ORNL/TM-2004/191

An Evaluation of UDP Transport Protocols

27 August 2004

Prepared by
Steven M. Carter
Research Associate

ORNL-2004/191

AN EVALUATION OF UDP TRANSPORT PROTOCOLS

Steven M. Carter
Tom Dunigan
Florence Fowler

Date Published: August 2004

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF TABLES

Figure **Page**

1. INTRODUCTION

Although the speed of LAN and WAN networking is growing at an exponential rate, the applications that use those networks have not followed suit. With fiber optic interconnects, gigahertz processor speeds, and 10 gigabit per second network interface cards, hardware does not seem to be the limiting factor. It is becoming increasingly obvious that the protocols that are the basis of networking today are ill-suited to a new generation of networking technology. For this reason, Oak Ridge National Laboratory is particularly interested in improving bulk transfers over high-bandwidth, high-latency networks because of its involvement in storage and in the transfer of data for cutting-edge scientific applications. This report summarizes our evaluation of a new group of protocols specifically designed to get more useful bandwidth from today's high speed, wide area networks.

2. BACKGROUND

2.1 TCP

TCP is the most widely used protocol to transfer data over IP networks. This is due in large part to the fact that TCP guarantees in order delivery, no loss of data, and fairness. Unfortunately, the implementation of these features makes TCP ill-suited to get optimal bandwidth from high-speed, high-latency network links. One approach to getting more bandwidth is to fix the problems inherent in TCP. Extensive tests have been conducted to pinpoint problems and suggest possible solutions for these problems. ORNL is a partner in the NET100/WEB100 project. This project endeavors to look inside the kernel by expanding kernel instrumentation and making these kernel variables available for reading and tuning through the linux /proc interface. ORNL has written and modified a number of tools which make use of this newly accessible information: the WAD (Work Around Daemon) , iperf100, ttcp100, WEBD, TRACED, and a web100-enabled bandwidth tester. ORNL has also simulated TCP using the TCP-over-UDP test harness, atou, to simulate the effects of changes to TCP proposed by Sally Floyd and others. Most of these proposals center around Slow Start and Congestion Avoidance.

TCP 's Slow Start and Congestion Avoidance algorithm work in concert to avoid inundating a network link with an initial blast of data, then keeping the flow of data at a rate which will avoid congestion and promote fairness amongst network flows. Congestion Avoidance and Slow Start require that two variables be kept for each connection: a congestion window(cwnd) and a slow start threshold size (ssthresh). Initially, the Slow Start algorithm restricts the sending rate to match the rate at which acknowledgments are returned by the other end of the connection. Unfortunately, long round-trip times can result in large delays in achieving the available rate of the connection. As the connection progresses, cwnd is doubled for each packet acknowledgment (ACK) received until a timeout occurs or duplicate ACK (indicating a lost packet) is received. Upon receiving 3 duplicate ACKs, cwnd is halved and its value stored in ssthresh. If there is a timeout, cwnd is reset to 1 or 2 segments. If cwnd is less than or equal to ssthresh, TCP is in Slow Start; otherwise, Congestion Avoidance takes over and cwnd is incremented by 1/cwnd for each ACK. This is an additive increase, compared to Slow Start's exponential increase. Again, a flow is penalized when round-trip times are long and the available capacity of the path may never be reached.

2.2 NETBLT

NETBLT is a transport layer protocol proposed in 1987 and described in RFC 998. It was specifically designed for high-bandwidth, high-latency networks including satellite channels. NETBLT differs from

TCP in that it uses a rate-based flow control scheme rather than a window-based flow control. The rate control parameters are negotiated during the connection initialization and periodically throughout the connection. The sender uses timers rather than ACKS to maintain the negotiated rate. Since the overhead of timing mechanisms on a per packet basis can lower performance, NETBLT's rate control consists of a burst size and a burst rate with burst_size/burst_rate equal to the average transmission time per packet. Both size and rate should be based on a combination of the capacities of the end points as well as that of the intermediate gateways and networks. NETBLT separates error control and flow control so that losses and retransmissions do not affect the flow rate. NETBLT uses a system of timers to ensure reliability in delivery of control messages and both sender and receiver send/receive control messages.

fc998 gives the following explanation of the protocol: "the sending client loads a buffer of data and calls down to the NETBLT layer to transfer it. The NETBLT layer breaks the buffer up into packets and sends these packets across the network in datagrams. The receiving NETBLT layer loads these packets into a matching buffer provided by the receiving client. When the last packet in the buffer has arrived, the receiving NETBLT checks to see that all packets in that buffer have been correctly received. If some packets are missing, the receiving NETBLT requests that they be resent. When the buffer has been completely transmitted, the receiving client is notified by its NETBLT layer. The receiving client disposes of the buffer and provides a new buffer to receive more data. The receiving NETBLT notifies the sender that the new buffer is ready and the sender prepares and sends the next buffer in the same manner."

As described, the NETBLT protocol is "lock-step". However, a multiple buffering capability together with various timeout/retransmit algorithms give rise to the claim that NETBLT gets good performance over long-delay channels without impairing performance over high-speed LANs. NETBLT, however, is not widely implemented. Unfortunately, being a transport layer design, NETBLT must be implemented at the kernel level. This has caused an impediment to the wide spread implementation of NETBLT. Although it is not being actively implemented, it is noteworthy because many UDP protocols borrow from its design and/or are based on its principals.

3. UDP BASED PROTOCOLS

We evaluated several UDP-based protocols, including SABUL, TSUNAMI, FOBS, UDT, QUANTA, and Hurricane. Each protocol differs on how it manages packet rate and packet loss. Table 1 summarizes the characteristics of SABUL, FOBS, and TSUNAMI.

3.1 CONTROL CHANNELS

With the exception of UDT and QUANTA, all of the proposals use one or more TCP connections for sending/receiving control information in addition to a UDP connection for sending/receiving data. In the case of SABUL and TSUNAMI, the control information is a unidirectional from receiver to sender. FOBS, however, sends control packets in both directions.

3.2 RATE CONTROL

Rate control is used to control the burstiness often observed in TCP flows. This burstiness may cause losses as router queues suddenly fill up and packets are dropped or network interface cards cannot keep up. Also, rate control allows a flow to more quickly fill a pipe without going through the initial ramping-up process characteristic of TCP. Rate control or inter-packet delay, which adjusts to packet loss and/or network congestion as reported by the receiver, has been added in some form to all the UDP protocols to counter the charges of unfair use of capacity and potential to create network problems.

3.2.1 TSUNAMI

TSUNAMI gives the user the ability to initialize many parameters including UDP buffer size, tolerated error rate, sending rate and slowdown/speedup factors with the 'set' command. If the user does not set sending rate, however, it starts out at 1000Mbs with a default tolerated loss rate of 7.9%. Since a block of file data(default 32768) is read and handed to UDP/IP, the rate control is actually implemented per block rather than per packet. The receiver uses the combination of the number of packets received (a multiple of 50) and a timed interval (>350ms) since the last update to determine when to send a REQUEST_ERROR_RATE packet containing a smoothed error rate. If the error rate is greater than the maximum tolerated rate, the sending rate is decreased; if it is less, the sending rate is increased.

3.2.2 SABUL

SABUL begins with a preset IPD (inter-packet delay) of 10 usec which it converts to CPU cycles. The receiver generates a SYN packet based on a timed interval(200ms) which signals the sender to use both the number of lost packets and the number of packets--including retransmits--sent since the last SYN time to calculate a current loss rate. This loss rate is then input to a weighted moving average formula to give a history-rate. If the history-rate is greater then a preset limit(.001), the IPD is increased; if less than the limit, the IPD is decreased; if equal, .1 is added. In a former release, SABUL attempted to keep the loss rate between an upper and lower limit. The latest implementation is similar in concept to TSUNAMI's in that both keep the delay between blocks/packets between an upper and lower limit.

SABUL is the only one of the three to implement the IPD(inter-packet delay) between individual packets as opposed to groups of packets. The delay is implemented by repeated calls to rtdsc() until the requisite number of clock cycles have passed. FOBS checks the sending rate after a burst of packets(25) and implements the delay with a gettimeofday() calculation until time to send the next burst. TSUNAMI uses a timed select() to implement the delay between blocks of data.

3.2.3 RBUDP

RBUDP(QUANTA) will not be evaluated at this time. QUANTA has some very interesting ideas such as forward error correction but has not implemented this and does not yet do file transfers as it is still in the very early stages of development.

3.2.4 FOBS

FOBS asks for the local and remote network interface card speed which it uses to determine a maximum beginning rate. The default tolerated loss rate is 1%. FOBS calculates a table of rates during sender initialization, linking these rates to a network state machine. After a segment of data(about 10000 1466-byte packets) has been transferred, the sender requests an update from the receiver. The reported packet loss from the receiver is used then to calculate the current bandwidth. The current bandwidth is compared against the pre-calculated table values to determine the current state of the network and pull a corresponding rate from the table.

Table 1: Comparison of UDP Protocols

Feature	*SABUL*	*Tsunami*	*FOBS*
TCP Control Port	*Yes--Control packets are sent from receiver to sender* *The sender can also generate and process a pseudo control packet upon the expiration of a timer*	*Yes--Control packets are sent from receiver to sender*	*Yes--2 control ports are used* *Control packets are sent both ways*
UDP Data Port	*Yes--Data is sent from sender to receiver*	*Yes--Data is sent from sender to receiver*	*Yes--Data is sent from sender to receiver*
Threaded Application	*Main thread does file I/O 2nd thread keeps track of timers and sends/receives packets*	*Server forks a process to handle receiver's request Receiver creates a thread for disk I/O*	*The sender and receiver are NOT threaded fobsd is but was not used for these tests*
Rate Control	*Yes--Inter-packet delay implemented by continuous calls to rtdsc()*	*Yes--Inter-block delay implemented by calls to gettimeofday() and select()*	*Yes--Inter-block delay implemented by continuous calls to gettimeofday()*
Tolerated Loss	*0.1%*	*user can set--7.9% default*	*1.0%*
Authentication	*No*	*Yes--via a shared secret*	*No*
Packet Size	*1472*	*32768 (default)*	*1470*
Socket Buffers	*40960000*	*20000000*	*DATA Socket SO_SNDBUF: 100000; SO_RCVBUF: Not Set?*
Congestion Control	*Adjusts sending rate every 200ms based on lost pkt info from recvr*	*Adjusts sending rate after receiving a ctrl pkt: REQUEST_ERROR_RATE from recvr sent after every 350ms or 50 Pkts*	*Very Limited adjustments are made after a "chunk" of data has been sent--does NOT assume lost packets are result of Congestion*
Reorder resilience	*None*	*None*	*Not built in but approx. 10000 pkts are sent before checking so sometimes things resolve themselves*
Duplicates	*Yes--losses and reordering cause unecessary retransmits. Most losses occur at start of transfer as IPD is not user adjustable*	*Since IP fragmentation is used, if one packet is lost in the block, all are resent. Also the last pkt is resent until a REQUEST_STOP pkt is recvd*	*Yes--Often resends the last 10000 pkts while waiting for WRITECOMPLETEPKT from recvr*
Diagnostics	*None*	*Some error messages are displayed*	*Prints out a summary of the transfer as it progresses*

4. TESTING

To evaluate the protocols, a series of tests were conducted over the Internet and a NISTNET testbed.

4.1 INTERNET TESTS

4.1.1 TEST SETUP

Firebird was used as the test host at ORNL. It has a single 1.4GHz Intel Pentium IV and 512MB of RAM running a Linux 2.4.20 kernel with web100 enhancements. It is connected with a SysKonnect Gigabit Ethernet NIC via a jumbo frame enabled VLAN to a Cisco 6500 switch. Tests indicate speeds of approximately 27MB/s writing to and 31MB/s reading from disk.

Net100 is the test endpoint at LBL. It has a single 1.4GHz AMD Athlon 4 and 256MB of RAM running a Linux 2.4.10 kernel with web100 enhancements. It is connected via Fast Ethernet. Tests indicate speeds of approximately 22MB/s writing to and 30MB/s reading from disk.

Ping shows a round-trip time of 68ms between firebird and net100.

4.1.2 MEMORY TO MEMORY

200 MByte Memory-to-memory tests were run from ORNL to LBL with the following results (Lost is the total figure the receiver reports to the sender as lost--based on how the application reports losses, the same losses could be reported more than once):

Table 2: 200MB Memory to Memory test from ORNL to LBL

Application	Rate (Mb/s)	Sent	Received	Lost	Re-sent	Duplicate
Tsunami	307.23	13361	13102	55	259	204
SABUL	343.49	140015	140005	15	15	5
FOBS	317.17	143147	138436	2816	6720	2009
Quanta	358.28	137742	137669	73	73	0
Iperf	489	145499	145499	0	0	0
Iperf100 -P 3	279.7	171303	171299	6	6	3

Table 3: 200MB Memory to Memory test from LBL to ORNL

Application	Rate (Mb/s)	Sent	Received	Lost	Re-sent	Duplicate
Tsunami	243.92	14963	13330	1689	1689	228
SABUL	151.76	184895	142441	47838	44895	2441
FOBS	256.17	142530	138484	2170	6103	2057
Quanta	310.37	142157	137742	4415	4415	0
iperf	420	142846	142843	3	0	0
Iperf100 -P 3	229.8	145491	145484	5	7	2

4.1.3 DISK TO DISK

Table 4: 200MB file transfer from LBL to ORNL (Best Case/Worst Case)

Application	Rate (Mb/s)	Sent	Received	Lost	Duplicate
Tsunami	47.17/8.2	146419/-	181840/-	44400/-	35421/-
SABUL	52.14/ 37.25	146220/ 146220	178297/ 216482	5463/ 4200	32077/ 70262
FOBS	45.01/ 28.52	146421/ 146421	151624/ 147787	2668/ 3503	5203/ 1366

4.2 NISTNET

To validate results obtained over the broader Internet where conditions are unpredictable and constantly changing, tests were performed under more controlled conditions using NISTNET. NISTNET is running on an old, slow Gateway machine, viper, with 64MB of memory and two 100Mbs Network Interface cards. One NIC is connected into a NETGEAR Fast Ethernet Switch and one into a local area network. The other two machines involved are dual processor pcs previously used in a cluster and have 512MB of memory and a 100Mbs Network Interface card. Pinto is connected into the same local area network and pinto10 is connected into the NETGEAR Switch. Conditions are not completely controlled in the local area network, but are observed to be mostly stable with rare exceptions.

4.2.1 TEST SETUP

NISTNET was configured to impart a 35ms delay in each direction. Ping verified the configuration, showing a round-trip time of 71ms.

4.2.2 MEMORY TO MEMORY

Table 5: 200MB memory to memory transfers over NISTNET (0 drop rate).

Application	Rate (Mb/s)	Sent	Re-sent	Lost	Duplicate
Tsunami	83.1	13103	1	0	1
SABUL	87.1	157823	0	0	0
FOBS	80.8	136570	143	1	142
Quanta	89.9	137742	0	0	0
iperf(TCP) -P 3	78.8	150571	7	7	0
iperf(UDP)	90.5	153849	0	0	0

Table 6: 200MB memory to memory transfers over NISTNET (.0107 drop rate).

Application	Rate (Mb/s)	Sent	Re-sent	Lost	Duplicate
Tsunami	82.5	13151	18	18	49
SABUL	83.5	154172	14172	47744	4230
FOBS	79.3	137642	1215	16	1199
Quanta	87.5	137727	15	15	0

4.2.3 DISK TO DISK

Table 7: 200MB file transfer over NISTNET (0.0107 drop rate).

Application	Rate (Mb/s)	Sent	Received	Lost	Re-sent	Duplicate
Tsunami	33.2	354961	176184	181210	207317	29765
SABUL	87.2	166131	148317	11544	19911	2097
FOBS	71.4	151175	150537	17	4754	4116

5. ANALYSYS

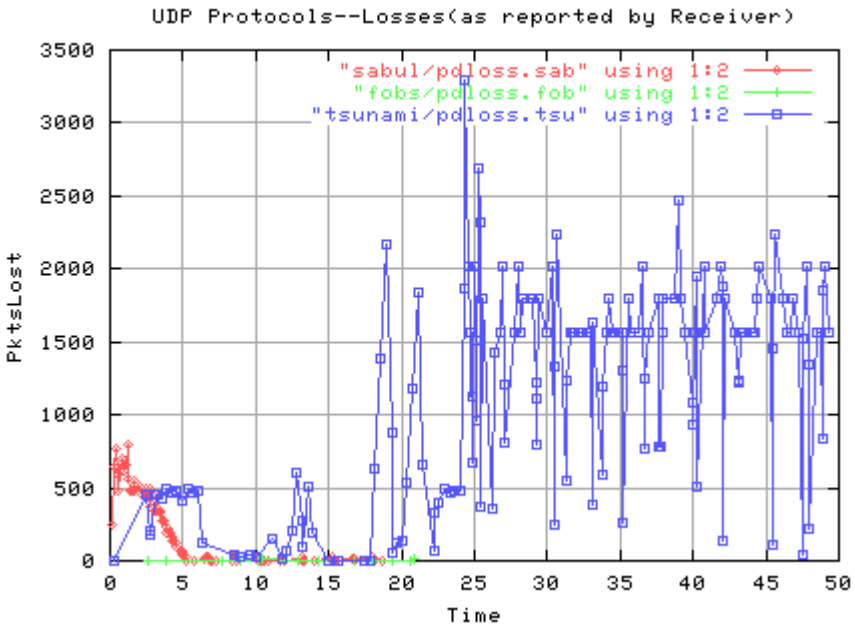

Figure 1: Losses reported by the receiver.

5.1 PACKET LOSS

The FOBS client reports losses upon notification from the server that a segment or chunk of data has been sent. The client sends a bitmap indicating the status of all packets it expects to receive in the current transfer. Figure 1 illustrates the reporting of losses at the end of each segment. FOBS reported a total of 35 lost packets, which is within the range expected.

SABUL notifies the server of losses in two ways. As soon as the SABUL client receives a packet number greater than the one expected, the server is notified of the loss. Also, every 20ms a collective loss report is sent. The client in this instance reported losses of 11544 packets and re-reported 8497 of these losses again in the periodic reports. The reason for SABUL's high loss rate is unknown. The IPD(inter-packet delay) starts out at 10us but in this transfer, was slowly increased to 118 us and generally stayed between 110 and 117 us. Many of the losses occur at the beginning of the transfer when the sending rate is high. The IPD cannot be adjusted by the user.

The TSUNAMI client also keeps a tally of lost packets and sends retransmit requests after receiving a multiple of 50 blocks if a preset interval(>350ms) has passed since the last request. Before sending the requests, the list of lost blocks is cross-checked with a tally of all blocks received so far to eliminate any lost blocks that may have come in. Even with the cross check, the client in this transfer issued 181210 retransmit requests. As shown in the graph, many of the requests were repeats and a large portion occur at the end of the transfer. Looking at the data reveals that 223 packets in a range between packet number 142033 and packet number 146411 were received over 100 times. Each packet was sent 722 times with the last packet being sent 1616 times and received 708 times.

5.2 DUPLICATE PACKETS

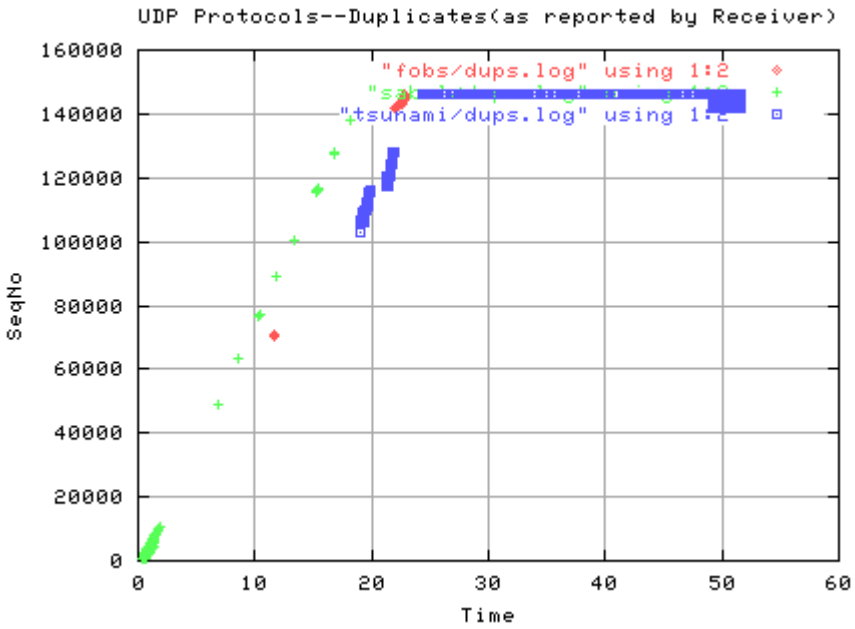

Figure 2: Duplicates reported by the receiver.

Most of SABUL's duplicate packets come at the beginning of the flow. Looking at the duplicates reported by the **SABUL** client, each of the 2097 duplicate packets were received exactly twice but the server sent the 19911 retransmits anywhere from 2 to 10 times. This may be the result of duplicate reporting or the fact that ACKS are perhaps spaced too far apart (ACKS are sent every 100ms).

The graph of TSUNAMI's duplicates seems to indicate some confusion or mis-communication near the end of the transfer. In addition, TSUNAMI continues to retransmit the last block/packet until receiving a REQUEST_STOP from the receiver. In one short, loss-free transfer from ORNL to LBL of 8,135 blocks/packets of size 1466, the last block was observed being transmitted 11,876 times. The client actually received 7,835 of the 11,876 before it quit. REQUEST_RESTART requests seem to cause the instability. When REQUEST_RESTARTs are sent by the receiver, often both sender and receiver had to be manually stopped as both somehow seemed to get confused and the rate fell below 4Mbps. This happened regularly on transfers from ORNL to LBL with files of 100mb or more. The message, "FULL! -- ring_reserve() blocking" also appeared at the client regularly during the transfer of large files. In order to complete the necessary file transfers with TSUNAMI, the retransmit table was enlarged and a block size of 16384 used in an attempt to eliminate REQUEST_RESTART requests.

FOBS keeps sending packets while waiting to hear from the client that a "chunk" of data has been written to the file. This means that the duplicates are mostly clustered at the end of each chunk. FOBS also transmits many unnecessary packets. In a file transfer involving 146,421 data packets(214649928 Bytes) and no losses, FOBS actually sent 150,100 packets. These packets are apparently sent while the sender is awaiting instructions from the receiver telling it what needs to be done next. In this case, after sending the first chunk, packets 70000-71229 were re-transmitted. Similar re-transmissions occurred after the second and third chunks. The receiver read all 71527 packets in the first chunk, sent a COMPLETEDPKT, and got ready for the next chunk. Before reading the first packet in the next chunk, packets 70000-71229 were read and thrown away. A 'scaleFactor' is used to keep the packets in sync with the correct iteration or the receiver might assume these retransmits are part of the next chunk.

5.3 AVERAGE BANDWIDTH

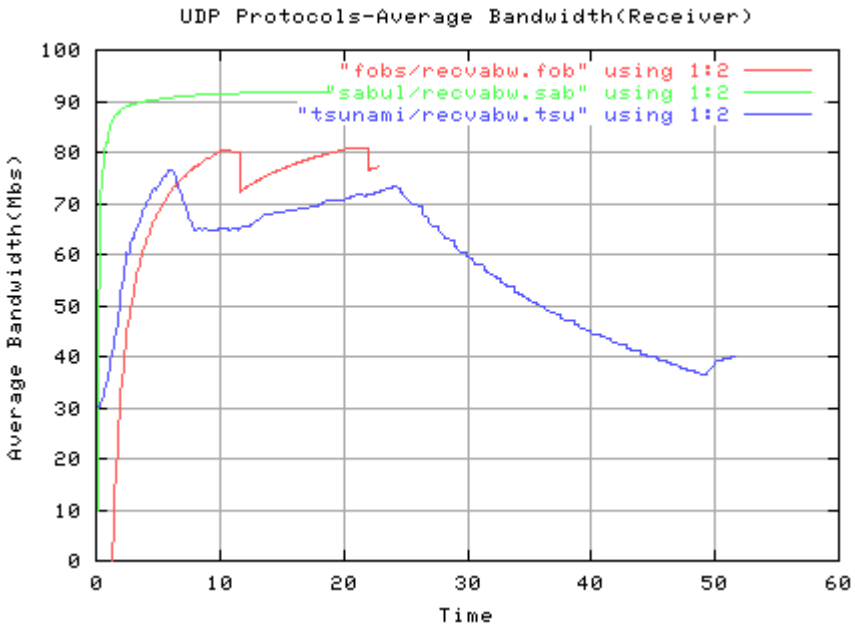

Figure 3: Average bandwidth reported by receiver.

The SABUL sender implements a delay between each packet whereas the other two applications implement a delay between blocks of packets similar to NETBLT. Even more important with regard to comparisons, SABUL attempts to adjust the sending rate about every 200ms based on lost-packet information. As mentioned above, the IPD starts out at 10us and, in this case, gradually increases to 118us. It then settles in at 110-118us for the main part of the transfer. There were 132 rate calculations performed by the sender during the transfer time of 18.79 seconds. The other two protocols operate in a more lock-step manner by transmitting new data in blocks or groups of packets before doing any retransmits or rate adjustments.

In the LAN file transfers above, SABUL usually wins the bandwidth prize. The exceptions occur when there are EXP(expiration) events. These are generated by the server if no ACK or ERR packet has been received during a specified interval(1000ms). The server then assumes all packets sent in the current period have been lost and adds them to the lost list. In runs using the NISTNET testbed, throughputs of 84.1, 87.2, and 87.4 Mbs were observed. But in one case where EXP events were generated, SABUL only achieved 30.9 Mbs.

FOBS transmits 10000 packets(size 1466) before recalculating the sending rate and a chunk-size of new data before doing any retransmits. FOBS actually calculates the bandwidth for the last 25 packets sent and, if it is greater than the desired rate, spins until it is less than or equal to the desired rate. In this case, the desired sending rate started out at 86 Mbs and because of the low packet loss, did not change. There were 17 rate calculations in the transfer time of 22.95 seconds.

FOBS' file transfer performance suffers because it is lock-step. In doing actual file transfers, the sender transmits one chunk of data and then waits for the receiver to signal that it has written the whole chunk to disk. Depending on RTT and disk speed, that can add up. With the 70ms RTT, FOBS achieved 79.9Mbs if the time spent writing the file was not counted. To be fair however, FOBS is the most consistent in it's performance with measured throughput of 72.1, 71,4, and 72.1 Mbs on runs in the NISTNET testbed.

TSUNAMI's algorithm is a little more obscure in that the receiver sends a rate request after a combination of 50 blocks(you define block size) and a preset interval(>350ms) has passed since the last rate request. The receiver in this example sent 87 rate requests in a transfer time of 51.72 seconds. The sender checks the sending rate between each block. If blocksize is the default(32768), this means the delay is implemented every 23 or so packets. If blocksize is 1466, as in this example, the delay becomes a true inter-packet delay. The max rate can be set by the user with the TSUNAMI client set command. In this case the rate was set to 80 Mbs giving a beginning inter-block delay of 438 us. This is not implemented directly however. After calculating the sending time for the last block, the delay is calculated as:

```
delay = ((sending-time + 50) < ipd_current) ?
  (ipd_current - delay - 50)) : 0;
```

A select() call is used to implement the delay.

TSUNAMI gives the least consistent performance even in the semi-controlled NISTNET environment. Using NISTNET, throughputs of 18.7, 33.2, 42.6, and 84.1 Mb/s were observed. The results illustrate the prolonged sending at the end of the transfer that has been mentioned before.

5.4 REORDERING/LOSS

TSUNAMI, SABUL and FOBS all do well as long as there is little reordering or loss. Reordering and/or loss seem to cause them all to transmit unneeded duplicates. FOBS gives priority to new data and transmits a chuck (default 100MB) of data before doing retransmits. TSUNAMI and SABUL give priority to requests for retransmits. The SABUL client reports any missing packets immediately as well as periodically every 20ms. The TSUNAMI client reports missing packets after the requirements for numbers of packets received (a multiple of 50) and amount of time since the last report (more than 350ms) have been satisfied.

SABUL also has problems when there is significant reordering/loss. Since the client reports every perceived loss immediately, this can mean a lot of control packets. In one case, the sender was observed having to deal with one control packet for nearly every data packet it was sending. Since a tabulation of lost packets is also sent every 20ms, the same loss may be reported more than once and the sender will count it again. This gives rise to an interesting phenomenon. If the tabulation of losses for the last period is greater than the number of packets sent during the same period, the sender solves the problem by assuming 100% loss for that period. With a shorter RTT (this scenario was with 150 usecs), perhaps fewer packets would be sent; but it is assumed these UDP protocols are meant for transferring large files on high-bandwidth, high-delay networks.
After much testing and studying, it is still not clear why SABUL and TSUNAMI report so many losses. One clue may be that the protocols that wait 10000 packets(FOBS) or more (iperf & quanta) to report losses do much better. That would seem to indicate that packets may not be received in strict order but more study needs to be done on this problem.

6. RESULTS & SUBSEQUENT WORK

Our tests over the Internet and NISTnet indicate that each of the UDP protocols has a number of shortcomings. Tsunami has several instabilities and often yields poor throughput as well as sending many duplicate packets. FOBS suffers because of its lock-step protocol. SABUL yields the best performance, and we expect its follow-on, UDT, to perform even better. None of the protocols perform well with packet re-ordering. Though all three protocols attempt to adapt to congestion, it is not clear how fair these

protocols are to competing TCP protocols. They don't appear to have a clear advantage over aggressive TCP protocols like Kelly's scalable TCP or Sally Floyd's HS TCP, but the UDP protocols do not require kernel modifications. They are probably best suited to enterprise subnets of the Internet.
As a follow-on to this work, ORNL is currently investigating and developing UDP protocols to be used solely on high-speed, dedicated links. One such protocol, Hurricane, was tested along with Tsunami, UDT, and FOBS over a dedicated loopback connection between ORNL and Atlanta.

6.1 SETUP

In order to incur a delay in packet transit, a loopback connection was created between ORNL and Atlanta (Figure 4). Since a dedicated circuit was not available for this route, one was emulated over a lightly loaded OC-192 circuit. The test host's throughput was limited to 1Gb/s per host to the ORNL router. Since there was plenty of unused bandwidth on the OC-192, each host is able to get the full line rate over the loopback connection.

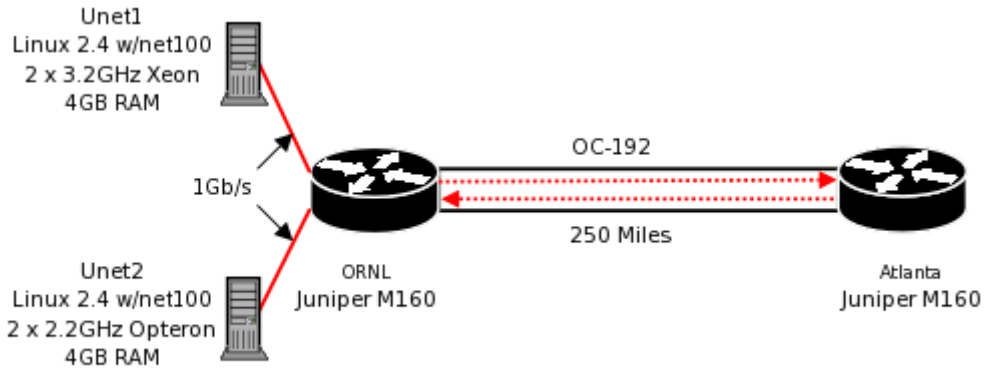

Figure 4: ORNL to Atlanta loopback testbed.

Filter Based Forwarding was used on the ORNL Juniper router to make it subvert its normal behavior of routing local interfaces to each other. Instead, the router forwards the packets to the Atlanta router. In turn, the Atlanta router sends the packets back to the ORNL router, completing the loopback connection. The 250 mile round trip resulted in a 10ms round trip time.

In order to attain the enough throughput from the host's filesystems to drive the network connection, a series of tests and modifications were made to the hosts's I/O susbsystems. With the modifications, each hosts was able to deliver more than the 1Gb/s needed.

6.2 UDT

Between the time of the previous tests and the dedicated link tests, UDT was released as a follow on to SABUL. UDT differs from SABUL in that it does not use a TCP based control channel. The lack of TCP control channel made UDT particularly interesting since Hurricane is also a UDP only protocol.

6.3 RESULTS

Table 8: ORNL to Atlanta loopback test results.

Application	Rate (Mb/s)
Hurricane	991
Tsunami	919
UDT	890
FOBS	708

The results were very promising. Hurricane out performed the other protocols tested and matched the bandwidth attained by iperf. With further development, it is hoped that Hurricane can provide the high data rate and low jitter needed by applications on dedicated networks such at the UltraScience Network.