grid applications in general can be demanding in terms of bandwidth requirements. A typical large scale scientific experiment involves at least two parties; a data producer and a data consumer. High Energy Physics (HEP) experiments such as LHC[17], D0 [16] and EDG[6] are good examples of this. In these projects the data is mostly produced at a few large experiment locations (the data producers) and many researchers analyze this data at their home institutes and universities (the data consumers). Often the researchers are geographically separated by large

Email addresses: antony@nikhef.nl (Antony Antony), jblom@science.uva.nl (Johan Blom), delaat@science.uva.nl (Cees de Laat), jason@nikhef.nl (Jason Lee), wsjouw@science.uva.nl (Wim Sjouw).

distances. The throughput requirements for these experiments are high. For example, EDG roughly estimates its network requirement at an average sustained flow of 400 Mbps for a duration of four months out of a year. Not only needs this data to be collected from the experiment but a large part of it needs to be transferred to various locations over the network. Network architectures are evolving to meet this unprecedented demand. Here we present research on the scalability of protocols to allow these kinds of applications to reach the required high bandwidths on new network architectures. One way to provide this bandwidth is by provisioning end-to-end paths, called Lambdas [15], up to several Gbps. In the fall of 2001 SURFnet[11] provisioned a 2.5 Gbps Lambda between Amsterdam and Chicago to be used only for research. Initial tests showed that only increasing the speed of links, switches and routers in the path was not sufficient to obtain a thruput at or near the available bandwidth. We conducted extensive experiments on this transatlantic link both to understand how transport protocols behave and what additional requirements high speed flows, such as TCP, impose on these kinds of networks. One architectural shift in our high-speed networking experiments is to minimize the number of routers (devices which process packets at layer 3 and above), and instead delegate packet forwarding to switching devices at layer 2 or below. Initial throughput measurements of a single TCP stream over such an extreme network infrastructure (i.e. the SURFnet Lambda) showed surprisingly poor results. This led us to study the dynamics of TCP at the microscopic level to better understand its behavior. The primary motivation of this work is the demand from HEP community to get maximum throughput over long distance links using single or few TCP streams. Currently the HEP community can not effectively use the available bandwidth. Simply increasing network capacity does not improve end-to-end performance. The exclusive availability of the SURFnet lambda for research has allowed us to investigate this problem.

The performance problems of TCP/IP for large data transfers over high bandwidth long latency path is a known problem [18]. The problem is to discover the bottleneck of a TCP flow (the slowest link in a chain of networks) between two work stations connected using a long latency high bandwidth path. There are several issues related to this problem: network characteristics (router, switches, slow links), the implementation of the TCP stack and specific parameters passed to the TCP algorithm at the hosts. In section 2 we examine the characteristics of the equipment and how this influences TCP and what requirements TCP imposes on the network. Section 3 briefly discusses some of the problems with the TCP algorithm on high bandwidth long delay paths. In section 4 we discuss the effect of host and operating system parameter tuning on performance. Section 5 shows test results from different modifications to the TCP/IP algorithms and particular those of HSTCP [3] implemented by

the Net100[4] project.

In the following sections we broadly classify the various stages of a TCP session into: Bandwidth Discovery (BD) phase (aka slow start), steady state [3], and congestion avoidance. In this work we focus mostly on the initial phase of a TCP flow, the BD, as we believe that this the most influential phase on the obtained bandwidth of TCP.

2 Properties of underlying network infrastructure

The first used configuration around the SURFnet Lambda (2.5 Gbps) is shown in the figure 1. Two high-end Work Station's (WS) were connected using Gigabit Ethernet via two Time Division Multiplexer (TDM) switches and a router. The TDM switches are capable of encapsulating Ethernet packets in SONET frames up to the rate of the specific SONET channel. The hosts were connected at 1 Gbps to a first version of the TDM switch. The linecard to backplane interface posed a 622 Mbps limitation to the datapath. This was due to a limitation of the linecard to backplane interface. In a subsequent version of that switch this bottleneck was alleviated. The network had a Round Trip Time (RTT) of about 100 ms and thus a very high bandwidth delay product. Initial TCP tests conducted over this link between Amsterdam and Chicago with the first version TDM switch in place showed rather poor results. Throughput obtained using a single

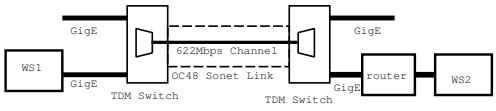

Fig. 1. Initial network setup. Two hosts connected back via two TDM switches interconnected at OC48 Link (96 msec RTT), sub channeled into an OC12.

stream TCP session was an order of magnitude less than the bottleneck capacity. Tuning the TCP stack showed only marginal improvements. On the other hand a multi-stream TCP session between the same two hosts achieved a throughput of about 80% of the bottleneck capacity. Also a UDP application (`iperf`) obtained a throughput a little higher than that of a multi-stream TCP session. Note that this path was exclusively used for research so there was no influence of background traffic or load on switches. Before shipping one TDM switch to Chicago we had also tested this setup locally (back to back) with a negligible round trip time and the throughputs were also close to linespeed.

The approach we took to understand the performance problem was to examine TCP at microscopic level. A quick look showed that TCP is bursty during the initial phase. We assumed that the TDM switch could not cope with big bursts and therefore loose packets. Since there is an intrinsic bottleneck of 622 Mbps in the TDM switch and the WS is connected at higher speed than the bottleneck, it is evident that host will be able to overflow the switch's memory on the input linecard. In the tested configurations (hardware) flow control was not operational. This is discussed in

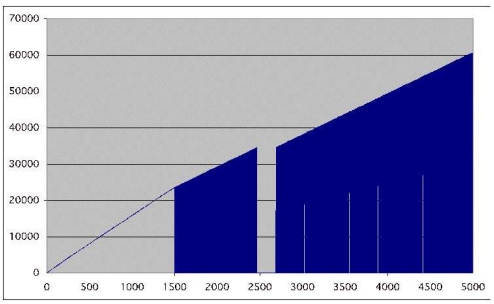

Fig. 2. relative arrival time (in μsec's) at the receiver vs packet number, for 5000 UDP packets send using UDPMon, from Amsterdam to Chicago

great detail in section 3. For the rest of this section we used streams of UDP packets to simulate the behavior of a TCP burst during the initial phase.

In order to estimate the maximum possible burstsize which does not cause packetloss in the switch, we used a tunable UDP stream. The assumption here is that the burst is similar to what occurs in a TCP flow during its initial phase.

Our setup is shown in figure 1. Two WS's, one configured as sender (Amsterdam) and the other configured as receiver (Chicago), were connected to TDM switches using a Gigabit Ethernet link. The switches were interconnected using a high bandwidth delay product path. Round Trip Time (RTT) of the link was about 100 msec and the provisioned capacity of the link was 622 Mbps. If the sender sends a continuous stream of packets as fast as it can (about 900 Mbps, limited by WS overhead) eventually a fraction of the packets will be dropped at the 622 Mbps bottleneck.

Using `UDPMon`[14], we send 5000 numbered UDP packets of size 1000 bytes were sent as fast as possible. Figure 2 shows the result. Horizontally is the packet number, vertically its arrival time. Dropped packets get an arrival time of zero. Therefore, the shaded area under the curve indicates lost packets. First loss occurs after 1500 packets. From then on approximately one out of every three packets is dropped. This points at the bottleneck mentioned before since the ratio of dropped packets agrees with the bandwidth ratio. The curve also shows that a continous block of about 150 packets is lost after this point. We assumed that these packets are dropped at the receiver. Studying the packet counters of the switches in the path supported this assumption. We believe that this is due the limitations of the receiver. The receiving WS is overwhelmed by the rate and it drops a series of packets from input buffer. We assume packets are dropped while it is being copied from Network Interface Card (NIC) memory to receiving process (UDP in kernel space) memory. A similar kind of receiver limitation is also discussed in section 3.2. Rest of the section we focus on the packet loss caused by intrinsic bottleneck, i.e. sender side TDM switch.

The number of packets dropped (N_d) by the switch during short periods of burst is related to the number of packets in the burst (N_b), incoming fast speed (f), outgoing slow speed (s), and buffer memory (M) available at the output port of the bottleneck link. We assume for simplicity an average packet length (l). The loss can

expressed as:

$$N_d = N_b \frac{(f-s)}{f} - \frac{M}{l} \quad (1)$$

Using equation 1, where we set $N_d = 0$ and $N_b = 1500$ as that is the maximum burst which got through, we computed the available memory on TDM switch as approximately 0.5 MBytes.

Once we know the memory at the bottleneck and burst of packets than pass through the TDM switch we can calculate the possible bandwidth a TCP flow can achieve during the initial phase with out packet loss. We assume there are no other bottlenecks in the end-to-end path and TCP has not encountered a congestion event.

To first order the throughput TCP can obtain is then approximated by

$$B = \frac{f}{(f-s)} \frac{M}{R} \quad (2)$$

where: B is the throughput TCP flow can achieve. R is the round trip time.

If we assume that TCP ideally tries to get to a stable situation where the throughput is the slow link speed, we can substitute B=s in equation 2. This leads to a memory requirement to support a high bandwidth delay product TCP flow as

$$M = \frac{(f-s)}{f} sR \quad (3)$$

For the network as shown in figure 1 to support a 622 Mbps end-to-end TCP flow minimum memory required is 3.1 MBytes ($f = 1Gbps, s = 622, R = 100msec$). From our understanding currently available Ethernet to TDM encapsulation devices don't have the required memory. Also from preliminary discussions with a few vendors, we understand these devices are not designed for high bandwidth delay networks. Instead they were designed for LAN and metropolitan area networks where the RTT is only a few milliseconds. Even in such cases, if the device is up-linked to a worldwide infrastructure and if the TCP flow passing through sees a bottleneck at a switch, all of the above applies.

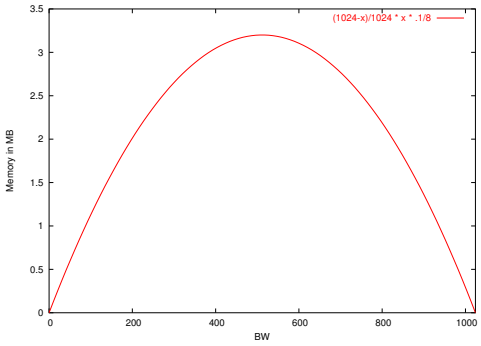

Fig. 3. Required memory at a bottleneck for an incoming speed of 1 Gbps and various output speeds for a RTT of 100 ms.

Figure 3 shows the memory required to support different end-to-end speeds of TCP flow for a given RTT $R = 100$. If we solve the quadratic equation 3 for s we can compute TCP throughput vs RTT for a given values of M and f.

$$s = \frac{f}{2}\left(1 + \sqrt{\frac{1-4M}{FR}}\right) \quad (4)$$

The result is plotted in figure 3.

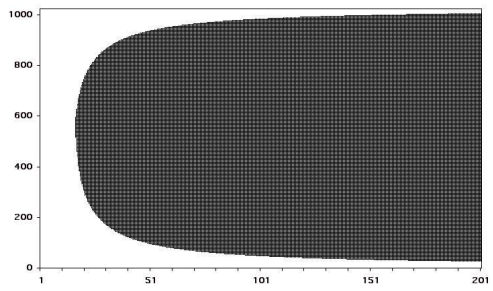

Fig. 4. Forbidden (shaded) area where packet loss may occur in single stream TCP flows for a given memory size of 0.5 MByte and an incoming speed of 1 Gbps. Horizontally is the round trip time of the desired destination and vertically the provisioned "slow" speed at a TDM switch.

For example using our TDM switch for a 150 ms RTT (Amsterdam to California) link the end-to-end throughput will be about 45 Mbps. The area inside the curve is a *forbidden* area for TCP flows as packet loss may occur. the values we get using this formula also matched with the throughput obtained TCP tests using iperf.

3 TCP

TCP is a sender controlled sliding *window* protocol [10]. New data up to "window size" is sent when old data has been acknowledged by the receiver. The window size is limited by the host and application parameters such as socket buffer size and $Cwnd$ [1]. TCP adjusts the $Cwnd$ dynamically using different algorithms depending on which phase the flow is currently in. We will focus here on understanding the slow start phase. We tested modifications to

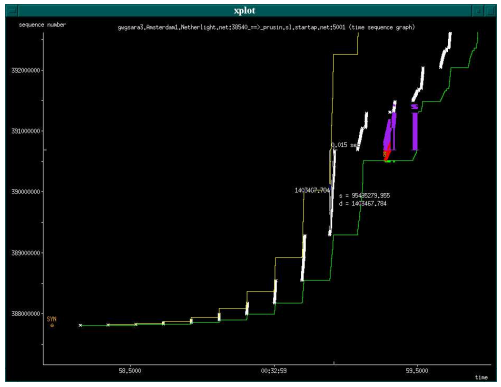

Fig. 5. Time sequence graph showing initial phase and congestion event after 10 RTTs

the congestion avoidance algorithm.

3.1 Bandwidth discovery phase (slow start)

This is the initial phase of a TCP flow. After the protocol handshake [8] the sender will try to discover what the available bandwidth is to compute accurately the correct value for $Cwnd$. This is done by injecting data into the network till a congestion event occurs. Fast convergence and accuracy of bandwidth discovery has a large influence on all three phases of TCP. This determines how fast a flow can reach steady state and the stability once it has reached steady state.

If during this initial phase no congestion events are generated, the $Cwnd$ effectively doubles every RTT. Thus ideally a flow should only be limited by an intrinsic bottle neck (i.e. packet loss). If the limiting bandwidth between two hosts is the speed of the sending host (i.e. slow NIC, slow CPU) then the BD phase will

always work correctly. However, if the connection between the hosts is limited by some other factor (i.e. router buffer, network capacity, etc) then the BD phase will fail due to the fact that the doubling of the congestion window can overrun the bottleneck by a large number of packets [2], thus causing large packet loss. The packet loss can be computed using equation 1 if we know the bottleneck speed and buffersize at the bottleneck. Limited slow start is a good solution to avoid buffer over run problems during BD. Limited slow start just stops doubling the congestion window after the window reaches a predefined threshold. After the congestion window reaches the threshold, it continues to open up, but at slower rate that is based on the size of the congestion window. This stops TCP from overshooting the bottleneck buffer by a large margin, and reduces the number of packets lost that occurs when it does overflow the bottleneck buffer.

In figure 5 we show our observation of large packet loss caused by a bottleneck. After 9 RTT 512 packets leave WS at 10 RTT it is doubled and this overruns the buffer causing large packet loss.

The HSTCP extension may be an alternative. We believe that proposed algorithm [3] is a good starting point. On an over-provisioned network one could have a faster algorithm to increase *Cwnd*. The requirement of such an algorithm is to do BW discovery as fast as possible with minimum or no packet loss.

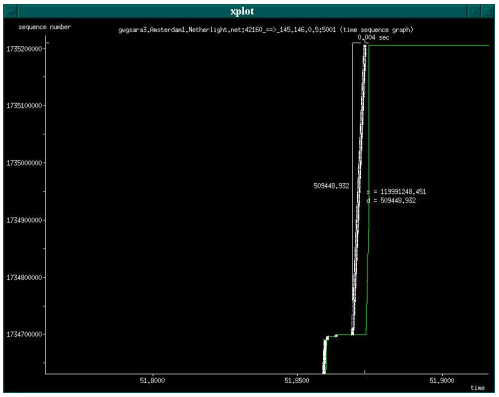

Fig. 6. Time Sequence graph showing instantaneous speed of flow

3.2 Receiver Limitations

Typically most PC Work Station (WS) hardware exhibits the property that the receiver capacity is less than the sender for an identically configured WS. This is due to the difference in overhead of sending a packet vs receiving a packet. Take, for example, two identical WS connected back-to-back, one configured as as the sender and other as the receiver. If the sender sends data as fast as it can, the receiver may not be able to keep up. When receiver is overloaded in this manner, it will drop packet and cause a congestion event.

Figure 6 shows the instantaneous speed of a flow during slow start using Time Sequence Graph (TSG). Notice that after 8 RTT's 512 packets leave the host. The sending host sends this data as IP packets as fast as it can. In this case the 512 packets are send in about 4 ms yielding an instantaneous speed close to 1 Gbps. This overruns the receiver buffer and causes the flow to fall out of the BD phase into congestion avoidance phase. Therefore, this case is similar

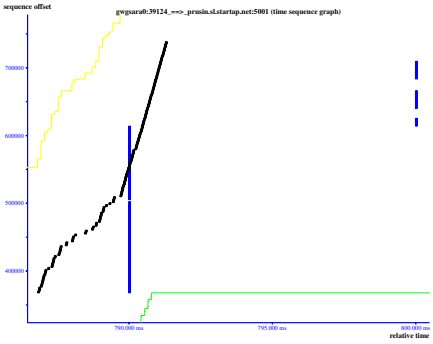

Fig. 7. combined traces from sender and receiver

to that of a buffer overflow at the bottleneck as discussed in the previous subsection. A solution would be to pace the packets such that the average speed approximately equals the bottleneck speed.

Figure 7 shows the combined time sequence graph of packets leaving sender and receiver. It clearly shows that the inter packet delay is very small at the receiving process. This may be due to effects of interrupt coalescing.

4 Host Parameters

Implementations of the TCP algorithms vary between OS's. The behavior of TCP depends on the particular implementation and architecture of the WS, such as host bus speed, devices sharing the bus, Network Interface Card (NIC), interrupt coalescing, inter packet delay, [14] etc. Thus using the same values as described in [9] on two different con-

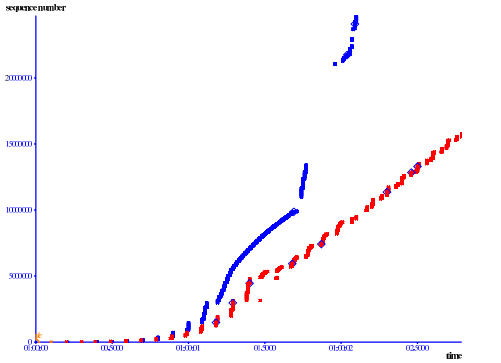

Fig. 8. TSG comparing initial phase of Linux and Mac OSX (red is Linux, and blue is Mac OS X).

figurations still can produce varying results, especially during the BD. These differences may become less noticeable if we average these values over long periods of time.

We refer to values specific to a configuration of a workstation as *the host parameters*. This also includes the TCP implementation. From our experience it has been observed that some seemingly slower hosts, in terms of CPU and bus speed, are not necessarily the slowest for TCP transfers. We assume this is due the fact that the slower hosts pace out the packets better than a faster WS, hence there is less chance for overflowing bottleneck queues in the path. The TSG in figure 8 shows a comparison between Mac OS X and linux 2.4.19. The data was captured at the receiver side using `tcpdump`.

4.1 Results from tuning TX queue length

Tuning the length of the Transmit Queue (TXQ) of the sending de-

vice had noticable effect on high bandwidth high delay path. This parameter can be adjusted using the UNIX command `ifconfig` option `txqueuelen <length>`, though one should keep in mind that the device is limited by the amount of available memory. We found that even though tuning of this variable can improve the performance of TCP by several factors, results are not very predictable and there doesn't seem to be an easy way to precompute what the length should be. Figure 9 shows the results of testing throughput over a Long Fat Network (LFN) with several hundred different queue lengths. The default queue length is around 400 packets, while, as shown in the graph, we continued to get increased performance till about 1500 packets. Having a large transmit queue is very helpful during the bandwidth discovery phase in absence of a congestion event since it allows one to reach the maximum throughput very quickly. If a congestion event does happen, the flow will fall back into the congestion avoidance phase. This is the same as if it was in steady state. The stream will then act like a TCP over a long latency link and will take many RTT's to recover.

In Figure 9 it can be seen that through adjusting the transmit queue one can clearly obtain improved throughput, but the throughput is not always very predictable. By monitoring the Web100 variable `|BytesRetrans|`during the tests we tried to identify packet loss to see if a failure of BW discovery phase was occurring. This was done to give indication if the oscillations in

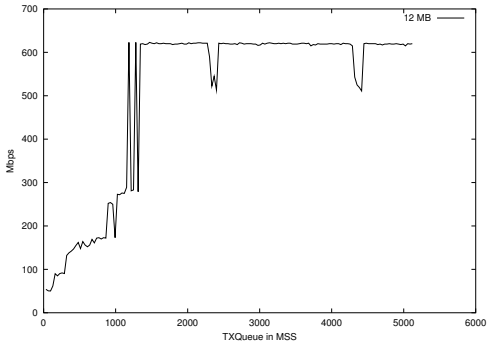

Fig. 9. TXQ in MSS vs throughput in Mbps. Steps used 32. using Net100 kernel capable of AIMD

throughput were due to packet retransmission. However, it turned out that there were no retransmissions during the tests. We now assume that the variances may be due to the dynamics of TXQ which causes an early congestion event (i.e. premature end of bandwidth discovery phase) and low throughput.

Net100 has implemented a workaround in the Linux TCP implementation to obtain a similar effect as tuning the transmit queue. This is done by brute force, where the influence of TXQ on TCP $Cwnd$ computation is removed from the TCP stack. Results using these modifications are show in Figure 12.

To test if host behavior varies between architectures a few tests were done using an Apple laptop (used as sender) connected at 1 Gbps. Initial results look very promising. One could get up to 354 Mbps between Amsterdam and Chicago. A closer look at the traces captured from the receiver is shown in figure 8. It clearly shows that the Apple hosts behaves differently compared to the Linux host during BD phase of TCP.

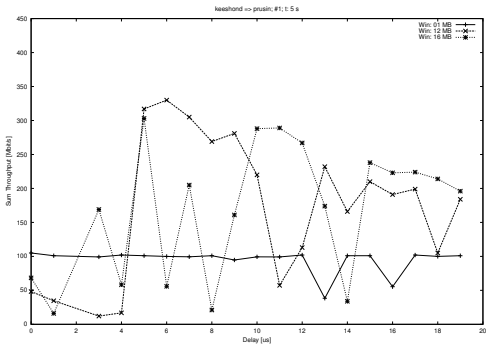

Fig. 10. Bandwidth vs. delay using iperf for duration 5 sec for varying socket sizes

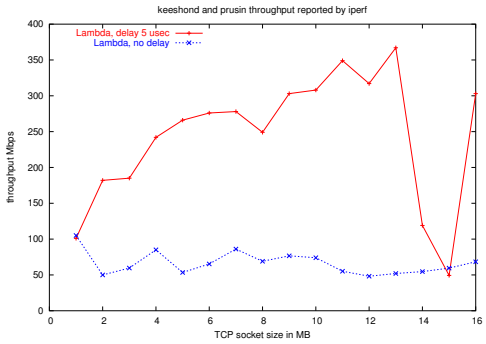

Fig. 11. Bandwidth vs socket buffer size using iperf, with delay (5 μ sec) and without delay

Apparently the OSX on the Apple paces the packets better than Linux, putting a larger inter packet delay between the packets. This could be due to slightly different implementation of TCP or differences in hardware architecture such as the NIC, motherboard architecture, CPU, etc.

5 Modifications to TCP/IP algorithm

5.1 Pacing out packets at device level

From the discussions in the preceding sections it is clear that some sort of pacing of the packets should improve performance. We implemented a delay at the device driver level, i.e. a blocking delay of O(μ)secs. Results are shown in Figure 10 and 11.

From the results we conclude that the sender should not burst but try to shape according to a leaky bucket algorithm. Though this may be hard to implement in the OS since it requires that the OS maintains a timer per TCP flow with μsec resolution, which could incur lots of overhead. Our initial suggestion is that future OS kernels should delegate the task of pacing the packets to NIC's and allow the NIC to implement this feature at the hardware level.

5.2 HSTCP modification

The HSTCP modifications [3] discussed here are still in the development stage. Most HSTCP extensions are aimed at improving performance for steady state congestion avoidance, yet it appears that the BW discovery phase may also benefit indirectly from this work. This is because if the BW discovery phase ends prematurely (i.e. before full utilization of the resources), the bottleneck utilization will be low, and then HSTCP modifications will improve utilization by ramping back up faster then traditional TCP algorithms. From the discussions above it is clear that in many cases initial stages end prematurely before it completes bandwidth discovery. Using HSTCP modifications *Cwnd*

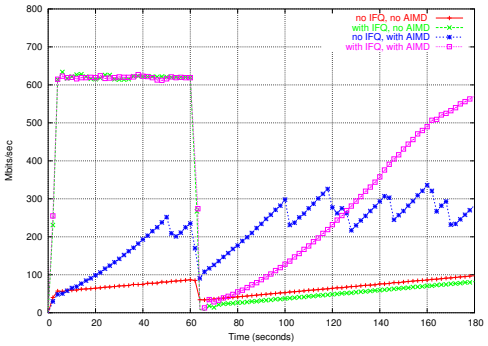

Fig. 12. BW vs time using HSTCP and IFQ modifications using Net100 kernel between Amsterdam and EVL, Chicago. Congestion was introduced at 60 sec by overloading receiver

increases slower than doubling. In effect TCP flow continues to discover bandwidth more quickly.

Figure 12 shows the results using HSTCP. We have run an `iperf` session for 180 seconds and created an congestion event at the receiver after 60 sec. Congestion was created by sending approximately 800 Mbps UDP from another host to the receiver. In the first case (red line) no IFQ modifications and no AIMD [3] modifications were enabled. Due to the short transfer queue length the flow exits the BD phase very early, at about 80 Mbps, and then continue in steady state where it increases $Cwnd$ at a rate of 1 MSS per RTT. After 60 sec the flow encounters the induced congestion and drops to about 40 MBps to continue to recover from it at the rate of one MSS per RTT. In the second (green line) case the IFQ control using the Net100 modifications turn off the transmit queue congestion detection of the NIC. This improves the BD phase, which now quickly enters high-speed steady state at about 618 Mbps. After the induced congestion event at 60 seconds it enters normal congestion avoidance phase drops down to about 35 Mbps and recovers it at the rate of 1 MSS per RTT. In the third case (pink) IFQ control using Net100 was again used to disable the transmit queue congestion detection of the NIC and AIMD modifications using the Net100 software were turned on. The BD phase is same as previous test. After the induced congestion event at 60 sec the flow drops to about 40 Mbps. While recovering from this AIMD comes into effect causing $Cwnd$ to increase based on the factor AI computed using values specified in [3]. Effectively the flow recovers from congestion event much quicker.

The fourth and last case (blue) was with the transmit queue congestion detection on the NIC enabled and AIMD turned on. Again the BW discovery phase ends prematurely as in case one. Since AIMD is active and there are no more congestion events the $Cwnd$ increases by the factor specified by AI. Note that there are some dips in the curve. Our understanding is that this is due to the poor default interface queue management.

HSTCP modifications and better control of IFQ clearly improves available BW utilization. Figure 13 shows the BW utilization (2 seconds average) of a long running running (3000 seconds) flow between Amsterdam and Chicago, USA. This test was running over a 622 Mbps transatlantic VPN with a background traffic at about 35 Mbps. Cumulative average

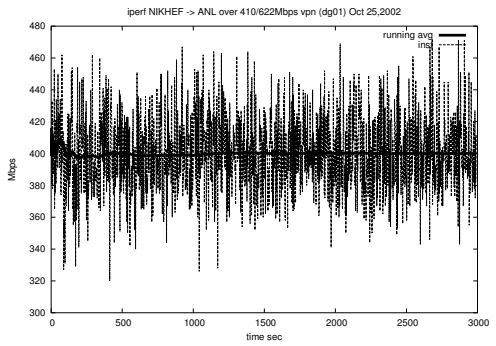

Fig. 13. BW vs time for a long term iperf session between NIKHEF and ANL over 622Mbps VPN

is about 405 Mbps, while the 2 second average oscillates between 320 to 460 Mbps showing that HSTCP improves the utilization by responding to congestion events faster than the non AIMD enabled TCP.

6 Conclusion

We have shown that tuning the host parameters and HSTCP is very important to make good use of available bandwidth over high bandwidth long delay networks. The maximum throughput obtained over the Trans-Atlantic link (96 msec RTT) was 730 Mbps using single TCP stream. Initial tests show that these modifications don't adversely affect other flows, but that still needs closer examination in an environment with a large number of heterogeneous flows. Also we have shown that for the current Linux TCP implementation the specifications of the underlying network infrastructure in terms of (artificial) bottlenecks, provisioned long haul forwarding paths, queue lengths and shaping properties define the upper limit of the single stream throughput no matter how well tuned the host parameters are.

7 Acknowledgments

The transatlantic links used for conducting this research are provided to us by SURFnet, TYCO and LEVEL3. Antony Antony and Hans Blom are funded by the IST Program of the European Union (grant IST-2001-32459) via the DataTAG project. Jason Lee was supported in part by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. The authors would like to thank Richard Hughes Jones, Brian Tierney and the Net100/WEB100 collaboration for their instrumenting TCP with modifications. We especially thank the iGrid2002 organizers for providing an excellent and exciting test bed.

8 References

References

[1] M. Allman, *et al.*, "TCP Congestion Control", http://www.ietf.org/rfc/rfc2581.txt

[2] Sally Floyd, "Limited Slow-Start for TCP with Large Congestion Windows", http://www.icir.org/floyd/hstcp.html

[3] S. Floyd, S. Ratnasamy, and S. Shenker, "Modifying TCP's Congestion Control for High Speeds", http://www.icir.org/floyd/hstcp.html 2001

[4] Net100: http://www.net100.org

[5] WEB100: http://www.web100.org

[6] EU DataGrid :http://www.eu-datagrid.org

[7] EU DataTag:http://www.datatag.org

[8] J. Postel,"Transmission Control Protocol", RFC 793, DARPA, September 1981.

[9] TCP Tuning Guide: http://www-didc.lbl.gov/TCP-tuning/

[10] Tom Dunigan, Matt Mathis, Brian Tierney, A TCP Tuning Daemon http://www.sc2002.org/paperpdfs/pap.pap151.pdf

[11] SURFnet 2.5 Gbps Lamda to Chicago, Press release http://www.surfnet.nl

[12] 10Gbps, (OC192) link to iGrid2002 http://www.startap.net/starlight/PUBLICATIONS/news-level3support.html

[13] iGrid2002 http://www.igrid2002.org/

[14] UDPMon :http://www.hep.man.ac.uk/ rich/net/

[15] Cees de Laat, Erik Radius, Steven Wallace, "The Rationale of Optical Networking", submitted for publication in FGCS special issue on iGrid2002, 2002.

[16] D http://www-d0.fnal.gov/

[17] LHC Home page: http://lhc-new-homepage.web.cern.ch/

[18] J. Lee, D. Gunter, B. Tierney, W. Allock, J. Bester, J.Bresnahan, S. Tuecke, "Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks", Proceedings of Computers in High Energy Physics 2001 (CHEP 2001), Beijing China, LBNL-46269.