

Benchmarking High Bandwidth-Delay Product Protocols

Richard Huang, Andrew Chien

*Department of Computer Science
University of California, San Diego
ryhuang, achien@cs.ucsd.edu*

Abstract

TCP is the de facto transport protocol of the Internet, but its shortcomings in the large bandwidth and large latency networks is well documented. We create a set of benchmarks for protocols designed for the large bandwidth-delay product networks. By using the benchmarks to evaluate the leading protocols in network environments that vary in their bandwidth and latency, we found that each protocol far outperformed TCP, although each has its limitations. In the largest bandwidth-delay product environment we tested, Pockets achieved a maximum bandwidth of 875Mbps, while RBUDP and UDT were not far behind at 837Mbps and 840Mbps respectively, compared to 4.57Mbps for TCP. In achieving the high bandwidth, Pockets required user input to use 256 parallel sockets while RBUDP required pre-determination of sending rate. UDT requires an orders of magnitude more data and time to achieve maximum bandwidth as compared to Pockets.

1 Introduction

While TCP (Transport Control Protocol) has been the de facto protocol for much of the Internet today, the general consensus among the research community is that standard TCP is not suitable for bulk data transfers in high bandwidth, large round trip time networks because of its slow start and its congestion control mechanism.

Recent advances in network bandwidth coupled with the shortcomings for TCP in the high bandwidth-delay product (BDP) environment have led to

much research in developing new protocols to utilize the high bandwidth and to sustain the high transfer rate. Each new protocol hopes to capitalize on a shortcoming for TCP.

In this paper, we designed a set of benchmarks to help us study protocols intended for the high bandwidth-delay product environment. We studied and compared to TCP these three protocols: Parallel Sockets (PSockets), Reliable Blast UDP (RBUDP), and UDP-based Data Transfer Protocol (UDT).

PSockets improved upon the existing TCP by utilizing multiple TCP sockets, thus artificially expanding the default window size of TCP. The possible downside is that many parallel sockets might be required, usurping system resources and available bandwidth from other protocols.

RBUDP avoids the per packet interaction of TCP. The sending rate is user-inputted, so RBUDP avoids the slow start phase of TCP. The drawback is the user needs prior knowledge of the network capacity for the smallest link between the sender and the receiver.

UDT retains some of the slow start characteristics and congestion control of TCP, but allows the application to set the flow control window size. Because of its similarities to TCP, it suffers from some of the same symptoms such as slow adjustments to network capacity.

We compare these protocols to untuned TCP, as well as measured TCP and UDP bandwidth from Iperf.

We designed a set of benchmarks by first identifying the desirable characteristics of a high BDP protocol. High utilization of network capacity is paramount. Fast adjustment to network capacity is important

for small file transfers. The protocol should be fair to TCP and be stable. We were also interested in the load the protocols places on the sender and the receiver. From these characteristics we created our benchmarks.

Our experiments showed that in the highest BDP environment, Pockets reached 875Mbps, RBUDP reached 837Mbps and UDT reached 840Mbps over a 100GB data transfer, as compared to 4.57Mbps for TCP. To accomplish the high bandwidth, Pockets required an exorbitant (256) number of parallel sockets; RBUDP had some data integrity issues; UDT still possess the slow start characteristics of TCP.

We discuss the protocols in Section 2 and the methodology in Section 3. In Section 4 we present our results. We discuss our observations of each of the protocols in Section 5. We finish by summarizing and providing directions for future work in Section 6.

2 Protocols

Two opposing camps exist for improvements to the standard TCP; one camp modifies UDP to transfer data reliably while the other camp improves upon the standard TCP implementation. The new protocols should perform at least as well as standard TCP in the low BDP networks and significantly outperform standard TCP in the high BDP networks.

While gathering data for this paper, we were limited to testing protocols that did not require root access. That ruled out the TCP variants we studied. High Speed TCP [1] improves the loss recovery time by changing the AIMD algorithm when the congestion window is higher. Scalable TCP [2] also modifies the congestion avoidance phase. FAST TCP [3] discards slow start, AIMD and congestion avoidance and implements two control mechanisms to share link resources fairly and achieve maximum link utilization.

As we did not have access to the routers, we also could not test XCP [4] and PTP [5]. Both XCP and PTP require congestion signaling from routers and do not

employ TCP's slow start or congestion avoidance mechanisms.

We were able to test Tsunami [6], an UDP-based file transfer protocol. However, as we were interested in a comparison among general transport protocols, the results were not included in this paper.

2.1 Parallel Sockets

Parallel Sockets (Pockets) [7] stripe data buffer over multiple sockets in equal parts. It overcomes the window size limit of TCP by aggregating the window sizes of each of its parallel sockets. Because the default TCP window size is 64K, the standard TCP socket will not fill up a big pipe. With a round trip time of 64ms and the capacity of 1 Gbps, standard TCP can ideally achieve 8 Mbps, far less than the capacity 1 Gbps. In practice, we observed approximately 4.5 Mbps for TCP. By using multiple sockets, Pockets is able to achieve the aggregate bandwidth of each of the sockets. We used Pockets code from DataSpace 2.2, released October 31, 2002.

2.2 Reliable Blast UDP

Reliable Blast UDP (RBUDP) [8] addresses problems of bulk data transfer by eliminating the slow start phase of TCP and obviating the need for per packet acknowledgements. It has two goals: to utilize the capacity of the network, and to avoid the per packet interaction of TCP so that acknowledgements are not sent per window of transmitted data, but aggregated and delivered at the end of a transmission phase. Using RBUDP requires a priori knowledge of the bottleneck network bandwidth. The authors suggested using Iperf [9] or netperf [10] to measure such bandwidth. Further, the authors developed app_perf (modified version of iperf) to take into account extra memory copy that most applications must perform.

In the data transmission phase, the sender sends the entire buffer at the user-

specified sending rate. The receiver acknowledges with a bitmap consisting of received packets. The sender responds by retransmitting the missing packets and the process repeats itself until there are no more missing packets.

RBUDP is part of QUANTA (Quality of Service Adaptive Networking Toolkit). We used QUANTA 0.2 version, released April 9, 2003.

2.4 Iperf

Iperf is a tool to measure maximum TCP and UDP bandwidth. Users need to adjust parameters such as TCP window size or UDP bandwidth to achieve the desired maximum TCP or UDP bandwidth. We obtained Iperf numbers for TCP and UDP as controls for our experiments. We used Iperf version 1.7.0, released March 2003. This version did not support 64 bit architecture, so we were not able to get numbers for the TeraGrid machines.

2.3 UDP-based Data Transfer Protocol

SABUL [11] (Simple Available Bandwidth Utilization Library)/UDT (UDP-based Data Transport) addresses problems of high speed reliable communication. It removes TCP window size constriction by using UDP for data transfers. SABUL uses UDP to transfer data and TCP to transfer control information while UDT uses UDP for both data and control information transfer.

The algorithm calculates an inter-packet time, which is updated by rate control. Packets are sent out every inter-packet time, but the number of outstanding packets cannot exceed a threshold, calculated by the flow control. The receiver receives and reorders data packets and sends back NAK packets and uses selective acknowledgement at every constant time. A retransmission packet has higher priority than first time packet.

We tested SABUL 2.3 and UDT 1.0, both released August 1, 2003.

3 Methodology

We discuss our rationale for choosing our metrics and how we performed our experiments here. We also list test driver template and our experimental configurations.

3.1 Metrics

We designed our metrics by identifying desirable characteristics of a high BDP protocol. These include: high network capacity utilization, fast adjustment to network capacity, fairness and stability, and CPU load.

3.1.1 Maximum Bandwidth

First and foremost, we were interested in the utilization of network capacity for a sustained period of time. For this metric, we decided to use 100GB of memory to memory transfer. We also tested 10GB transfers to make verify consistency in transfer rates at a smaller transfer size. Because of the hardware limitations, we chose 400 MB as the buffer size for the sender and receiver. We defined the maximum bandwidth as the bandwidth achieved by the protocols for the 100 GB transfer. Our results show the average of 10 trials for 10 GB transfers and 5 trials for 100GB transfers.

3.1.2 Fast Adjustment to Capacity

To measure how fast a protocol reaches network capacity, we decided to measure data set sizes at 50% and 90% maximum bandwidth. These two values provide insight as to how fast each protocol is in reaching maximum bandwidth. Another logical metric is the time to maximum bandwidth.

For each iteration of sending the buffer, we note the time, the instantaneous and average bandwidth. The time to

maximum bandwidth is the first time when the average bandwidth is equal to or greater than the maximum bandwidth we achieved.

For values smaller than 400MB, we use trial and error to determine 50% and 90% data set sizes. For values greater than 400MB, we employ the same technique as the one for reaching maximum bandwidth.

3.1.3 Fairness, Stability, and CPU Load

Fairness issues leads to a plethora of possibilities in layout and protocol combinations. Multi-senders and multi-receivers are possible. We decided to examine fairness in a future study.

To test for stability, we modify the contents of the memory buffers before sending. At the receiver, we confirm the integrity of the data buffers.

We noted the CPU utilization on both the sender and the receiver machines by running top on each of the machines while we run the protocol.

3.2 Template for Test Drivers

While each protocol may require a unique modification to the template we developed (e.g. number of sockets for Psockets or maximum bandwidth for RBUDP), we used the following starting template for our test drivers:

On the sender side:

```
appserver <port> <buf size> <iterations>
```

```
initialize arguments, buffer
set up port, wait for receiver to connect
for (i = 0; i < iterations; i++) {
    server->send(buffer, size);
    print inst. rate, ave. rate, time
}
```

On the receiver side:

```
appliance <sender ip> <sender port> <buf
size> <iterations>
```

```
initialize arguments, buffer
connect to sender
for (i = 0; i < iterations; i++) {
```

```
    receiver->receive(buffer, size)
    print inst. rate, ave. rate, time
}
```

3.3 Experimental Configurations

We gathered the data in three different environments: cluster, SDSC, and TeraGrid. In the cluster environment, the machines we used were dual Intel Xeon 2.4Ghz Processors with 1 GB DDR RAM running Linux Redhat 7.3, kernel 2.4.18. The bandwidth between two nodes were 1 Gbps with RTT of 0.2ms.

We had access to the San Diego SuperComputer Center (SDSC) OptIPuter machine slic04.sdsc.edu and also the TeraGrid machines tg-login1.sdsc.teragrid.org and tg-login2.sdsc.teragrid.org at SDSC and tg-login1.ncsa.teragrid.org and tg-login2.ncsa.teragrid.org at NCSA. slic04 has four Intel i686 Processors with 1 GB of DDR RAM running Linux Redhat 7.3, kernel 2.4.18. The TeraGrid machines have Madison 1.3GHz CPU's running SuSE SLES8/2.4.19-SMP with 8 GB of RAM. The bandwidth between the SDSC OptIPuter and SDSC TeraGrid machines is 1 Gbps with RTT of 0.38ms. The bandwidth between the SDSC and NCSA TeraGrid machines was also 1 Gbps with RTT of 0.61ms. The bandwidth between SDSC OptIPuter and NCSA TeraGrid machines was determined to be 562Mbps for TCP and 578Mbps for UDP using Iperf. The RTT is again 61ms.

4 Experimental Results

In this section we present the experimental results for each of the protocols and how they compare to each other. We discuss and analyze the results for each protocol in detail in Section 5.

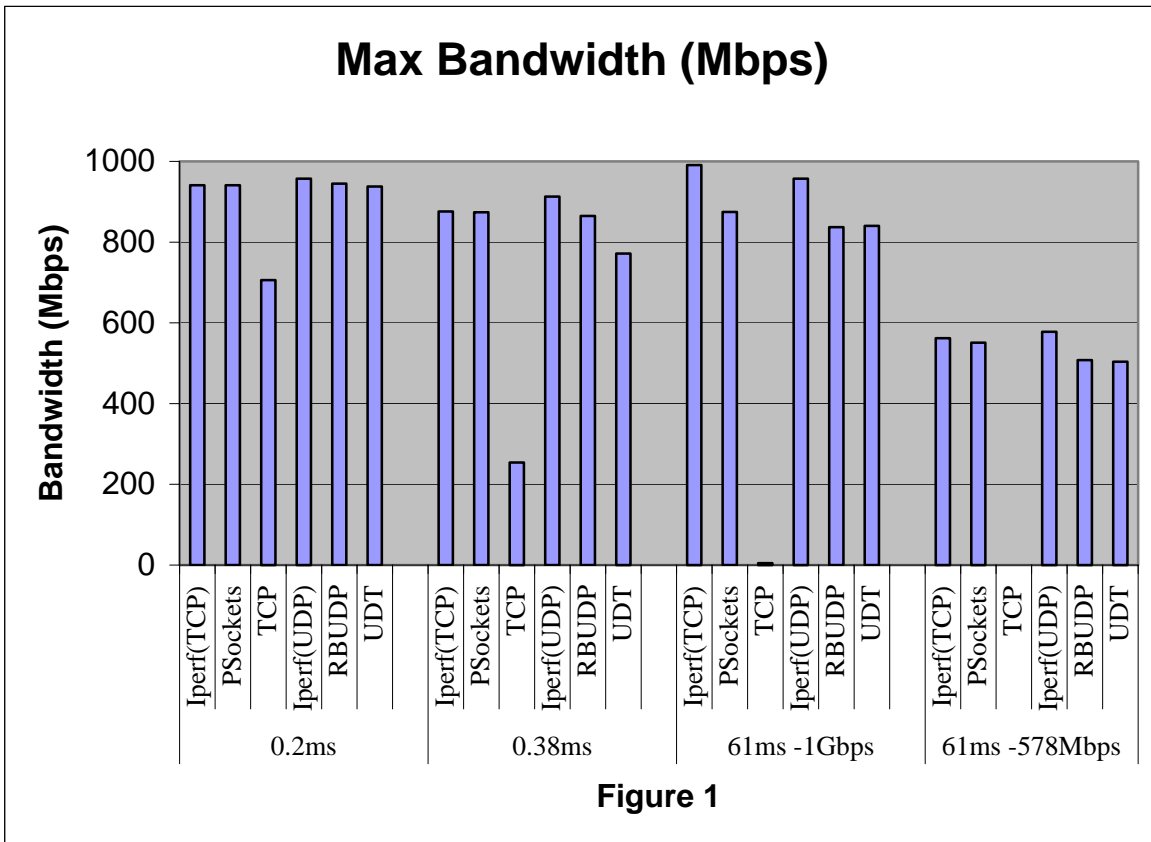
4.1 Maximum Bandwidth

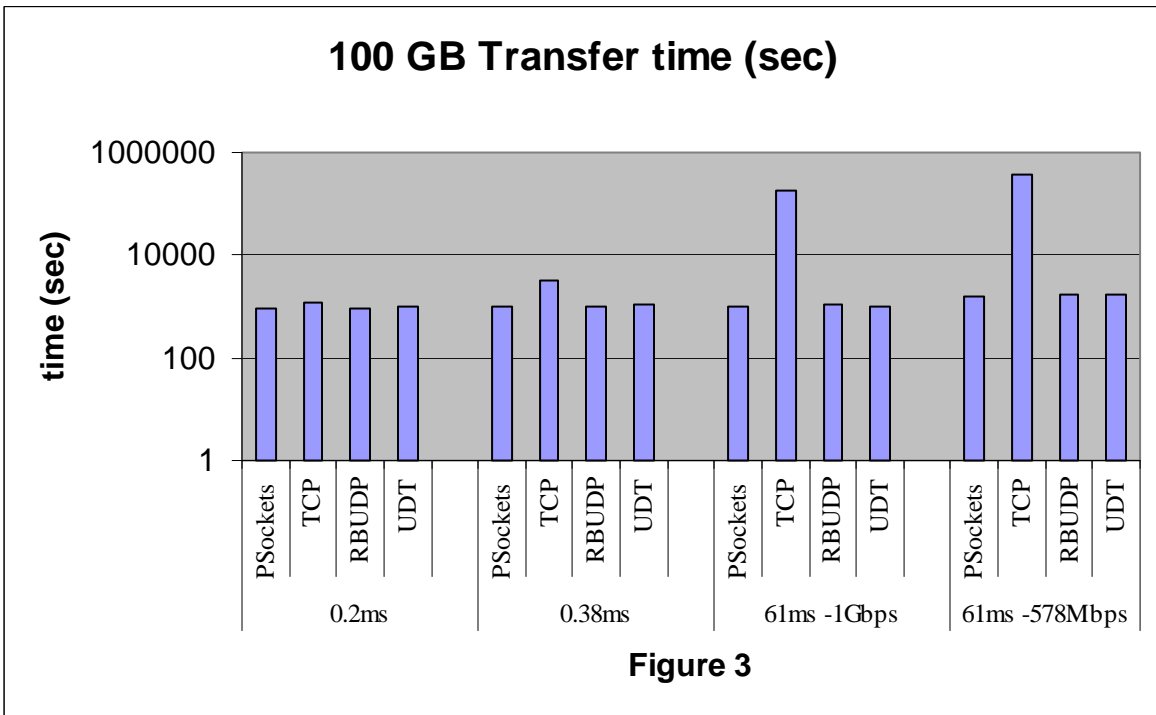
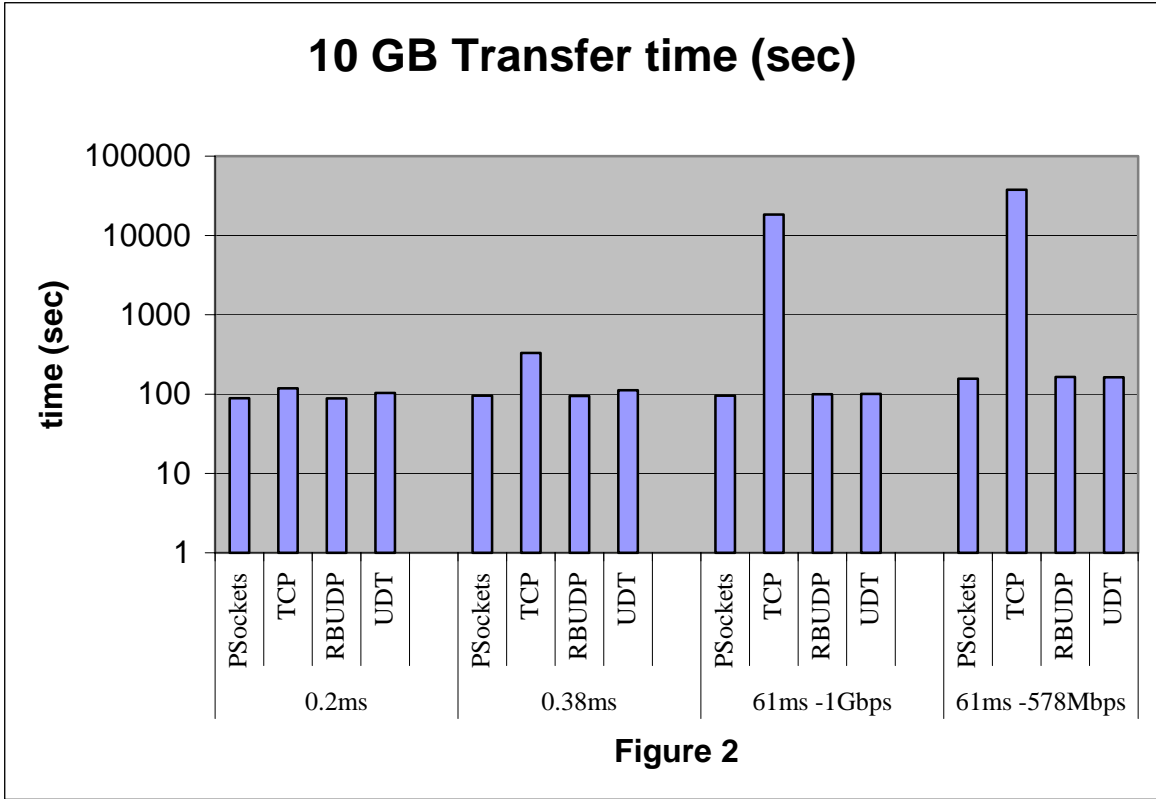
Psockets performed at similar peak bandwidth as the numbers from Iperf in all environments. RBUDP nearly matched

Iperf's measured UDP bandwidth in the cluster environment, but performed worse when the RTT increases. UDT was able to nearly match Iperf's measured UDP bandwidth in the cluster environment, but drops off noticeably in the other environments. The performance for untuned TCP drops as the RTT is increased, down to 4.57 Mbps at 61ms RTT.

One would expect the utilization rate to decrease as the distance or bandwidth

between two points increase, as traffic becomes more unpredictable and packets are more likely to be dropped, especially with UDP packets. This is what we observed with P.Sockets and RBUDP. UDT exhibited the unexpected behavior of achieving slightly higher bandwidth in the 61ms RTT environment than the 0.38ms environment, suggesting more packet drops in the 0.38ms environment.





4.2 Fast Adjustment to Capacity

TCP and PSocketS reached maximum bandwidth fairly quickly without much data transfer. With the exponential growth of window size in slow start phase multiplied by the number of parallel sockets, we expected this from PSocketS. TCP was able to achieve maximum bandwidth quickly because of its lower maximum bandwidth. Both protocols reached 90% of maximum bandwidth about an order of magnitude more data than when it was at 50% of maximum bandwidth.

In comparison, UDT requires almost two orders of magnitude more data to reach 50% and 90% of maximum bandwidth. This implies that when smaller files are being transferred, UDT may not reach its

maximum bandwidth before the termination of the file transfer.

As expected, all protocols require more data and more time to reach maximum bandwidth for a larger RTT to fill up the pipe with packets. We did not show data for RBUDP here because its sending rate is defined by user input.

4.3 CPU Utilization

PSocketS requires more processing at the receiver side while UDT requires more processing at the sender side. RBUDP requires about an equal amount of processing power on both sides. All protocols consume far more CPU cycles than TCP.

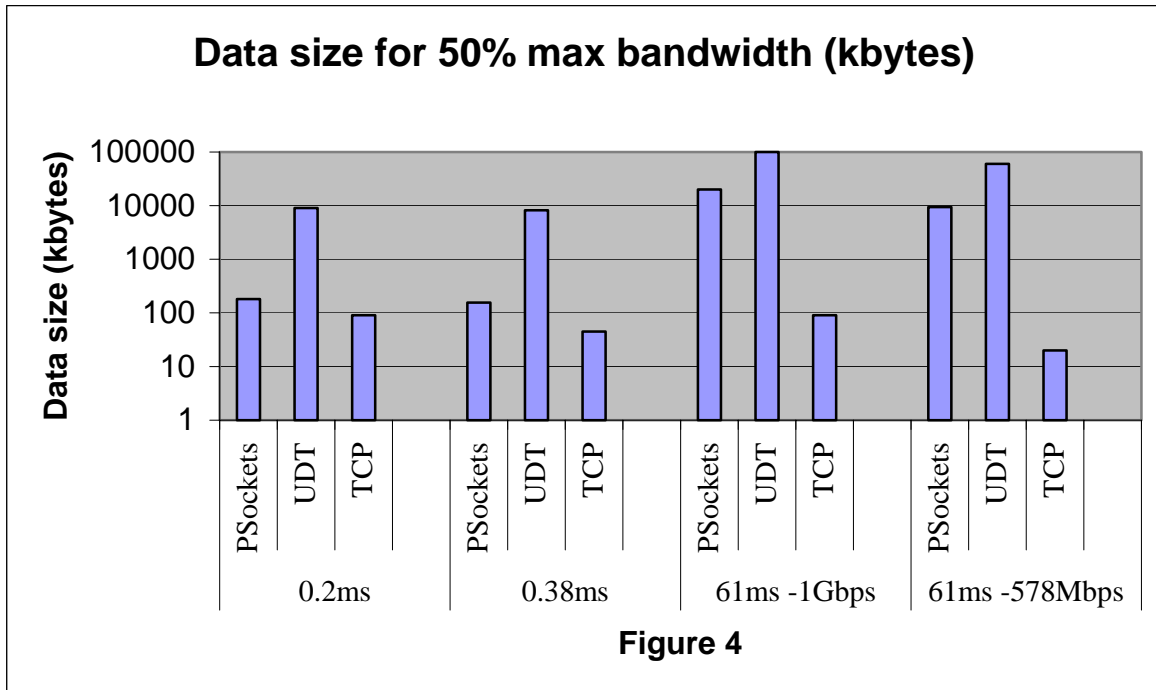
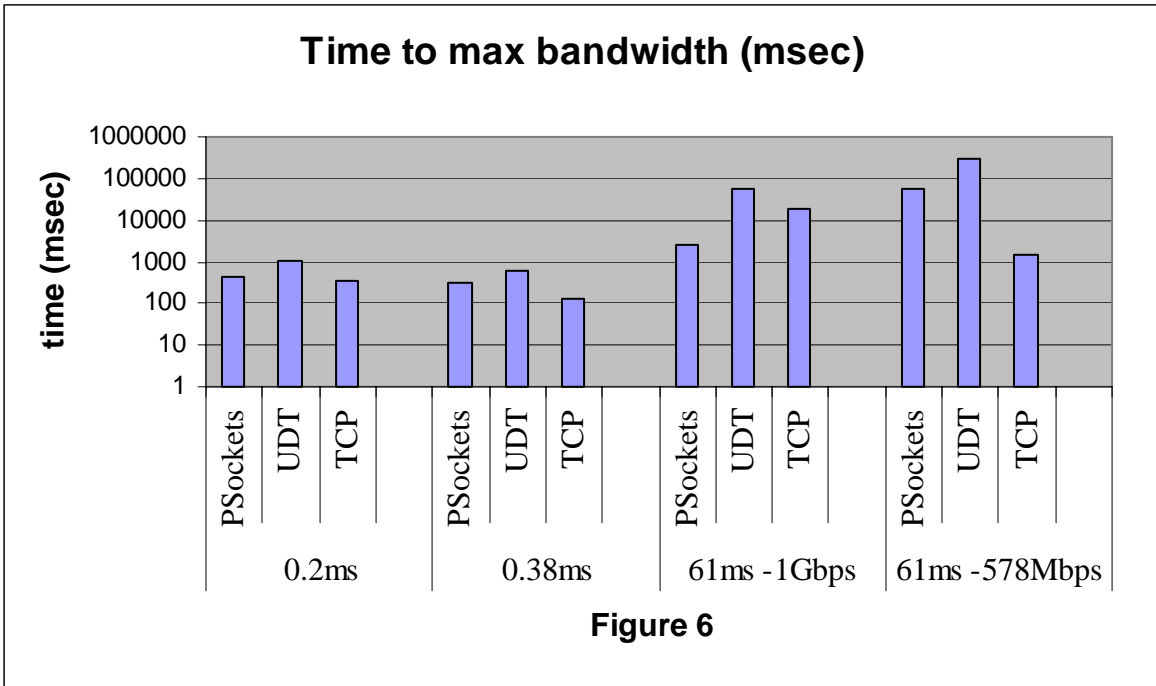
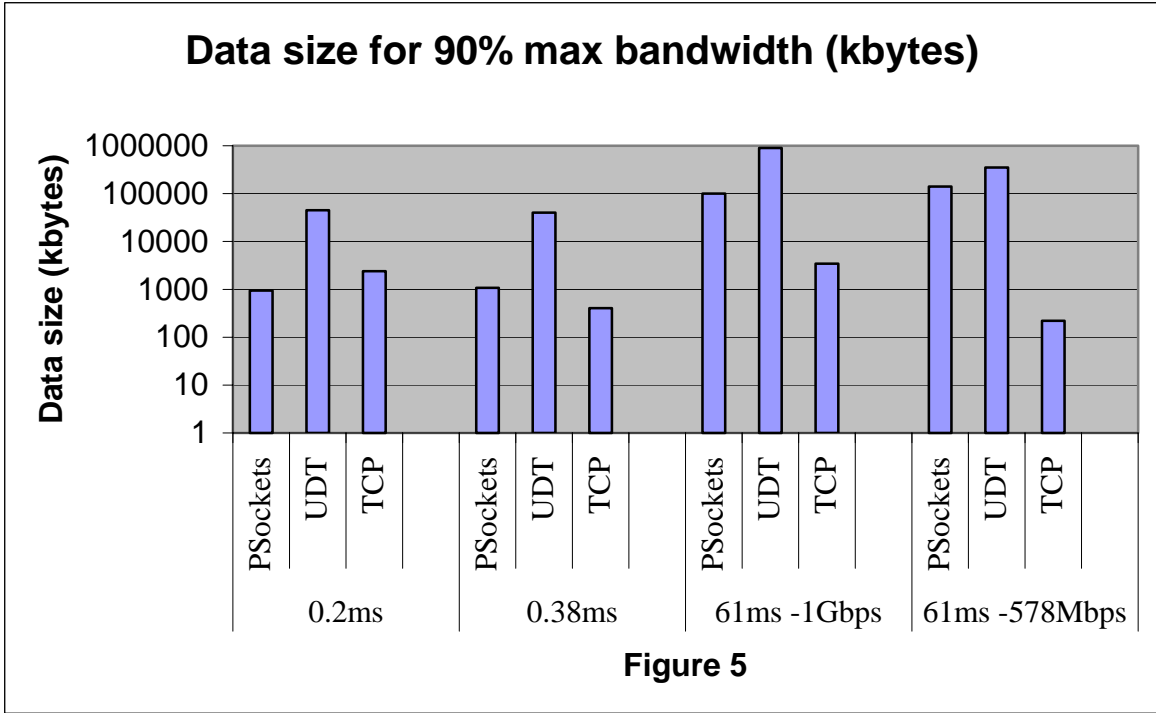


Figure 4



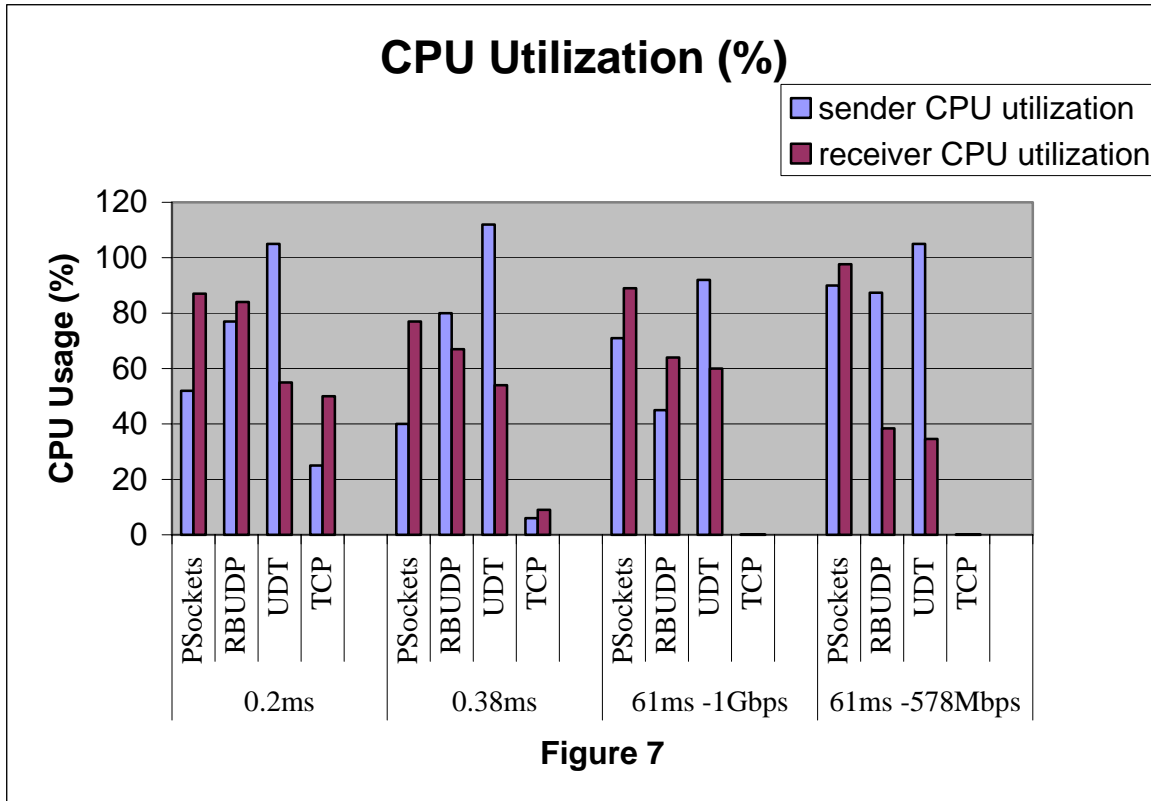


Figure 7

5 Discussion

We discuss the results for each protocol in detail. First we present unusual observations about each protocol, then its advantages followed by its disadvantages.

5.1 PSockets

Through trial and error, we found 8 parallel sockets to be optimal for both the cluster and SDSC machines. We found that using more parallel sockets did not improve performance.

For the TeraGrid machines with RTT of 61ms, we had to use 256 parallel sockets. More parallel sockets started to degrade performance. We suspect the overhead of maintaining so many open sockets nullified any performance gains beyond the 256 sockets. Generally the default number of sockets a system can support is 1024, so that might be an issue

when higher bandwidth is available and more parallel sockets are required.

Another interesting point we observed was that bandwidth does not scale exactly linearly with the number of sockets, even when the pipe is not full. In the TeraGrid environment, with one socket, the maximum bandwidth is 4.57 Mbps, yet 100 parallel sockets produce less than 400 Mbps. Again, we suspect that the overhead of maintaining so many open sockets is depressing the performance somewhat.

PSockets outperformed all other protocols in this study in peak bandwidth. Although we need to determine the optimal number of sockets to use, in practice, this can be done fairly quickly if we're not concerned with the absolute maximum.

To achieve such high bandwidth, PSockets required an inordinate number of parallel sockets. It is doubtful whether adding more sockets is scalable in the near future when higher bandwidth is available or when the distance between two points is

increased. In terms of fairness, it is again doubtful whether this protocol is viable to coexist with other protocols not using so many sockets.

5.2 RBUDP

With the knowledge of the capacity of the links we used, we did not need to use Iperf or Netperf. RBUDP achieved somewhat lower bandwidth than Pockets, but still respectable 837Mbps in the 61ms RTT environment.

The major advantage of RBUDP is the instantaneous adjustment to the network capacity. The protocol can begin to send at peak capacity as soon as it starts. It maintains the transfer rate and makes small adjustments as necessary without needing to saturate the network and scale down the number of packets when congestion is detected (as in TCP).

The major drawback is the requirement for initial sending rate, i.e. the network capacity. It is unclear how this protocol would react in a dynamic environment if the network capacity drastically changes from the initial inputted sending rate.

In checking for the integrity of the data transferred, we occasionally found corrupt data.

5.3 SABUL/UDT

Initially we tested SABUL, but after struggling to make SABUL 64 bit compatible so it would run on the TeraGrid machines, we were not able to get good performance from SABUL, so we felt it was unfair to run benchmarks for SABUL for a platform it did not support. UDT is a successor to SABUL, and fortunately UDT does run on 64 bit architecture, so we were able to proceed with no problems. The interesting observation we made was that UDT gradually reaches a bandwidth higher than the final maximum bandwidth, but slowly decrease in the rate over time.

Unlike Pockets or RBUDP, UDT did not require user input for some dynamic variable. It is a stand-alone protocol that can determine the capacity of the network and adjusts its rates accordingly. UDT is designed to be a reliable protocol; we did not find any integrity problems, even though UDT is using UDP.

The negative on UDT is that its performance is worse than Pockets or RBUDP. UDT also did not improve on the slow start and was slow to ramp up to the maximum bandwidth. The somewhat disturbing trend of decreasing average rates for UDT over time was most likely due to congestion and dropped packets.

6 Summary and Future Work

We created a set of benchmarks to aid our study of high BDP protocols. We designed the metrics by listing the desirable characteristics for a high BDP protocol, then benchmarking those characteristics. Our benchmarks include *Maximum Bandwidth*, *Time to Transfer 10GB and 100GB*, *Data size for 50% and 90% Maximum Bandwidth*, *Time to Maximum Bandwidth*, and *CPU Utilization*.

In the largest BDP environment we tested, our experiments show that Pockets achieved a maximum bandwidth of 875Mbps, RBUDP achieved 837Mbps, and UDT achieved 840Mbps, as compared to 4.57Mbps for TCP.

TCP's shortcomings in the high BDP environment are mostly attributed to its slow start phase, its congestion avoidance mechanism, and the default window size. Pockets overcome the default window size by aggregating the window sizes of multiple TCP sockets. RBUDP avoids the slow start and the window size and uses different congestion control mechanism. UDT improves on the window size to achieve higher bandwidth.

One area we still need to address is the issue of fairness. Specifically we would like to know how fair each protocol is to TCP and to each other. We would also like

to study the effect of packet losses as the packets travel from higher capacity links to lower capacity links.

Another area we were not able to study is the pool of TCP-variant protocols, as we don't have root access to the machines used in this study. When we acquire more machines and set up dummynet [12], we can simulate latency and bandwidth. At that time we will be able to test the TCP-variants and also higher bandwidth-delay products.

References

1. Floyd, S., *High Speed TCP for Large Congestion Windows*. 2003.
2. Kelly, T. *Scalable TCP: Improving Performance in High Speed Wide Area Networks*. in *Scalable TCP: Improving Performance in High Speed Wide Area Networks*. 2003. Geneva.
3. C. Jin, D.W., S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, *Fast TCP: From Theory to Experiments*. IEEE Communications Magazine, 2003.
4. Dina Katabi, M.H.a.C.R. *Congestion Control for High Bandwidth-Delay Product Networks*. in *ACM Sigcomm*. 2002. Pittsburg.
5. Welzl, M. *Traceable Congestion Control*. in *ICQT 2002 (International Workshop on Internet Charging and QoS Technologies)*. 2002. Zürich.
6. <http://www.indiana.edu/~anml/anmlresearch.html>.
7. R. L. Grossman, H.S., S. Bailey. *PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks*. in *Supercomputing 2000*. 2000. Dallas.
8. E. He. J. Leigh, O.Y., T.A. DeFanti, *Reliable Blast UDP: High Performance Bulk Data Transfer*. IEEE Cluster Computing 2002, 2002.
9. <http://dast.nlanr.net/Projects/Iperf/>.
10. <http://netperf.org/netperf/NetperfPage.html>.
11. H. Sivakumar, R.L.G., M. Mazzucco, Y. Pan, Q. Zhang, *Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks*. Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks, 2003.
12. Rizzo, L., *Dummynet: a simple approach to the evaluation of network protocols*. ACM Computer Communication Review, 1997. **27**(1).