Characterization and Evaluation of TCP and UDP-based Transport on Real Networks

R. Les Cottrell, Saad Ansari, Parakram Khandpur, Ruchi Gupta, Richard Hughes-Jones, Michael Chen, Larry McIntosh, Frank Leers

***Abstract*—Standard TCP (Reno TCP) does not perform well on fast long distance networks, due to its AIMD congestion control algorithm. In this paper we consider the effectiveness of various alternatives, in particular with respect to their applicability to a production environment. We then characterize and evaluate the achievable throughput, stability and intra-protocol fairness of different TCP stacks (Scalable, HSTCP, HTCP, Fast TCP, Reno, BICTCP, HSTCP-LP and LTCP) and a UDP based application level transport protocol (UDTv2) on both production and testbed networks. The characterization is made with respect to both the transient traffic (entry and exit of different streams) and the steady state traffic on production Academic and Research networks, using paths with RTTs differing by a factor of 10. We also report on measurements made with 10Gbits/sec NICs with and without TCP Offload Engines, on 10Gbits/s dedicated paths set up for SC2004.**

***Index Terms*—TCP, throughput, high-performance networking, UDT, TCP Offload Engine**

I. Introduction

HIGH Energy Physics (HEP) and other data intensive sciences have a growing need to share large volumes of data between computers and data centers distributed worldwide. Currently most bulk-data is transferred using applications based on TCP. The limitations of the standard (New-Reno based [1]) TCP Additive Increase Multiplicative Decrease (AIMD) algorithm for fast long distance networks have resulted in users by-passing the limitations by the use of multiple parallel TCP streams. Simultaneously optimizing the window size and number of streams is time consuming and complex and for some paths the optimum can vary within a few hours.

We have therefore installed and evaluated several new advanced TCP stacks to see how they compare with New-Reno based TCP stacks on production Academic and Research (A&R) networks with Gbits/s capacity paths. All these stacks require only the sender to be modified. We excluded using Dynamic Right Sizing [2] since it requires modifying the receiver hosts which were not under our control.

For us the important performance features are the achievable throughput, the support for the protocol (is it easy to install, is it kept up to date with the latest operating system releases/patches, is the author responsive etc.), the stability (i.e. how stable is the throughput as the network load changes), and the fairness.

The support issues for a production data-intensive science environment are critical, and may have little to do with the technical implementation. For example, at SLAC, the production operating system for most of the data movers is Solaris while most of today's advanced TCP protocol stacks have only been developed for Linux and so are not applicable. Further it is unclear that the production system administrators or the security people will want a modified non-vendor supported TCP stack on a production Internet connected host. Even if they do, they will probably require that the TCP stack patches keep pace with the operating system patches in place at the production site which may not be a goal for the protocol developer.

Therefore, we are also highly interested in bulk-data transfer mechanisms that do not require modifying the TCP stack. This is a major reason why the use of standard TCP with multiple parallel streams persists. An attractive alternative that we explore in this paper is to use a UDP based Data Transfer application such as UDT [3] which requires no system level changes as it runs entirely in user space.

Another approach of using large Maximum Transfer Units (MTUs) of over 1500Bytes has limited applicability except in testbeds in our case, since it is not an Ethernet standard, may interfere with some UDP based applications and is thus not supported on the SLAC Local Area Network (LAN).

Other practical considerations in achieving high throughput, on 10 Gbits/s testbeds and LANs, include system limitations such as bus bandwidth and cpu speed. These become critical as one tries to achieve throughputs of over 6-7 Gbits/s. Besides configuring to optimize the interrupt coalescence, the buffer sizes between the Network Interface Card (NIC) and the kernel, and procuring the fastest cpus and buses commonly available off the shelf; we are also interested in evaluating emerging techniques such as TCP Offload Engines (TOE). This promises to reduce the cpu utilization for a given transfer rate. However, it currently restricts one to using the NIC vendor's distributed TCP stack

Manuscript received January 21, 2004. This work was supported in part by the U.S. DOE Contract No. DE-AC03-76SF00515.

R. L. Cottrell, P. Khandpur and Ruchi Gupta are with the Stanford Linear Accelerator Center (SLAC), 2575 Sand Hill Road, Menlo Park, CA 94025 (phone: +1-926-2523; fax: +1-650-926-3329, email: {cottrell, parakram, ruchig}@slac.stanford.edu).

S. Ansari was with SLAC. He is now with Microsoft.

R. Hughes-Jones is with The Department of Physics and Astronomy, The University of Manchester, Manchester, England (email: r.hughes-jones@man.ac.uk).

M. Chen is with Chelsio, 370 San Aleso Avenue, Sunnyvale, CA 94085 (email: mike.chen@chelsio.com)

L. McIntosh and F. Leers are with Sun Microsystems Inc., 9515 Towne Center Drive, San Diego CA 92121 (email: {larry.mcintosh, frank.leers}@sun.com).

(usually New-Reno) which may lack the maturity of a host TCP stack, and also does not have the flexibility of a modifiable stack such as in Linux.

Finally we are also interested in the new TCP stacks that are or will be soon be available in standard distributions (in particular Solaris 10 and Linux 2.6), especially as they pertain to 10Gbits/s paths.

Section II describes the experimental setups for the A&R production network measurements and the SC2004[1] 10Gbits/s testbed paths. Section III describes the methodologies, Section IV gives the results, and Section V gives the conclusions.

II. Experimental Setups

A. Transport code

The advanced TCP stacks for Linux that we chose to evaluate included: standard Linux New-Reno (Reno), HSTCP [4], HTCP [5], Scalable [6], Fast-TCP [7], LTCP [8], HSTCP-LP [9], and BICTCP [10]. Descriptions of the algorithms employed by these TCP stacks can be found in the original papers and in most cases in [11]. We downloaded the latest available versions of the stacks as of April 2004 and installed them on a host at SLAC. In some cases this required compiling from source, in others we simply obtained the binaries.

We downloaded the UDTv2 sources from SourceForge. UDT is a UDP based transport protocol, developed to achieve the single-stream throughput, efficiency and fairness of the existing TCP stacks while being implemented in user space so it requires no kernel modifications.

We obtained a pre-release copy of Solaris 10 from Sun's Solaris Development Engineers at Build Level 69. This was installed on both a Sun Fire V40z and a Sun Fire V20z. The Sun Fire V40z was a quad 2.4 GHz cpu AMD Opteron system. The Sun Fire V20z was a dual 2.4 GHz cpu AMD Opteron system.

B. A&R Network Measurements

The experimental setup on the sender side for the production A&R networks used two hosts (Intel Xeon 3.06GHz) each with a 1GE NIC. The hosts ran Linux 2.4.19 through Linux 2.4.25 kernels, patched with the advanced TCP stacks. On the receiver's side various Intel x86 hosts with > 1.4GHz cpus and 1GE NICs were used with a standard Linux kernel without any patch. On the sender side one host runs **ping** and the other runs **iperf**[2] with the advanced TCP stack. We run **iperf** with a report interval of 1 second. With **iperf** we specify the maximum window size the congestion window can reach as 16384KBytes. Moreover, the size of the NIC's transmit queue length was fixed at 1000 except for FAST tcp for which it was kept at 100. For the receiver side we have chosen hosts at three sites depending on the Round Trip Time (RTT) seen from SLAC, small (Caltech – RTT 10ms), medium (University of Florida (UFL) – RTT 80ms) and large (CERN – RTT 180 ms). The Caltech route was 9 hops via Stanford, CENIC (Stanford, Sunnyvale, LA), and LosNettos. The UFL route was 13 hops via Stanford, CENIC (Stanford, Sunnyvale, LA) and Abilene (LA, Houston, Atlanta). The CERN route was 10 hops via ESnet (Sunnyvale, Chicago) and CERN.

All hosts except that at UFL were set to have maximum send and receive TCP buffer/window sizes of 33.5 Mbytes. UFL was set to 8.4Mbytes.

C. SC2004

For SC2004, we had dedicated access to two dedicated 10Gbits/s circuits from the SLAC/FNAL booth at SC2004 in Pittsburgh to the Level(3) and QWest PoPs at Sunnyvale (in the San Francisco Bay Area, California). In addition we had 10 Sun Fire V20z's 2.4 GHz dual cpu AMD Opteron based systems and one Sun Fire V40z quad 2.4 GHz AMD Opteron 850 based systems. All these Sun Fire systems ran Solaris 10 or Red Hat Enterprise Level 3 (RHEL3) based Linux 2.6. Six of these hosts were at Pittsburgh and five at Sunnyvale. The above hosts had a mix of 10Gbits/s T110 NICs from Chelsio[3] (with TOE), and S2IO[4] (Xframe including TCP Checksum Offload and TCP Large Send Offload (LSO)). The NICs were installed in the 64 bit 133MHz PCI-X bus slots. The Chelsio NIC provided access to its configuration parameters via SNMP.

Most of the hosts were connected to one of two (one at Pittsburgh, the other at the Level(3) PoP in Sunnyvale) Cisco 650x router/switches. Two hosts were connected to a Juniper T320 router at the QWest PoP in Sunnyvale.

III. Methodology

A. A&R Network Methodology

We run four TCP flows using **iperf**, one after the other, each separated by an interval of 2 minutes, and the complete test ran for approximately 16 minutes. Simultaneously we ran pings from the second host at SLAC to the remote host at one second intervals. The incremental throughputs were recorded each second. The flows leave in a LIFO (Last In First Out order). As shown below in Figure 1, we divide the experiment into seven regions (regions 1, 2, 3, 5, 6 & 7 for 2 minutes and region 4 for 4 minutes) and statistics are collected for each of the seven regions as well as per individual flows. Aggregate throughput values are also collected for each of the regions as well as for the overall test.

Deleted: Simultaneoulsy

The 2 minute interval is chosen so that the regions are long enough that usually over 95% of the measurement is made after a flow has completed its initial slow start [12] and is in the more stable AIMD state. The intent is to observe whether the competing flows equally share the bandwidth (fairness) as flows are added/subtracted, and how quickly (if at all) they get to a stable state after a new flow is added/subtracted (stability).

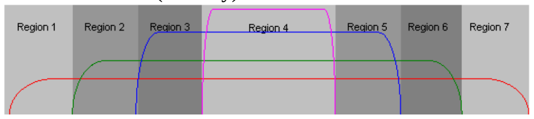

Figure 1: Seven flow regions

[1] http://www.sc-conference.org/sc2004/
[2] http://dast.nlanr.net/Projects/Iperf/
[3] http://www.chelsio.com/
[4] http://www.s2io.com/

For each remote host (Caltech, UFL, CERN) and for each protocol, we typically made three to five 16 minute measurements at different times to reduce the impact of anomalous measurements. Most of the measurements were made at off-peak hours in order to minimize our impact on other network users. The host configurations, measurements and the cpu utilization were recorded (using the Unix `time` command). The data was analyzed to extract the throughputs, stability and fairness. Time-series of the data were plotted and made available together with the data via a web site[5].

B. *SC2004 Methodology*

We made a master system disk starting from RHEL3 patched to Linux 2.6.6, including the various TCP stacks, support for the Chelsio and S2io 10GE NICs, and a common set of testing utilities (e.g. **iperf**, **udpmon**) and support scripts. The system disk was replicated to the system disks for all hosts. To simplify matters we did not have a network file system (such as NFS) but relied on manually keeping the host configurations adequately in step. For security we used the Linux `iptables` facility and due to lack of time we used `/etc/hosts` instead of domain name services.

IV. Results

A. *A&R Results*

To assist in characterizing the stability and fairness quantitatively we use the definitions given in [11] and [14]. That is if we define the average throughput as μ, its standard deviation as s, and then the stability $S = s/\mu$, and the intra-protocol fairness index F is:

$$\left(\sum_{i=1}^{n}\mu_i\right)^2 \Big/ n\sum_{i=1}^{n}\mu_i{}^2$$

In general, all the protocols work well in terms of stability and fairness for the shortest RTT (13.6 ms for Caltech). As the RTT extends to 80 ms (UFL) and 164 ms (CERN), the differences in the performance of the protocols increasingly manifest themselves, e.g. if we take the average S (smaller values of S are better) for all tests, then for Caltech: S=0.21; for UFL: S=0.29; and for CERN: S=0.42; similarly for F (larger values indicate increased fairness), F= 0.9 (Caltech), 0.83 (UFL) and 0.77 (CERN). We will thus focus most of our discussion on examples from the longest RTT.

In Figure 2, a stacked graph of **iperf** achievable throughput per flow for Reno TCP is shown for SLAC to CERN. The measured throughputs are smoothed over 5 second intervals to remove large fluctuations seen in the one second data reports. We observe that the aggregate throughput is not able to recover back to near its initial value even after the flows 2, 3, and 4 have departed the network (due to the slow recovery behavior of AIMD), i.e. Reno is not very stable. Further it can be seen that the throughputs are often not shared equally by different flows (unfair). When a new flow joins congestion may occur (e.g. during the impact of the new flow's initial slow start) and the existing flow may be throttled and take a long time to recover. For a stable and fair protocol we would want that whenever a flow joins or leaves the network, the aggregate remains stable utilizing all the available bandwidth, and the throughputs should be fairly distributed among all the flows.

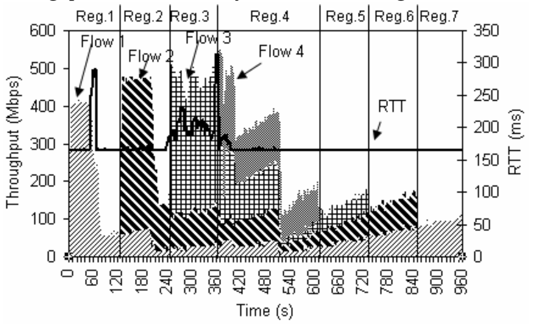

Figure 2: **Iperf** achievable TCP throughputs and RTT for Reno TCP flows joining and leaving the network between SLAC and CERN.

These results appear to confirm the theoretical and simulation results seen by the Hamilton Institute team [13] where packets being sent in bursts lead to lockout, gross unfairness, relatively long convergence times following the bursts, and the new flow often grabbing more than its fair share.

It is also seen that when the aggregate throughput is close to the maximum, the RTT is also extended (in this case by up to 25%). The increases in RTT around the 60 seconds mark are seen to correlate with throttling back the throughput as the protocol detects the congestion.

A second example is seen in Figure 3 for HTCP flows from SLAC to CERN. It is seen that the aggregate bandwidth is more stable, with the exception of when the next to last flow leaves the network at around 840 seconds. It is also observed that the individual flows do a better job of fairly sharing the available bandwidth as new flows are added. Also > 2 flows appears to achieve more throughput and two flows appear to be more stable than > 2 flows. On the other hand the RTT (marked as plus (+) signs) increases when there are multiple flows and is much more variable for the case of more than two flows (varies from 160 to 350msec).

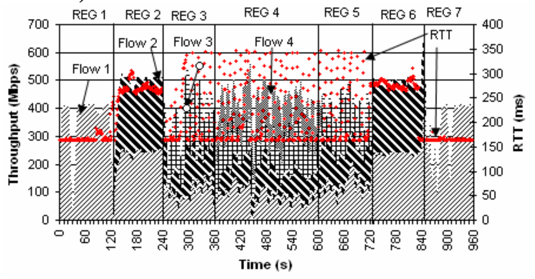

Figure 3: **Iperf** achievable TCP throughputs and RTT for HTCP flows joining and leaving the network between SLAC and CERN.

Fig.4 shows an example of Fast-TCP flows from SLAC to CERN, The aggregate throughput is around 400 Mbits/s with occasional large drops, and the RTTs are much more consistent (standard deviation(RTT) ~ 9ms compared to HTCP's 57 ms and Reno's 22ms). However, the second flow

[5] http://www.slac.stanford.edu/grp/scs/net/papers/pfld05/ruchig/Fairness/

never appears to achieve close to the throughputs of the other flows so the fairness is poor.

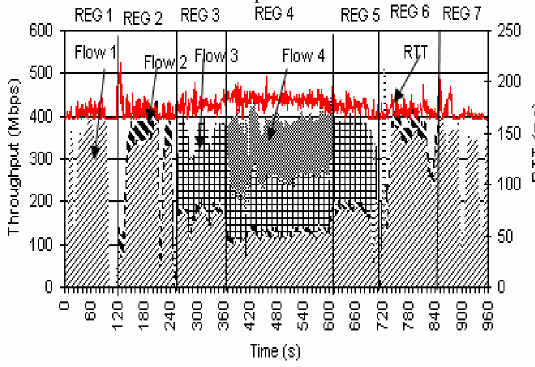

Figure 4: **Iperf** achievable TCP throughputs and RTT for Fast-TCP flows joining and leaving the network between SLAC and CERN.

Fig. 5 shows an example of UDTv2 flows from SLAC to CERN. The aggregate throughputs fluctuate around 390±136Mbits/s. The stability and intra-protocol fairness is comparable to the better TCP implementations. The RTTs (marked as crosses) fluctuate similarly to those seen in Fig. 3 for HTCP.

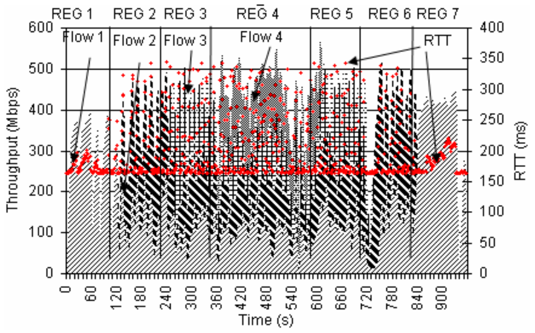

Figure 5: **Iperf** achievable throughput and RTT for UDTv2 flows joining and leaving the network between SLAC and CERN.

To summarize all the protocols for the SLAC to CERN flows, Table 1 shows aggregate (for all seven regions) values for average throughput (μ) in Mbits/s, standard deviation (s), stability (S) = s/μ, minimum and maximum (excluding regions 1 and 7) fairness indices (F), the sender percentage cpu utilization (average over the flows), MHz/Mbps and the standard deviation of the RTTs..

On the CERN link, the best performers in terms of throughput are Scalable, BICTCP and HTCP; the poorest are Reno, HSTCP-LP (as expected since it deliberately backs off in the face of other traffic) and HSTCP. Reno, HSTCP and HSTCP-LP (since it is based on HSTCP this is not surprising) appear to have difficulties recovering aggregate throughput as flows are removed. The most stable protocols appear to be HTCP and BICTCP, the least stable are Reno and HSTCP. HTCP and BICTCP are also the fairest protocols. Reno, Fast-TCP, HSTCP and HSTCP-LP are the least fair with this definition of fair.

As might be expected, Fast-TCP, which uses RTT of the TCP acknowledgement packets for its congestion control, is seen to be the best performer in terms of minimal impact to the ping RTT and presumably the queue congestion.

Table 1: Aggregate statistics for all seven flow regions for SLAC to CERN.

TCP Stack	Avg (μ) Mbps	Std dev (s)	Stab -ility (S= s/μ)	Fairn-ess Min-Max	cpu % util	MHz / Mbps	Std dev (RTT) ms.
Reno	248	163	0.66	0.60-0.99	0.02	0.63	22
HSTCP	255	187	0.73	0.79-0.99	0.028	0.90	25
HTCP	402	113	0.28	0.99-1.0	0.03	0.65	57
Scalable	423	115	0.27	0.82-0.99	0.033	0.64	22
Fast-TCP	335	110	0.33	0.58-0.8	0.028	0.66	9
LTCP	376	137	0.36	0.56-1.0	0.035	0.67	41
HSTCP-LP	228	114	0.50	0.64-0.99	0.01	0.65	33
BICTCP	412	117	0.28	0.98-99	0.033	0.71	55
UDTv2	390	136	0.35	0.95-1.0	0.075	1.2	49

UDTv2 is seen to perform similarly to the TCP implementations. The current version of UDT uses mixed window and rate control and is seen to be about twice as cpu intensive/throughput as the TCP protocols. This is an area the UDT authors are working on, and may be expected to improve. Earlier UDT versions that used a cpu spin loop to rate limit the emitting of packets were more cpu intensive by greater than an order of magnitude.

The TCP protocols' cpu utilizations do not differ significantly between protocols. The standard deviations for the MHz/Mbits/s taken across all the flows for a given protocol and site is in the range 0.006-0.02. The higher value of the cpu utilization/throughput for HSTCP (0.9 compared to 0.69 ± 0.08 MHz/Mbits/s) is caused by its poor throughput performance. The current cpu utilization/throughput values are at the low end of those seen in [15]. This is at least partially due to the current A&R measurements being made with a single parallel stream while those in [15] were made with multiple streams which (see later in the current paper) tend to increase the cpu/throughput ratio.

Detailed stability and fairness indices for each of the 7 regions for SLAC to CERN are shown in Table 2 In regions 1 and 7 since there is only one flow, the fairness is naturally going to be 1 as the single flow gets all the available bandwidth.

Table 2: Stability / Fairness for each of the seven flow regions for SLAC to CERN.

TCP stack	Reg. 1	Reg. 2	Reg. 3	Reg. 4	Reg. 5	Reg. 6	Reg. 7
Reno	0.70 / 1.0	0.46 / 0.7	0.24 / 0.6	0.49 / 0.85	0.46 / 0.98	0.23 / 0.99	0.40 / 1.0
HSTCP	0.13 /1.0	0.30 / 0.95	0.24 / 0.97	0.55 / 0.79	1.02 / 0.99	0.63 / 1.0	0.40 / 1.0
HTCP	0.33 / 1.0	0.26/ 1.0	0.30/ 0.99	0.24/ 0.99	0.20 / 1.0	0.12/ 1.0	0.40/ 1.0
Scalable	0.13 / 1.0	0.36/ 0.99	0.35/ 0.82	0.17/ 0.99	0.21 / 1.0	0.36/ 0.99	0.18 / 1.0
Fast-TCP	0.61 / 1.0	0.36/ 0.62	0.28/ 0.74	0.16/ 0.8	0.30 /0.73	0.23/ 0.58	0.38/ 1.0
LTCP	0.67 / 1.0	0.18/ 0.99	0.33/ 0.82	0.31/ 0.76	0.25 / 0.56	0.19/ 0.97	0.15/ 1.0
HSTCP-LP	0.12 / 1.0	0.28 / 0.64	0.23/ 0.70	0.26/ 0.75	0.21 / 0.71	0.69/ 0.99	0.35/ 1.0
BICTCP	0.35 / 1.0	0.30/ 0.98	0.14/ 0.98	0.17/ 0.98	0.12 / 0.99	0.19/ 0.98	0.34 / 1.0
UDTv2	0.39 / 1.0	0.25/ 1.0	0.50/ 0.95	0.25/ 1.0	0.24/ 1.0	0.26/ 0.99	0.43/ 1.0

B. SC2004 Results

Since the main goal of the SC2004 activities was to demonstrate high bandwidth utilization for the SC2004 Bandwidth Challenge [16], and since we had only access to the two 10Gbits/s paths for three days, we were unable to make completely exhaustive tests. However, we feel there is useful information to impart.

1) S2io NIC with no TOE

We set the maximum TCP window sizes to 20Mbytes, and the NIC-kernel buffer size (`txqueuelen`) to 1000. To validate the performance of the hosts and NICs we first directly connected two hosts back to back via a multimode fiber pair. Then we connected through a Cisco 6509 switch. With one 2.4GHz V20z running Solaris 10 using one S2io 10 GE NIC installed in the host we were able to sustain sending 7.46 ± 0.07 Gbits/s to a 2.2GHz Linux 2.6.5 host with an S2io 10GE NIC. This was using 40 – 50 parallel streams and the default Solaris maximum TCP buffer/window size of 350Kbytes and Linux TCP buffer/window size of 104KBytes. Besides recording the aggregate and incremental throughputs at 5 second intervals, we also recorded the cpu utilization for each flow.

Using two S2io 10GE NICs in the V40z to send through a Cisco 650x switch to two 2.2GHz V20zs each with a 10GE S2io NIC, we were able to achieve 11.5 ± 0.2Gbits/s from the single host.

2) Chelsio TOE NIC

Similar LAN tests were made with the Chelsio TOE NICs with similar aggregate performances. For the WAN tests, the host at Pittsburgh was a V20z with dual 2.4GHz AMD Opteron 64 bit cpus, and the host at Sunnyvale was a similar V20z but with dual 1.6 GHz cpus. Both hosts were running Linux 2.6.6. Since the TOE NIC only implemented the New-Reno TCP stack we used multiple parallel streams on the WAN links between Pittsburgh and Sunnyvale. We set the MTU to the standard 1500Bytes, the `txqueuelen` to 1000, and tuned the TCP send window size and numbers of streams to optimize the **iperf** throughputs. The eventual settings were a 2 MByte window and 16 streams. We also set the **iperf** read/write buffers (-l option) to 128 KBytes. We recorded throughputs, sender side cpu utilization, NIC and kernel interface configurations (`ifconfig`) before and after each run.

The Chelsio NICs performed very predictably over a period of three weeks. We were able to achieve 7.42 ± 0.009 Gbits/s for two hours. The stream average was 463.3 ± 0.8 Mbits/s. The cpu utilization was 148 ± 20%. The critical limitation was the 64 bit 133 MHz PCI-X bus. The raw bandwidth for a 133MHz PCI-X bus is 8.53 Gbit/s, but the PCI-X specification [17] states that data transfers are broken into PCI-X Segments with a length determined by the maximum memory read byte count (mmrbc). This allows interleaving of concurrent data transfers by allowing re-arbitration for the PCI-X bus. The arbitration and DMA resumption suspend data transfer for a time, that is dependent on the chipset and interface involved, leading to a reduction in the effective throughput of the bus. Previous experience [11] and measurements with the Intel Pro/10GbE LR NIC [18] indicated that the effective bus throughput was ~ 6.83Gbit/s with the mmrbc of 4096 bytes..

With two pairs of V20z hosts on each end of the path from Pittsburgh to the CENIC PoP in Sunnyvale we achieved sending 9.07Gbits/s recorded by **iperf** (equivalent to 9.43 Gbits/s with headers etc.) in the forward direction and simultaneously sending 5.44 Gbits/s reported by **iperf** (equivalent to 5.6 Gbits/s) in the reverse direction for a total of ~ 15 Gbits/s.

We measured the cpu utilization as a function of throughput and number of parallel streams as seen in Fig. 6. The curves are to guide the eye and have no deep significance. It is seen that cpu utilization is not very linear in achievable throughput.

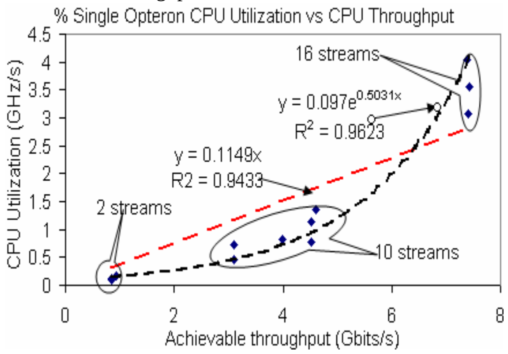

Figure 6: CPU utilization of a single Opteron 2.4 GHz cpu versus the achievable **iperf** TCP throughput.

Plotting the sender side GHz utilized / Gbits/s versus the number of streams results in Fig. 7 where we have also added in the S2io NIC results. It is apparent that using the Chelsio TOE NIC with Linux 2.6.6 sender, one utilizes roughly three times less cpu cycles compared to an S2io (non-TOE) NIC on Solaris 10. Some of this could be attributed to the lack of LSO support in the version of Solaris 10 that we used. Unfortunately we had insufficient time to repeat these tests with identical operating systems. From studies elsewhere [15] it does not appear there are significant differences between earlier versions of Solaris and Linux in the cpu utilization/achieved throughput. Further industry claims[6] of large reductions in cpu utilization would seem to indicate the main effect was from the TOE rather than the operating system difference. There are also competing industry claims and third-party tests indicating that a NIC with stateless offloads - like LSO (a default feature on many Operating Systems) and Large Receive Offload - achieve reductions in cpu utilization that is comparable to a TOE.

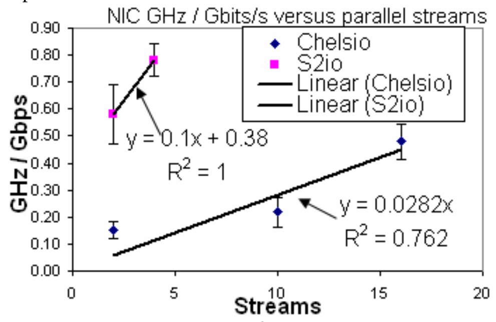

Figure 7: CPU utilization as GHz used for a given TCP achievable throughput versus the number of parallel TCP streams.

Finally we made measurements with UDT, however, due to lack of time these were only local (i.e. within the booth at SC2004). We were able to achieve 4.45 Gbits/s between two V20Zs with 2.4GHz cpus and Chelsio NICs. From discussions with the UDT authors[7], we believe this is partly due to implementation efficiency as it runs out of cpu performance.

V. CONCLUSION

A. A&R Network Conclusions

From our experiments on the A&R Network testbed, we observe that from the point of view of throughput, Scalable, BICTCP and HTCP are the best performers while Reno, HSTCP-LP and HSTCP are the poorest ones. As far as the stability goes, HTCP and BICTCP are the most stable TCP protocol implementations while Reno and HSTCP are the least stable. Moreover, HTCP and BICTCP also have the best intra-protocol fairness while Reno, Fast-TCP, HSTCP and HSTCP-LP are less fair.

In terms of minimizing the impact on the RTT Fast-TCP has the best performance among the various TCP stacks that we tested.

Our evaluation of UDT on 1Gbits/s paths reveals that UDT is closing the performance gap between itself and the advanced TCP stacks implementations. However, UDT is much more cpu intensive than the advanced TCP implementations for the same throughput.

[6] See for example: http://www.broadcom.com/products/product.php?product_id=BCM5706 and http://www.veritest.com/clients/reports/chelsio/default.asp

[7] *Currently on Opterons we can only reach about 5.2Gb/s at most. This is partly due to the implementation efficiency as the CPU will be used up. We found that even with iperf/UDP the performance is still less than iperf/TCP (6Gb/s vs. 7.xGb/s). We are still investigating this problem, but I suspect it is because that many of TCP functionalities have been off loaded to hardware (e.g., NIC) and since TCP is much more often used than UDP, its implementation is highly optimized (while UDP is not).* Yunhong Gu, UIC UDT developer, private communication

B. SC2004 Conclusions

In our experiments on the SC2004 testbed, we achieved almost identical performance on the WAN as the LAN with the use of Chelsio NICs. This was evident as we were able to saturate over 99% of a 10Gbits/s cross-country link with two pairs of hosts each with a 10GE Chelsio NIC. Moreover, we observed that the Chelsio TOE NIC's performance was very stable on uncongested paths.

Measurements of cpu utilization helped us gain insight into the performance speed-up resulting due to the use of TOE. The sender cpu utilization for a V20z running Linux 2.6.6 with a Chelsio TOE NIC was roughly a factor 3 less than that for a V20z running Solaris 10 with a S2io 10GE NIC. This is believed to be mainly due to the effects of the TOE. Also, some of this could be possibly attributed to the lack of LSO support in the version of Solaris 10 that we used.

We also studied the performance of S2io NICs and were able to send ~ 11.5 Gbits/s from a single V40z Solaris 10 host (with two 10GE NICs) to two V20z's connected via a single switch.

Among other things, we were able to demonstrate the smooth inter-working of Chelsio and S2io 10GE NICS as well as Cisco 650x switch/routers and a Juniper T320. We also observed that on a 2.4GHz Opteron throughput is limited by the PCI-X 64 bit 133MHz bus. Finally, UDTv2 was unable to achieve as much throughput as multi-stream Reno on 10Gbits/s paths.

C. Future Work

A criticism of the current work is the lack of inter-protocol comparisons such as made in [11]. In particular comparisons against the most prevalent transport protocol (Reno with a single and multiple streams) are needed. This needs to be addressed for both the production A&R networks and 10Gbits/s testbeds.

The TOE vs. non-TOE comparisons are very important especially for multi-Gbits/s achievable throughputs where cpu utilization may be a limiting factor as bus speeds increase. We believe further work is needed to understand the relations between operating systems, buses, and levels of off-loading. To this end we would also like to make new measurements with Solaris 10 when they have support for LSO support and also with newer advanced NICs.

In terms of achievable throughput, newer buses such as PCI-X 2.0 and PCI-Express promise to remove one of the bottlenecks and so will need evaluating.

Both the TCP stacks and UDT are evolving and so newer versions need to be evaluated as well as other TCP stacks such as TCP-Africa and TCP-Westwood.

ACKNOWLEDGMENT

We gratefully acknowledge Sun Microsystems for the loan of the V20zs and V40z, Chelsio for the loan of TOE NICs, Cisco for 10Gbits/s switch/router equipment, NLR for loan of 10Gbits/s circuit from Sunnyvale to Pittsburgh, ESnet for dedicated access to 10Gbits/s circuit from Sunnyvale to Pittsburgh, ESnet and CENIC for collocation space in their Sunnyvale PoPs, John Goebel for help with Linux 2.6. building, installation, patching etc., S2io for help with using their NICs. Yunhong Gu and Robert Grossman for help with UDT. We acknowledge useful discussions with

Douglas Leith of the Hamilton Institute. Also we must not forget Olivier Martin, Sylvan Ravot for help with the CERN setup, Harvey Newman and Suresh Singh for help with the Caltech setup, and Chris Griffin for help with the UFL setup, and Gary Buhrmaster and Phil DeMar for assistance in setting up in the SC2004 booth and copious advice.

References

[1] S. Floyd, T. Henderson and T. Gurtov, "RFC 3782 - The NewReno Modification to TCP's Fast Recovery Algorithm", available at http://www.faqs.org/rfcs/rfc3782.html

[2] W. Feng, M. Fisk, M. Gardner and E. Weigle, "Dynamic Right Sizing: An automated lightweight, and scalable technique for enhancing grid performance", in *7th IFIP/IEEE International Workshop, PfHSN 2002*, Berlin, April 2002.

[3] Y. Gu and R. L. Grossman, "UDT: An Application Level Transport Protocol for Grid Computing", *Second International Workshop on Protocols for Long Distance Protocols*, 2005.

[4] S. Floyd. "RFC 3649: HighSpeed TCP for large congestion windows", available at http://www.faqs.org/rfcs/rfc3782.html, Dec 2003.

[5] R. Shorten, D. Leith, J. Foy, and R. Kildu. "Analysis and Design of Congestion Control in Synchronized Communication Networks", 2003.

[6] T. Kelly. "Scalable TCP. Improving Performance in Highspeed Wide Area Networks", 2002.

[7] C. Jin, D. Wei, S.H. Low, G. Bushmaster, J. Bunn, D.H. Choe, R. L. Cottrell, J.C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. "Fast TCP – from Theory to Experiments", PFLDnet 2003.

[8] S. Bhandarkar, S. Jain and A.L.N. Reddy. "LTCP: A Layering Technique for Improving the Performance on TCP in Highspeed Networks", available http://ee.tamu.edu/~reddy/papers/jogc2003.pdf

[9] A. Kuzmanovic and E.W. Knightly. TCP-LP: A Distributed Algorithm for Low Priority Data Transfer. In IEEE INFOCOM, San Francisco, April 2003. Also see A. Kuzmanovic, E.W. Knightly and R. L. Cottrell, "HSTCP-LP: A Protocol for Low-Priority Bulk Data Transfer in High-Speed High-RTT Networks", PFLDnet 2004.

[10] L. Xu, K. Harfoush, and I. Rhee. "Binary Increase Congestion Control (BIC) for Fast, Long-Distance Networks", INFOCOM 2004.

[11] H. Bullot, R. L. Cottrell, and R. Hughes-Jones. "Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks", PFLDnet 2004.

[12] A. Tirumala, R. L. Cottrell, T. Dunigan, "Measuring end-to-end bandwidth with Iperf using Web100", SLAC-PUB-9733, published at PAM2003, April 2003.

[13] D. Leith, R. Shorten. "H-TCP Protocol for High-Speed Long Distance Networks", PFLDnet 2004.

[14] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", In Computer Networks and ISDN systems, pages 1-14, June 1989.

[15] R. L. Cottrell, C. Logg, and I-Heng Mei, "Experiences and Results from a New High Performance Network and Application Monitoring Toolkit", PAM 2003, April, also SLAC-PUB-9641.

[16] R. L. Cottrell, "High Speed Terabyte Data Transfers for Physics - SC2004 Bandwidth Challenge Proposal", available http://www-iepm.slac.stanford.edu/monitoring/bulk/sc2004/hiperf.html

[17] PCI Special Interest Group, "PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0a", July 2000.

[18] R. Hughes-Jones, P. Clarke, S. Dallison, "Performance of 1 and 10 Gigabit Ethernet Cards with Server Quality Motherboards," Future Generation Computer Systems Special issue, 2004