Reliable Blast UDP : Predictable High Performance Bulk Data Transfer

Eric He, Jason Leigh, Oliver Yu,
Thomas A. DeFanti

Electronic Visualization Laboratory
University of Illinois at Chicago
cavern@evl.uic.edu
www.evl.uic.edu/cavern

Abstract

High speed bulk data transfer is an important part of many data-intensive scientific applications. This paper describes an aggressive bulk data transfer scheme, called Reliable Blast UDP (RBUDP), intended for extremely high bandwidth, dedicated- or Quality-of-Service-enabled networks, such as optically switched networks. This paper also provides an analytical model to predict RBUDP's performance and compares the results of our model against our implementation of RBUDP. Our results show that RBUDP performs extremely efficiently over high speed dedicated networks and our model is able to provide good estimates of its performance.

1 Introduction

Quanta (the Quality of Service Adaptive Networking Toolkit) [10] is a toolkit based on our prior work on CAVERNsoft, which provided a set of APIs for bridging graphics and networking, to allow developers to more easily build applications for collaborative, immersive environments [5][6][9]. CAVERNsoft, and consequently Quanta provide a rich set of tools and data distribution mechanisms including: message passing, distributed shared memory, remote procedure calls, remote file I/O, forward error corrected UDP, parallel TCP for bulk data transfer, and collaborative performance monitoring. Work in Quanta is underway to develop mechanisms for enabling dedicated light path reservations (often described as "lambdas") on optically switched networks; and new protocols for maximizing data delivery over these lambdas. Quanta's experimental testbeds are two optically switched networks Starlight and OMNInet. Starlight is a project managed by the University of Illinois at Chicago, to provide an IP-over-Dense Wave Division Multiplexing (DWDM) peering point for national and international optical networks. The goal is to develop a "petri dish" for growing an experimental, optically connected Grid whereby clusters of computing resources can directly "dial-up" lambdas between them and use the extreme quantities of bandwidth (on the order of 1-10 Gigabits/s) as a long distance system bus [11]. OMNInet is a project supported by Nortel Networks, SBC Communications Inc. and Ameritech to assess and validate next-generation optical technologies, architectures and applications in metropolitan area networks [8].

In this paper we address one aspect of our work on Quanta- the development of an aggressive bulk data transfer scheme intended for high bandwidth, dedicated- or Quality-of-Service- enabled networks, such as those on StarLight and OMNInet. In the following sections we will introduce the problem of bulk data transfer over long fat networks; provide an algorithm for a transfer scheme, called Reliable Blast UDP (RBUDP); propose an analytical model to predict its performance; and compare the results of our model against our implementation of RBUDP.

2 The Problem of Bulk Data Transfers

Even if networked applications could make Gigabit "lambda reservations," it does not however guarantee that they will be able to make full use of that bandwidth. This problem is particularly evident when one attempts to perform large bulk data transfers over long distance, high speed networks (often referred to as "long fat networks" or LFNs) [12].

LFNs such as those between the US and Europe or Asia have extremely high round-trip latencies (at best 120ms). This latency results in gross bandwidth *under*-utilization when TCP is used for data delivery. This is because

E. He, J. Leigh, O. Yu, T. A. DeFanti, *Reliable Blast UDP : Predictable High Performance Bulk Data Transfer*, IEEE Cluster Computing 2002, Chicago, Illinois, Sept, 2002.

TCP's windowing mechanism imposes a limit on the amount of data it will send before it waits for an acknowledgement. The long delays that occur over international networks means that TCP will spend an inordinate amount of time waiting for acknowledgments, which in turn means that the client's data transmission will never reach the peak available capacity of the network. Traditionally this is "remedied" by adjusting TCP's window and buffer sizes to match the *bandwidth * delay* product (or capacity) of the network. For example, for a 1Gbps connection between Chicago and Amsterdam, with an average round trip time of 110ms, the capacity is 1024*0.11/8 = 14.1 Mbytes. Adjusting TCP window size is problematic for several reasons: firstly, on some operating systems (such as IRIX for the SGI,) the window size can only be modified by building a new version of the kernel- hence this is not an operation a user-level application can invoke. Secondly, one needs to know the current capacity of the network in order to set the window size correctly. The current capacity varies with the amount of background traffic already on the network and the path to the destination.

Several alternative solutions are possible. One solution is to provide TCP with better estimates of the current capacity of a link. This is the approach of the WEB100 Consortium [14]. The consortium is developing techniques to modify router operating systems to report available bandwidth over a network link. Furthermore they are modifying operating systems kernels to allow better monitoring of TCP performance. Another solution is to use striped (or parallel) TCP [Park00, Leigh01, Allcock01]. In parallel TCP, the payload is divided into N partitions which are delivered over N TCP connections. Both Leigh (in CAVERNsoft) and Allcock (in GridFTP) have shown that parallel TCP can provide throughput as high as 80% of a network's available bandwidth, however its performance is unstable when excessive numbers of sockets are used. Furthermore it is difficult to predict the correct number of sockets to use.

In this paper we take a more aggressive approach by using UDP augmented with aggregated acknowledgments to provide a reliable bulk data transmission scheme. We call this Reliable Blast UDP (RBUDP). A similar scheme called NetBLT was first proposed in 1985 (RFC969) by Clark et al [3]. We extend Clark's work by providing both analytical and experimental results to show that RBUDP can provide the performance predictability that is lacking in parallel TCP. Furthermore we will provide an equation similar to TCP's *bandwidth*delay* product to allow one to predict RBUDP performance. This prediction will be useful in the future, for network resource reservation on the Grid.

It is important to remember that we intend aggressive protocols such as parallel TCP and Reliable Blast UDP for high speed dedicated links or links over which quality of service is available. We do not intend these protocols for use over the broader Internet.

3 Reliable Blast UDP

Reliable Blast UDP has two goals. The first is to keep the network pipe as full as possible during bulk data transfer. The second goal is to avoid TCP's per-packet interaction so that acknowledgments are not sent per window of transmitted data, but aggregated and delivered at the end of a transmission phase. Figure 1 below illustrates the RBUDP data delivery scheme. In the first data transmission phase (A to B in the figure), RBUDP sends the entire payload at a user-specified sending rate using UDP datagrams. Since UDP is an unreliable protocol, some datagrams may become lost due to congestion or an inability of the receiving host from reading the packets rapidly enough. The receiver therefore must keep a tally of the packets that are received in order to determine which packets must be retransmitted. At the end of the bulk data transmission phase, the sender sends a DONE signal via TCP (C in the figure) so that the receiver knows that no more UDP packets will arrive. The receiver responds by sending an Acknowledgment consisting of a bitmap tally of the received packets (D in the figure). The sender responds by resending the missing packets, and the process repeats itself until no more packets need to be retransmitted.

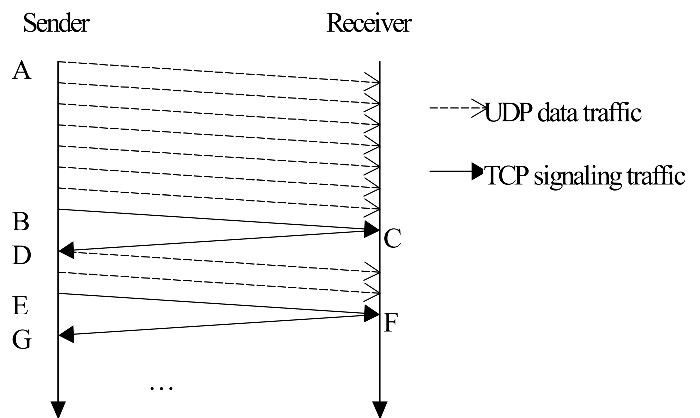

Figure 1. The time sequence diagram of RBUDP

In RBUDP, the most important input parameter is the sending rate of the UDP blasts. To minimize loss, the sending rate should not be larger than the bandwidth of the bottleneck link (typically a router). Tools such as Iperf [5] and netperf [8] are typically used to measure the bottleneck bandwidth. In theory if one could send data

just below this rate, data loss should be near zero. In practice however, other factors need to be considered. In our first implementation of RBUDP, we chose a send rate of 5% less than the available network bandwidth predicted by Iperf. Surprisingly this resulted in approximately 33% loss! After further investigation we found that the problem was in the end host rather than the network. Specifically, the receiver was not fast enough to keep up with the network while moving data from the kernel buffer to application buffers. When we used a faster computer as the receiver, the loss rate decreased to less than 2%. The details of this experiment are further discussed in Section 5.

The chief problem with using Iperf as a measure of possible throughput over a link is that it does not take into account the fact that in a real application, data is not simply streamed to a receiver and discarded. It has to be moved into main memory for the application to use. This has motivated us to produce app_perf (a modified version of iperf) to take into account an extra memory copy that most applications must perform. We can therefore use app_perf as a more realistic bound for how well a transmission scheme should be able to reasonably obtain. In the experiments detailed in Section 5, we however include both iperf and app_perf's prediction of available bandwidth.

Three versions of RBUDP were developed:

1. RBUDP without scatter/gather optimization – this is a naïve implementation of RBUDP where each incoming packet is examined (to determine where it should go in the application's memory buffer) and then moved there.
2. RBUDP with scatter/gather optimization – this implementation takes advantage of the fact that most incoming packets are likely to arrive in order, and if transmission rates are below the maximum throughput of the network, packets are unlikely to be lost. The algorithm works by using readv() to directly move the data from kernel memory to its predicted location in the application's memory. After performing this readv() the packet header is examined to determine if it was placed in the correct location. If it was not (either because it was an out-of-order packet, or an intermediate packet was lost), then the packet is moved to the correct location in the user's memory buffer.
3. "Fake" RBUDP – this implementation is the same as the scheme without the scatter/gather optimization except the incoming data is never moved to application memory. This was used to examine the overhead of the RBUDP protocol compared to raw transmission of UDP packets via Iperf.

Experiments that compare these versions of the protocol, and an analytical model of RBUDP, will be presented in Section 5 and 4 respectively.

4 Analytical Model for RBUDP

The purpose of developing an analytical model for RBUDP is two-fold. Firstly we wanted to develop an equation similar to the "bandwidth * delay product" equation for TCP, to allow us to predict RBUDP performance over a given network. Secondly we wanted to systematically identify the factors that influenced the overall performance of RBUDP so that we can predict how much benefit any potential enhancement in the RBUDP algorithm might provide.

First of all, all variables are defined as follows:

$B_{achievable}$ = achievable bandwidth
B_{send} = chosen send rate
S_{total} = total data size to send (ie payload)
T_{total} = total predicted send time
T_{prop} = propagation delay
$T_{udpSend_i}$ = time to send UDP blast on i^{th} iteration.
$S_{udpSend_i}$=size of UDP blast (initial size is payload size)
N_{resend} = number of times to resend (depends on loss%)
T_{ack} = time to acknowledge a blast (at least 1 ACK is always needed)
L_i = % packet loss on i^{th} iteration

In our model we are attempting to predict the achievable bandwidth ($B_{achievable}$)of RBUDP:

$$B_{achievable} = \frac{S_{total}}{T_{total}} \tag{1}$$

Following the RBUDP algorithm, we estimate T_{total} as:

$$\begin{aligned} T_{total} &= \left(T_{prop} + T_{udpSend_0}\right) \\ &+ \left(\sum_{i=1}^{N_{resend}} (T_{prop} + T_{udpSend_i})\right) \\ &+ \left((N_{resend}+1) * (T_{ack} + T_{prop})\right) \end{aligned} \tag{2}$$

In (2), the first term is the time to send the main payload, the second term is the time to transmit missing packets, called T_{resend}, the last term is the time to send each acknowledgement.

Specifically:

$$T_{udpSend0} = \frac{S_{total}}{B_{send}}$$

$$T_{udpSend_i} = \frac{L_{i-1} * S_{udpSend_i-1}}{B_{send}}$$

$$T_{ack} = \frac{S_{ack}}{B_{send}}$$

$$S_{ack} = \left(\frac{S_{total}}{S_{packet}}\right) / 8$$

$$T_{ack} = \left(\frac{S_{total}}{8 * S_{packet}}\right) / B_{send}$$

S_{packet} = 1.5Kbytes

Consequently:

$$T_{total} = \left(T_{prop} + \frac{S_{total}}{B_{send}}\right) + \left((N_{resend} * T_{prop}) + \sum_{i=1}^{N_{resend}} \frac{L_{i-1} * S_{udpSend_i-1}}{B_{send}}\right) + \left((N_{resend} + 1) * \left(\frac{S_{total}}{8 * S_{packet} * B_{send}} + T_{prop}\right)\right) \quad (3)$$

Given this equation, let us consider two possible situations - one where no loss occurs, and one where loss does occur. If no loss occurs, we can eliminate the middle term so that the best achievable performance can be computed using:

$$T_{best} = \left(T_{prop} + \frac{S_{total}}{B_{send}}\right) + \left(\frac{S_{total}}{8 * S_{packet} * B_{send}} + T_{prop}\right)$$

$$B_{best} = \frac{S_{total}}{\frac{S_{total}}{B_{send}} + \frac{S_{total}}{8 * S_{packet} * B_{send}} + 2T_{prop}} \quad (4)$$

In the denominator, $\frac{S_{total}}{8 * S_{packet} * B_{send}}$ is very small compared to other factors and can be omitted.

We can then derive the ratio of B_{best} and B_{send} as:

$$\frac{B_{best}}{B_{send}} = \frac{1}{1 + \frac{RTT * B_{send}}{S_{total}}} \quad (5)$$

where:

$2*T_{prop}$ is RTT (Round Trip Time).

This ratio shows that in order to maximize throughput, we should strive to minimize $\frac{RTT * B_{send}}{S_{total}}$ by maximizing the size of the data we wish to deliver. For example, given T_{prop} for Chicago to Amsterdam is 55ms, and B_{send} is 600 Mbps, and if we wish to achieve a throughput of 90% of the sending rate, then the payload, S_{total} needs to be at least 74.25 Megabytes.

In Section 5 (Figure 4) we will use equation 3 to compare the theoretical best rate B_{best} against experimental results, over a variety of send rates (B_{send}).

Furthermore we will compare B_{best} against experimental results with varying payload sizes (S_{total}) (Section 5, Figure 6).

Now let us turn to consider the situation where loss does occur. We will take a simplifying assumption that a constant loss rate of L occurs at every pass of the algorithm. We realize that in a real network subsequent losses in the retransmit phases should be smaller, rather than constant, because we will be retransmitting a significantly smaller payload at each iteration. However to estimate that accurately would require us to develop a model for the buffer in the intervening routers too. Hence we can take our simplifying assumption as a worst-case estimate.

So, given loss rate L, retransmits will occur until the amount of data left is less than 1 packet. That is:

$$S_{total} * L^{Nresend} < S_{packet}$$

Therefore:

$$L^{Nresend} = S_{packet} / S_{total}$$

$$\Rightarrow N_{resend} = \lfloor \log_L (S_{packet} / S_{total}) \rfloor \quad (6)$$

The data size of all retransmits is therefore:

$$S_{resend} = S_{total} * \frac{L(1 - L^{\lfloor \log_L (S_{packet} / S_{total}) \rfloor})}{1 - L} \quad (7)$$

We can now plug (6) and (7) back into equation (3) to produce our new estimate of $B_{achievable}$ given constant loss rate L. In Section 5 (Figure 6) we will put this prediction to use comparing an experimental situation where packet loss was observed.

5 Experimental Results

The testbed network consisted of an OC-12 link (622Mbps) brought by SURFnet from Amsterdam to the StarLight facility in Chicago. There was little-to-no traffic on the link when the experiments were performed. Linux PCs were placed at each end of the link. The specifications of each PC are shown in Table 1 below. Keenhond (in Amsterdam) was the faster PC, Prusin (in Chicago) was the slower one.

Host Name	CPU	Memory Size	System Bandwidth
keeshond.nikhef.nl (Amsterdam)	Pentium III 1.0GHz	2.0G Bytes	258 MByte/s
prusin.sl.startap.net (Chicago)	Pentium III 650MHz	768M Bytes	171 Mbytes/s

Table 1. Specification of host PCs in the experimental testbed

In the first set of experiments, data was sent via RBUDP from the faster PC to the slower PC (from Amsterdam to Chicago). In the second set of experiments data was sent in the opposite direction. This allowed us to examine the performance of RBUDP when the bottleneck was either at the processor or in the network. The three versions of RBUDP described in Section 3 were compared against predicted results from our analytical model. A third set of experiments examined RBUDP throughput for different payload sizes.

5.1 From the Fast PC to the Slow PC (Amsterdam to Chicago) – when the Bottleneck is in the Receiving Host Computer

In this experiment, Iperf measured maximum available bandwidth at 576 Mbps, and app_perf measured maximum possible throughput at 490 Mbps. In Figure 2 we plot these thresholds as lines across the top of the graph. Plotting the achieved throughput at various sending rates for the three RBUDP algorithms we notice that at sending rates below the network capacity, RBUDP performs well. I.e. RBUDP gives the application exactly what the application asks for. We also notice that as the sending rates approach the capacity of the network, Fake RBUDP achieves almost the same throughput as Iperf, and the lack of scatter/gather optimization begins to hurt performance because the under-powered CPU is unable to keep up with handling the incoming packets.

5.2 From the Slow PC to the Fast PC (Chicago to Amsterdam) – when the Bottleneck is in the Network

We repeated the experiment in the opposite direction. This time the bottleneck is in the network rather than in the receiving PC. Figures 4 and 5 show that when the host computer is fast enough iperf and app_perf performances match as do the different implementations of RBUDP. Furthermore there is a close match between our experimental results and our prediction from equation 4 (which estimated RBUDP performance when loss rate is zero.)

5.3 Effect of Payload Size on Throughput

From the analysis in Section 4, we know that the propagation time is the primary factor affecting RBUDP overhead. For smaller payloads, the time spent in the acknowledgement phase is almost constant while the time spent blasting UDP packets decreases. In Figures 6 we compare an experimental situation where we send data at 550Mbps (experiencing no loss) against our theoretical prediction which assumes no loss (equation 3.) Furthermore we compare an experimental situation sending data at 610Mbps experiencing 7% loss, against our theoretical prediction where we assume a constant 7% loss per iteration.

Firstly, the results show that RBUDP performs best for large payloads. Secondly the results show that a 7% packet loss does not impact throughput greatly for large payloads. Thirdly our analytical models for no loss and 7% loss provide good boundaries for our experimental results.

6 Conclusions

RBUDP is a very aggressive protocol designed for dedicated- or QoS-enabled high bandwidth networks (such as our aforementioned DiffServ and IP-over-DWDM testbeds). It eliminates TCP's slow-start and congestion control mechanisms, and aggregates acknowledgments so that the full bandwidth of a link is used for pure data delivery. For large bulk transfers, RBUDP can provide delivery at precise, user-specified sending rates. RBUDP performs at its best for large payloads rather than smaller ones, because with smaller payloads the time to deliver the payload approaches the time to acknowledge the payload. The scatter-gather algorithm to reduce memory copies, provides better performance over the non-scatter-gather algorithm for slower CPUs when the loss rate is not very high. This benefit is expected to increase for faster networks.

We have provided an analytical model that provides a good prediction of RBUDP performance. This prediction can be used as a rule of thumb in a manner similar to the *bandwidth * delay* product for TCP. Furthermore this prediction can be used to estimate how future ideas for improving the algorithm might impact RBUDP performance.

Work has begun to combine our work with similar work at the Laboratory for Advanced Computing, at the University of Illinois at Chicago to add rate and congestion control to RBUDP to produce a complete data transfer protocol called *LambdaFTP*, for parallelized data distribution over optically switched networks.

7 Acknowledgments

We would like to thank Cees de Laat at University of Amsterdam for providing the endpoint at SARA in Amsterdam to perform these experiments.

The virtual reality and advanced networking research, collaborations, and outreach programs at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago are made possible by major funding from the National Science Foundation (NSF), awards EIA-9802090, EIA-9871058, EIA-0115809, ANI-9980480, ANI-9730202, ANI-0123399 and ANI-0129527, as well as the NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement ACI-9619019 to the National Computational Science Alliance. EVL also receives funding from the US Department of Energy (DOE) Science Grid program and the DOE ASCI VIEWS program. In addition, EVL receives funding from the State of Illinois, Microsoft Research, General Motors Research, and Pacific Interface on behalf of NTT Optical Network Systems Laboratory in Japan.

StarLight is a service mark of the Board of Trustees of the University of Illinois at Chicago and the Board of Trustees of Northwestern University.

8 References

[1] W. Allcock, J. Bester, J. Bresnahan, et al., Data Management and Transfer in High-Performance Computational Grid Environments. Parallel Computing, 2001.

[2] B. St. Arnaud, R. Hatem, W. Hong, M. Blanchet, F. Parent, Optical BGP Networks, http://www.canet3.net.

[3] D. D. Clark, M. L. Lambert, L., Zhang, NETBLT: A High Throughput Transport Protocol : *ACM* pp. 353-359, 1988.

[4] http://dast.nlanr.net/Projects/Iperf/

[5] J. Leigh, O. Yu, D. Schonfeld, R. Ansari, et al., “Adaptive Networking for Tele-Immersion,” in Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE), May 16-18, Stuttgart, Germany, 2001.

[6] J. Leigh., A. Johnson, T. A., DeFanti, Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments,. In the proceedings of Supercomputing '97 San Jose, California, Nov 15-21, 1997.

[7] http://netperf.org/netperf/NetperfPage.html

[8]www.evl.uic.edu/activity/template_act_project.php3?indi=147

[9] K. Park, Y. Cho, N. Krishnaprasad, C. Scharver, M. Lewis, J. Leigh, A. Johnson, “CAVERNsoft G2: A Toolkit for High Performance Tele-Immersive Collaboration,” Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000, October 22-25, 2000, Seoul, Korea, pp. 8-15

[10] www.evl.uic.edu/cavern/teranode/quanta

[11] www.startap.net/starlight

[12] W. R. Stevens, "TCP/IP Illustrated," vol. 1: Addison Wesley, 1994, pp. 344-350.

[13] W. R. Stevens, “Unix Networking Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI,” Addison Wesley, 1998, pp.357.

[14] www.web100.org

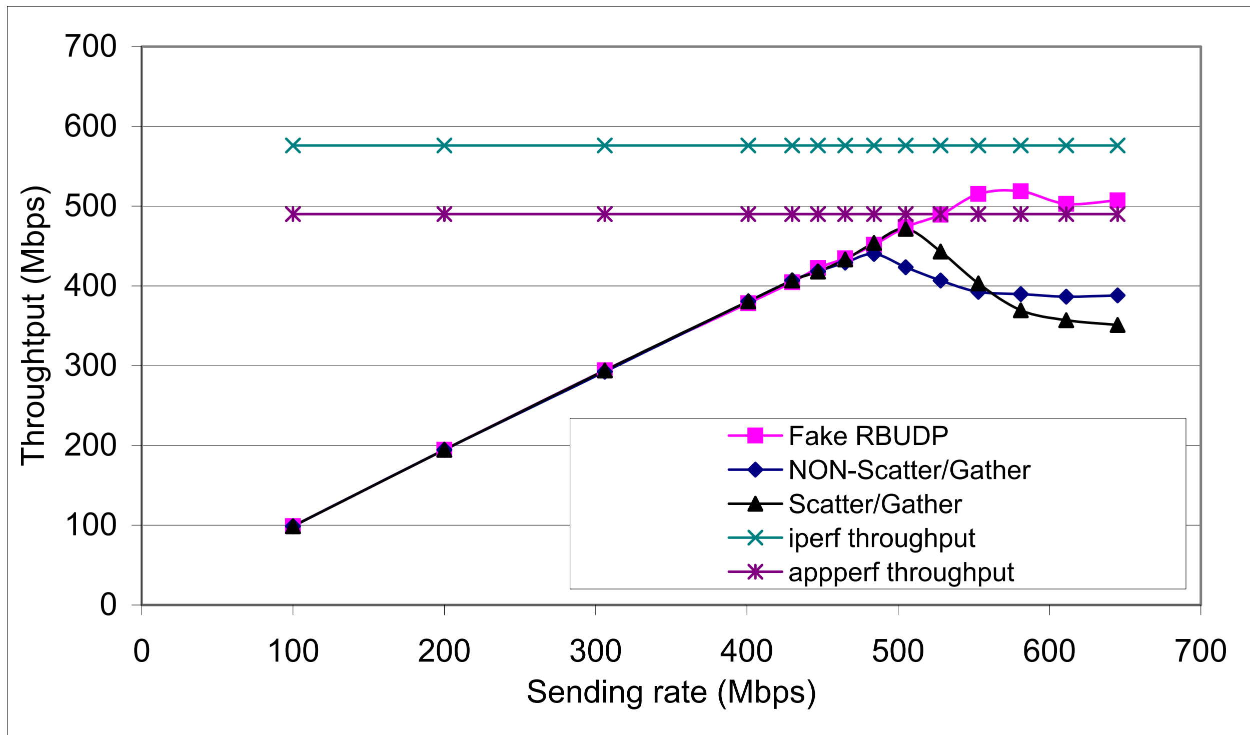

Figure 2. RBUDP throughput from Amsterdam to Chicago. Payload is 600MB. Bottleneck is in the receiving host.

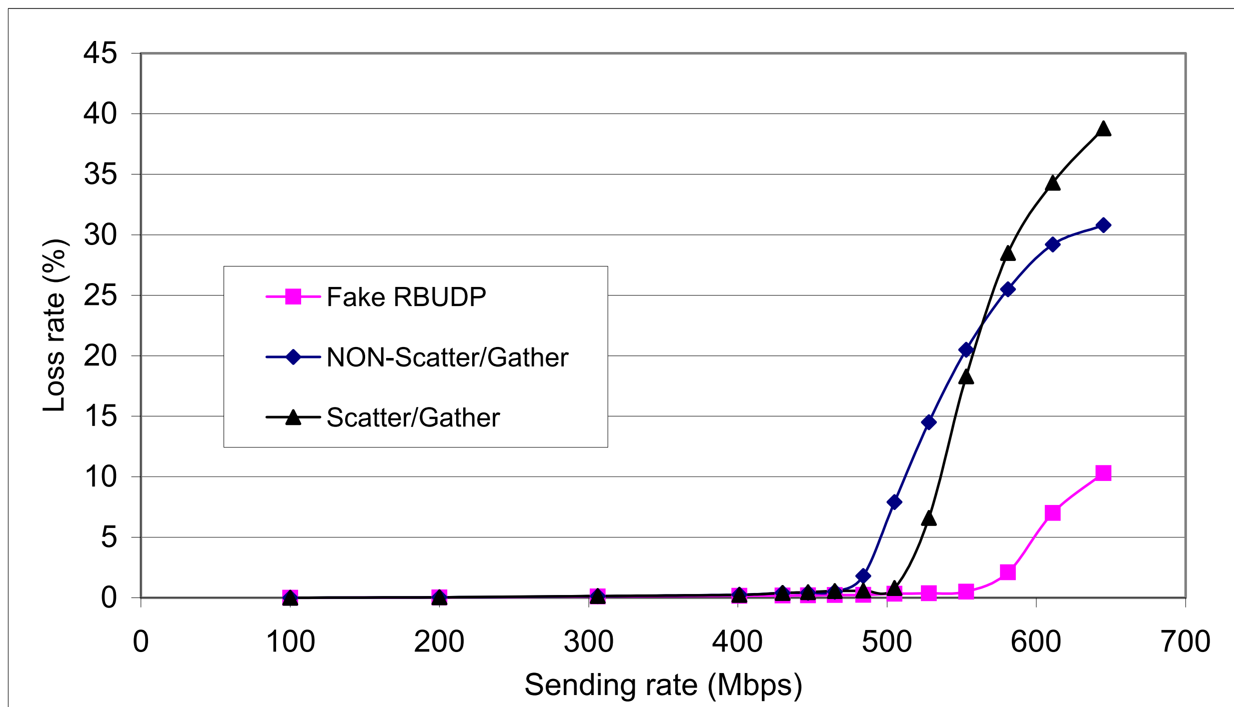

Figure 3. Loss rate of the first UDP blast from Amsterdam to Chicago.

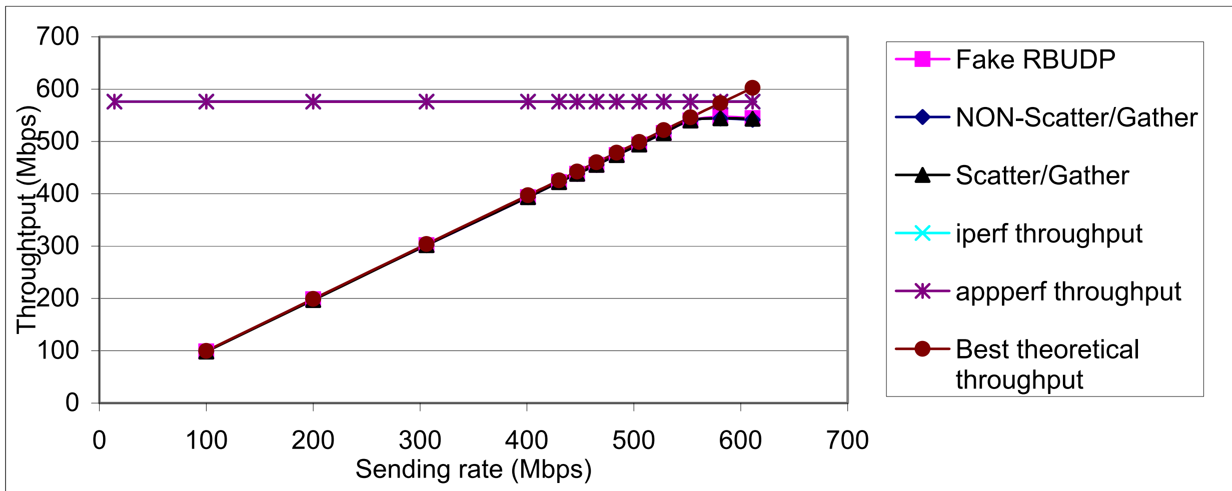

Figure 4. RBUDP throughput from Chicago to Amsterdam . Payload is 600MB. Bottleneck is in the network.

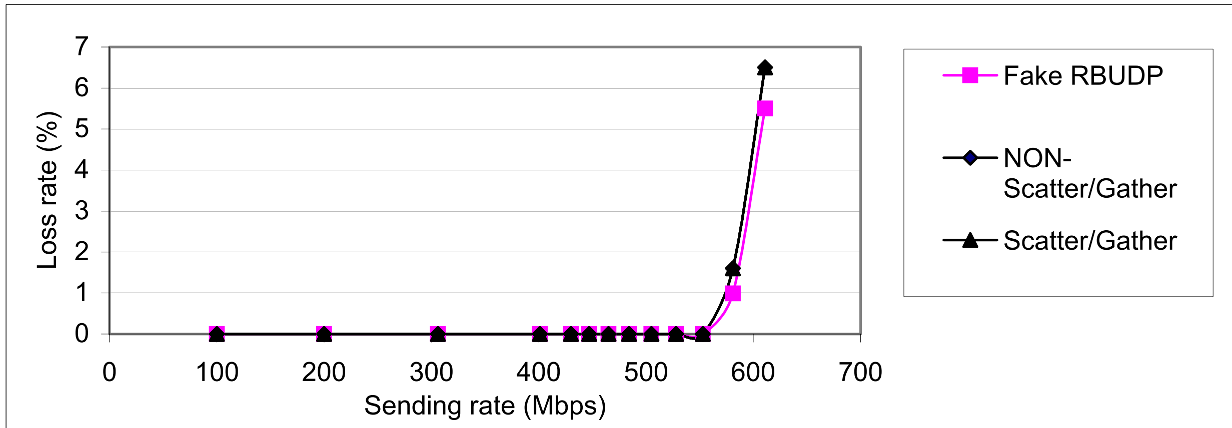

Figure 5. Loss rate of the first UDP blast from Chicago to Amsterdam.

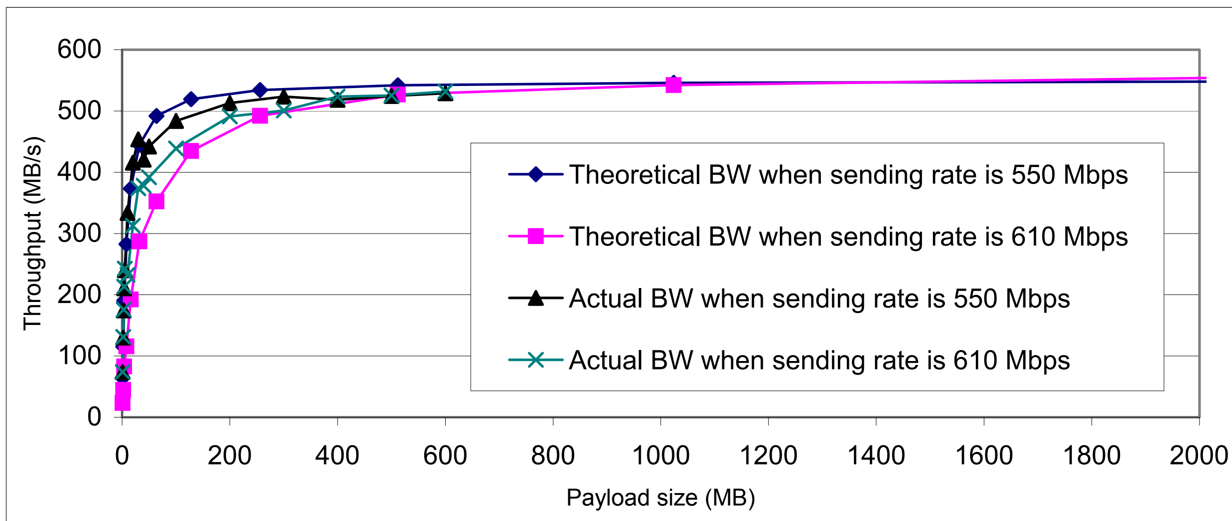

Figure 6. Throughput vs. Payload Size

E. He, J. Leigh, O. Yu, T. A. DeFanti, *Reliable Blast UDP : Predictable High Performance Bulk Data Transfer*, IEEE Cluster Computing 2002, Chicago, Illinois, Sept, 2002.