Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

ELSEVIER

Future Generation Computer Systems 1005 (2003) 1–15

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

Quanta: a toolkit for high performance data delivery over photonic networks

Eric He, Javid Alimohideen, Josh Eliason, Naveen K. Krishnaprasad, Jason Leigh*, Oliver Yu, Thomas A. DeFanti

Electronic Visualization Laboratory, University of Illinois at Chicago, MC152, 1120 SEO, 851 S. Morgan St., Chicago, IL 60620, USA

Abstract

Quanta is a cross-platform adaptive networking toolkit for supporting the data delivery requirements of interactive and bandwidth intensive applications, such as Amplified Collaboration Environments. One of the unique goals of Quanta is to provide applications with the ability to provision optical pathways (commonly referred to as Lambdas) in dedicated photonic networks. This paper will introduce Quanta's architecture and capabilities, with particular attention given to its aggressive and predictable high performance data transport scheme called Reliable Blast UDP (RBUDP). We provide an analytical model to predict RBUDP's performance and compare the results of our model against experimental results performed over a high speed wide-area network.

Keywords: Quanta; Photonic network; Reliable Blast UDP

1. Introduction

Amplified Collaboration Environments (ACE) are physical meeting spaces that enable distantly located groups to work in intensive collaboration campaigns that are augmented by advanced collaboration, computation, and visualization systems. One example of an ACE is the *Continuum* (Fig. 1) at the Electronic Visualization Laboratory [4], at the University of Illinois at Chicago, and at the Technology Research, Education and Commercialization Center in DuPage County, Illinois [25]. ACEs are based on the concept of the "War Room" or "Project Room" which have been shown to increase the productivity of collocated working teams by a factor of 2 [29]. The goal of the Continuum is to provide the same, if not greater, benefits for distributed teams. To this end, the Continuum integrates a broad range of technologies that include: multi-party video conferencing (via the AccessGrid [1]), electronic touch screens (for intuitive shared white-boarding), passive stereoscopic displays (such as the GeoWall, for displaying data sets in true 3D [13]), high resolution tiled displays (for displaying large visualizations or mosaics of visualizations), and PDAs and laptops for wireless control of these systems. Taken as a whole, each of these systems requires one or more computers to support. Hence a full Continuum will require a compute cluster per site. Furthermore this compute cluster must also be connected to other possibly distributed computing clusters, which might house massive data sets that are being shared in the collaborative environment. At EVL, we have developed a computing paradigm called the *Optiputer*

* Corresponding author.
E-mail addresses: spiff@evl.uic.edu, cavern@evl.uic.edu (J. Leigh).
URL: http://www.evl.uic.edu/cavern.

0167-739X/03/$ – see front matter © 2003 Published by Elsevier Science B.V.
doi:10.1016/S0167-739X(03)00071-2

Fig. 1. The continuum—an amplified collaboration environment.

as the primary means for supporting future generation networked applications such as the Continuum. Quanta is the networking middleware for supporting applications modeled after the Optiputer.

The Optiputer [18] is a National Science Foundation funded project to interconnect distributed storage, computing and visualization resources using photonic networks. The main goal of the project is to exploit the trend that network capacity is increasing at a rate far exceeding processor speed, while at the same time plummeting in cost. This allows one to experiment with a new paradigm in distributed computing—where the photonic networks serve as the computer's system bus and compute clusters taken as a whole, serve as the peripherals in a potentially, planetary-scale computer. We differentiate photonic networks from optical networks as networks comprised of optical fibers and MEMS optical switching devices. There is no translation of the photons to electrons and hence no routing within photonic switches. Applications that control these networks will direct photons directly from the start point to the end point of a series of photonic switches and hence will have full control of the available bandwidth in these allocated light paths.

In order to optimize data delivery in Optiputer applications such as ACEs, advances need to be made at several of the OSI network layers. At the physical layer, shared packet-switched Internet should be replaced by exclusive photonic switched networks to guarantee high bandwidth. We are currently developing the Photonic Interdomain Negotiator (PIN) to support this capability. At the data link layer, Multiple Protocol Label Switching (MPLS) or Virtual LAN (VLAN) replaces the slow and inefficient layer 3 switching, while at the same time providing Quality-of-Service. The Internet Protocol (IP) is still used at layer 3 in order to maintain compatibility with the Internet. At the transport layer, there is already consensus among network researchers that the current TCP implementations are not suitable for long distance high performance data transfer. Either TCP needs to be modified radically or new transport protocols should be introduced.

We intend to address the data transport problem with Quanta, a cross-platform adaptive networking toolkit for supporting the diverse networking requirements of interactive and data intensive Optiputer applications. The goal is to provide an easy to use system that will allow programmers to specify the data transfer characteristics of their application at a high level, and let Quanta transparently translate these requirements into appropriate networking decisions. The decisions will include making necessary QoS reservations, and adaptively utilizing the transport protocols to fulfill the user's data transfer requirements.

Quanta currently uses an optical network (Starlight) and a photonic network (Omninet) as experimental testbeds. Starlight [24] 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. Plans are underway to convert Starlight to a photonic network. OMNInet is a project supported by Nortel Networks, SBC Communications Inc. and Ameritech to assess and validate next-generation photonic technologies, architectures and applications in metropolitan area networks [17].

This paper begins with a description of Quanta's architecture and capabilities, with particular attention given to the development of high performance data transport schemes. We describe an algorithm for an aggressive bulk data transfer scheme called Reliable Blast UDP (RBUDP), provide an analytical model to predict its performance, and compare the results of our model against our implementation of RBUDP. Finally we extend the analytical model to support high throughput reliable data streaming, and compare it with the graphics streaming experiments performed during the IGrid 2002 conference in Amsterdam, The Netherlands.

2. Related work

In 1995, Gilder [7] predicted that network bandwidth would triple every year for the next 25 years. So far his prediction seems to be approximately correct. Each fiber optical wavelength channel can run at 10 Gbps. Wavelength division multiplexing gives 128 or more channels per fiber, resulting in a combined bandwidth of 1 terabits per second (almost 20,000,000 times faster than 56 Kbps modem connection). Consequently, this has led to a situation where straightforward use of the BSD socket library cannot take advantage of the high bandwidth available, making commonly used networking protocols unsuitable for high-end applications. 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) [26].

LFNs such as those between the US and Europe or Asia have extremely high round-trip latencies (at best 120 ms). This latency results in gross bandwidth under utilization when TCP is used for data delivery. This is because TCP's windowing mechanism imposes a limit on the amount of data it will send before it waits for an acknowledgement. International networks have long delays causing TCP to spend an inordinate amount of time waiting for acknowledgments. Consequently, 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* $\times$ *delay* product (or capacity) of the network. For example, for a 1 Gbps connection between Chicago and Amsterdam, with an average Round Trip Time (RTT) of 110 ms, the capacity is $1024 \times 0.11/8 = 14.1$ MB. 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. The WEB100 Consortium [28], which takes this approach, is developing techniques to modify router operating systems to report available bandwidth over a network link. They are also modifying operating systems kernels to allow better monitoring of TCP performance. Another solution is to use parallel TCP. In parallel TCP, the payload being delivered is divided into N partitions, which are delivered over N TCP connections. Leigh et al. [12,19] and Allcock et al. [2] have shown that parallel TCP can provide throughput as high as 80% of a network's available bandwidth, but its performance is unstable when excessive numbers of sockets are used. Moreover, it is difficult to predict the correct number of sockets to use. However, there is a growing community of high bandwidth network users that are realizing that there is no need for a congestion control mechanism if the application is able to reserve a dedicated network path such as in the case of photonic networks. As a consequence, there is now great interest in developing UDP-based protocols to improve bandwidth use. Simple Available Bandwidth Utilization Library (SABUL) [8] and Tsunami [27] are two recent examples. Our Reliable Blast UDP (RBUDP) protocol which we developed in 2000 is another [12]. The unique contribution of RBUDP is that we are able to provide an analytical model to predict its performance. This kind of predictability is important for data intensive, interactive applications.

3. Overall design of quanta

Quanta emerged from almost a decade's experience in connecting immersive CAVE systems [3] to each other and to supercomputers—this concept is called Tele-Immersion [14]. Quanta's predecessor is CAVERNsoft [19], which has been widely used by the CAVE community to develop advanced tele-immersive applications. Consequently, Quanta inherits all of the data sharing abstractions that have been found to be useful for developing these applications; and networked applications in general. Quanta aims to provide an Adaptive Network Controller (ANC) (Fig. 2) and three supporting services: a Resource Monitor, a Quality-of-Service provisioner and a collection of data transport mechanisms and data sharing abstractions. The ANC's first role is to take application-specified data delivery requirements (e.g. bandwidth, latency, jitter, reliability, etc) and translate them into networking and computational resource allocations needed to meet the applications' demands. The ANC will monitor the current state of the network or QoS capability, select an optimal transmission protocol, and make QoS requests (if available.) If QoS is available, the ANC contacts the Admission Control system to determine if the desired bandwidth is available, and then makes a reservation using the Signaling Controller. Once the strategy has been activated, the ANC will monitor the progress of the data transmission and adjust networking and computational parameters to sustain the desired performance. To accommodate multiple simultaneous and heterogeneous network flows, the ANC may alter some of the low-level transport protocol parameters (such as buffer size) or may adjust QoS reservations dynamically. The Signaling Controller maintains device-independence via a plug-in architecture that dynamically loads-in service-provider-specific libraries to signal for QoS. In the case of our photonic testbeds, the signaling controller interacts with PIN to make light path reservations.

3.1. Photonic interdomain negotiator: interdomain light path provisioning for quanta

Work is currently underway to develop a software infrastructure for light path provisioning on photonic networks (Fig. 3). While Quanta can ensure that data is optimally delivered over these light paths, it presently does not have the ability to allocate these dedicated light paths. This is the role of PIN. An application wishing to allocate a light path between two

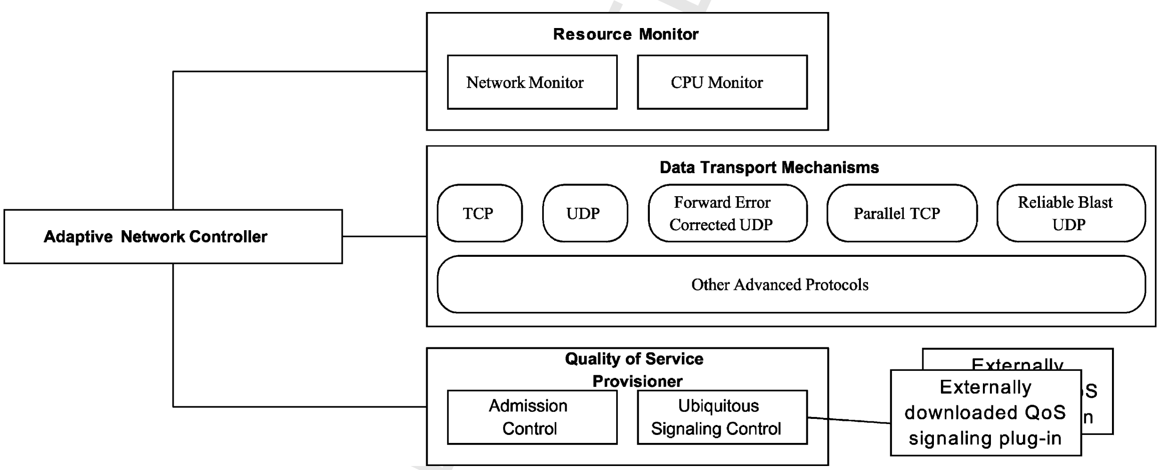


Fig. 2. Quanta's adaptive networking system.

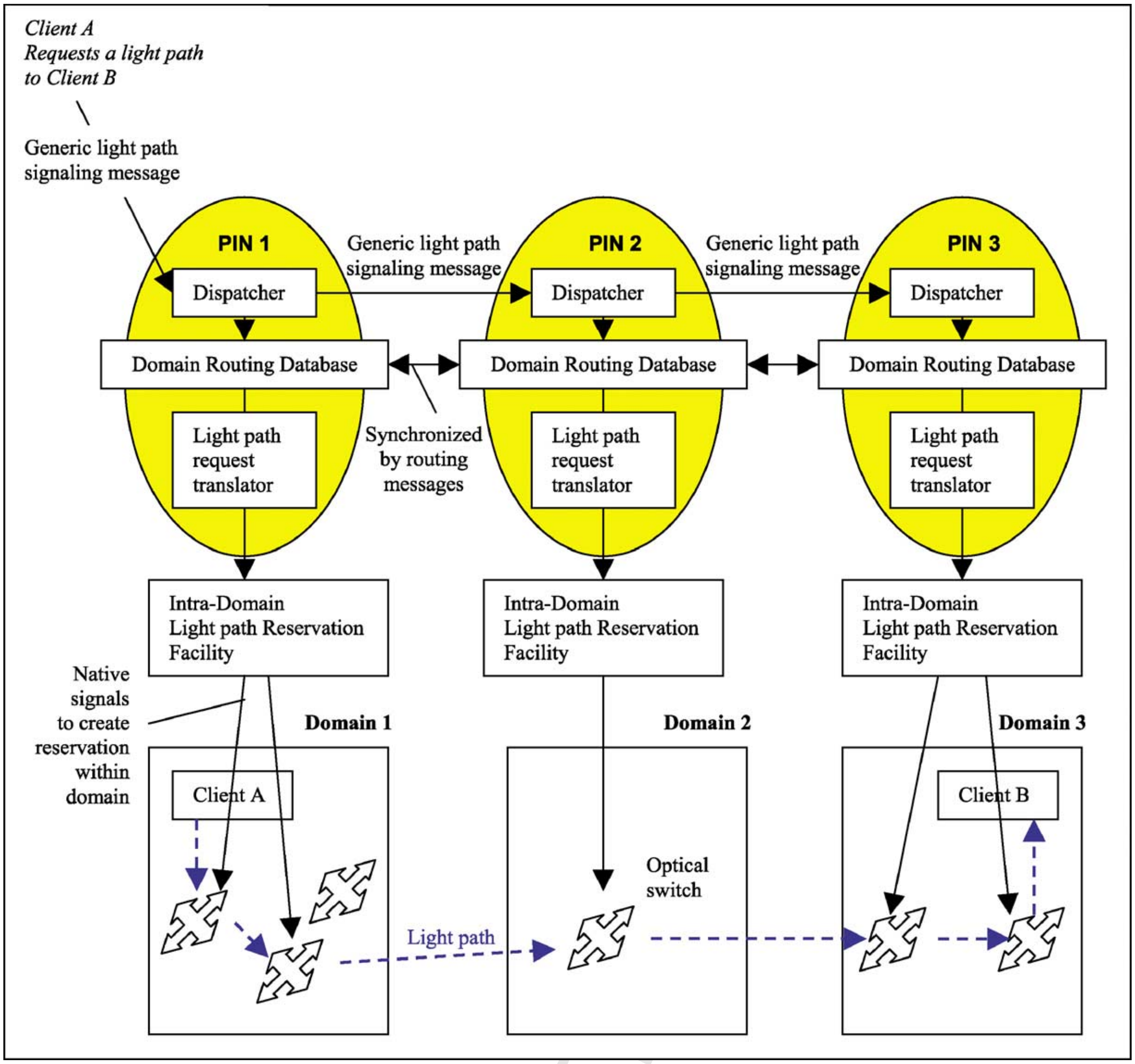


Fig. 3. Architecture of the interdomain light path provisioner.

end points, contacts its local PIN which will dispatch generic light path signaling messages to neighboring PINs until the final destination is reached. Each PIN will translate the generic light path signaling message into the native photonic signaling message that is understood by the local intradomain light path signaling facility. This facility then signals the photonic switch to make adjustments to its internal MEMS switches to establish the connection. At the present time a prototype of PIN is being developed, and TL1 command sets and APIs from multiple vendors such as Nortel, Glimmerglass, Calient and IMMI, are being examined to identify common commands that PIN will need to support.

3.2. Quanta's data transport and data sharing capabilities

Quanta's data transport capabilities include C++ classes that simplify socket-level programming of TCP, UDP and multicast communications (these are encapsulated in the C++ classes: QUANTAnet_tcp_c, QUANTAnet_udp_c, QUANTAnet_mcast_c, respectively). The reader is encouraged to examine the Quanta API manual [20] for a detailed explanation of how one goes about using the individual C++ classes. The names of the C++ classes are provided here as a reference. All the data transport classes have performance monitoring built into them so that an application can easily determine how much bandwidth it is using and how much latency it is experiencing. As Quanta is a cross-platform toolkit, it provides a data packing API that allows applications to ensure that their transmissions are correctly translated into the format of the target computer system (QUANTAnet_datapack_c). Quanta also provides a set of threading and mutual exclusion classes (QUANTAts_mutex_c, QUANTAts_thread_c, QUANTAts_condition_c).

Quanta provides a number of data sharing abstractions. These are described below:

- *QUANTAnet_tcpReflector_c and QUANTAnet_udpReflector_c*: Data reflection is a unicast method for emulating multicast. Clients send information to a central server rather than a single multicast address and the reflector repeats/reflects that same information to all other subscribing clients. From our experience we have found that this is one of the most heavily used capabilities for supporting data sharing in collaborative applications. The UDP reflector provides both unicast reflection and multicast bridging. This enables groups of clients to operate multicast within separated domains and share information across them using a bridge rather than having to set up a multicast tunnel, which often requires system administrator privileges. The TCP reflector is similar to the UDP reflector in that it places boundaries on TCP messages (making them discrete) instead of broadcasting them as a continuous stream.
- *QUANTAdb_c, QUANTAmisc_observer_c and QUANTAmisc_subject_c*: Quanta provides persistent distributed shared memory emulation via the QUANTAdb (or database) class. This is essentially a client/server database with automatic data reflection. Hence any updates to the database are propagated to all subscribers of the database. Clients are notified either via a traditional callback function or via a subject/observer mechanism [6]. This is essentially an object-oriented replacement for callbacks. The subject maintains a list of its observers for specific events and each observer will be triggered whenever the specific event occurs. The database assumes a Unix-like directory hierarchy with the leaf nodes containing the individual data values. These data values are intended to be small to expedite state information sharing rather than bulk data sharing.
- *QUANTAnet_rpc_c*: To complement Quanta's distributed shared memory and message passing capabilities, remote procedure calling is also provided. This allows clients and servers to invoke each other's functions and procedures. This is a widely used technique for distributed computing.
- *QUANTAnet_http_c*: This is a C++ class to access WEB servers.
- *QUANTAnet_parallelTcp_c*: This class works like Quanta's regular TCP socket class except a data buffer is partitioned and transmitted over several sockets rather than just one. Parallel TCP has been shown to be able to overcome the LFN problem, however the performance becomes unstable when too many parallel sockets are used [19].
- *QUANTAnet_remoteFileIO32_c, QUANTAnet_remoteFileIO64_c, QUANTAnet_remoteParallelFileIO32_c and QUANTAnet_remoteParallelFileIO64_c classes*: The Remote File I/O classes provide the capability for uploading and downloading files from a remote server. The provision of both 32 and 64 bit versions as well as parallel socket versions of the class allows for the efficient delivery of all file sizes, including those larger than 2 GB. The 64 bit version effectively allows one to deliver Terabyte files.
- *QUANTAnet_fecClient_c and QUANTAnet_fecServer_c*: For long distance networks such as international networks, latencies are high (on the order of hundreds of milliseconds). In advanced collaborative applications, we would ideally like state updates in the shared environment to occur with a minimum amount of latency and with a high degree of reliability. Data should be transmitted reliably over long distances without the acknowledgement typically used in protocols such as TCP. We have applied Forward Error Correction (FEC) to achieve this [5].

FEC collects between 1 and N (typically 2 or 3) data packets and performs a bit-wise operation on the packets (such as XOR), to produce a “redundant” packet. This packet is delivered along with the regular UDP traffic as a separate UDP stream. If any data packets are lost, FEC packets can be used to reconstruct the missing packet. By using such a scheme the latency and jitter can be reduced for reliable transmission over long distance networks.

QUANTAnet_rbudpSender_c and QUANTAnet_rbudpReceiver_c: When operating over dedicated networks the probability of packet loss is low. To take advantage of this opportunity one can use UDP augmented with acknowledgements. The Reliable Blast UDP (RBUDP) scheme works by “blasting” the contents of a data file at just below the available bandwidth without asking the remote site to acknowledge any of the packets [12]. Hence, all the available bandwidth is used for pure data transmission. At the remote site, a tally is kept for all the packets that have arrived and, after some timeout period, a list of missing packets is sent back to the sending client. The sender reacts by resending all the missing packets and again waiting for another negative acknowledgement, and so on. The next section focuses deeply into RBUDP, as it has recently gained significant importance as a technique in overcoming TCP’s inability to fill high bandwidth networks.

4. Reliable Blast UDP

Reliable Blast UDP [9] has two main 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. Fig. 4 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.

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 band-

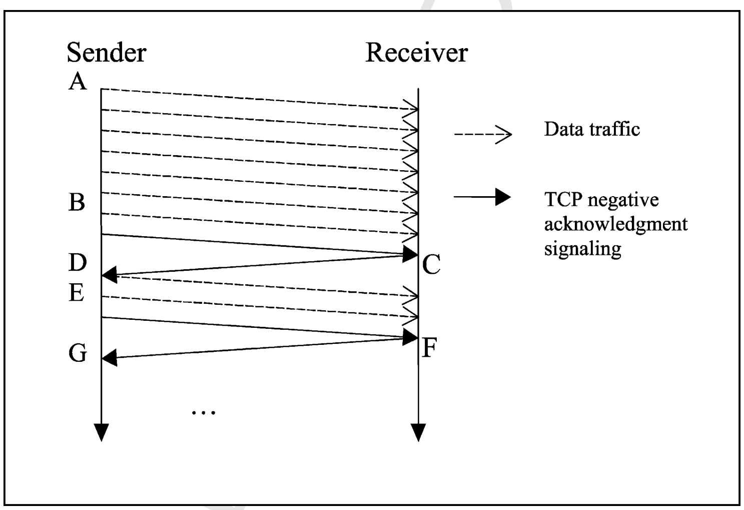


Fig. 4. Time sequence diagram of RBUDP.

width of the bottleneck link. Tools such as iperf [10] and netperf [16] 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 4.2.

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 4.2, we will include both iperf and app_perf's prediction of available bandwidth.

Three versions of RBUDP were developed:

- *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.
- *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. This optimization can improve the throughput by 10% when the receiving host is slower than the network. [9]
- *"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 next.

4.1. Analytical model for RBUDP

The purpose of developing an analytical model for RBUDP is twofold. 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.

Firstly, all variables are defined as follows: $B_{\text{achievable}}$ is the achievable bandwidth, B_{send} the chosen send rate, S_{total} the total data size to send (i.e., payload), T_{total} the total predicted send time, T_{prop} the propagation delay, T_{udpSend_i} the time to send UDP blast on ith iteration, N_{resend} the number of times to resend (depends on loss%), T_{ack} the time to acknowledge a blast (at least 1 ACK is always needed), L_i the % packet loss on ith iteration.

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

$$B_{\text{available}} = \frac{S_{\text{total}}}{T_{\text{total}}} \qquad (1)$$

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

$$\begin{aligned} T_{\text{total}} = {} & (T_{\text{prop}} + T_{\text{udpSend}_0}) \\ & + \left(\sum_{i=1}^{N_{\text{resend}}} (T_{\text{prop}} + T_{\text{udpSend}_i}) \right) \\ & + (N_{\text{resend}} + 1)(T_{\text{ack}} + T_{\text{prop}}) \end{aligned} \qquad (2)$$

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:

$$T_{udpSend_0} = \frac{S_{total}}{B_{send}}, \qquad T_{udpSend_i} = \frac{L_{i-1} S_{udpSend_{i-1}}}{B_{send}},$$

$$T_{ack} = \frac{S_{ack}}{B_{send}}, \qquad S_{ack} = \frac{S_{total}/S_{packet}}{8},$$

$$T_{ack} = \frac{S_{total}/8S_{packet}}{B_{send}}, \qquad S_{packet} = 1.5\,\text{KB}$$

Consequently:

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

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:

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

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

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

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

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

where $2T_{prop}$ is RTT.

This ratio shows that in order to maximize throughput, we should strive to minimize 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 50 ms, 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 67.5 MB.

In Section 4.2 (Fig. 5), we will use Eq. (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}) (Fig. 7).

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 is likely to 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 one packet. Therefore the number of retransmits required can be estimated as

$$N_{resend} = \left\lfloor \log L \left(\frac{S_{packet}}{S_{total}}\right)\right\rfloor \quad (6)$$

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 Eq. (3) to produce our new estimate of $B_{achievable}$ given constant loss rate L. In Fig. 7, we will put this prediction to use comparing an experimental situation where packet loss was observed.

4.2. Experimental results

The testbed network consisted of an OC-48 link (2.5 Gbps) 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 is shown in Table 1. Wgsara (in Amsterdam) was the slower PC, Charybdis (in Chicago) was the faster one. The network bottleneck resides in the Gigabit Ethernet cards of host computers.

In the first set of experiments, data was sent via RBUDP from the faster PC to the slower PC (from

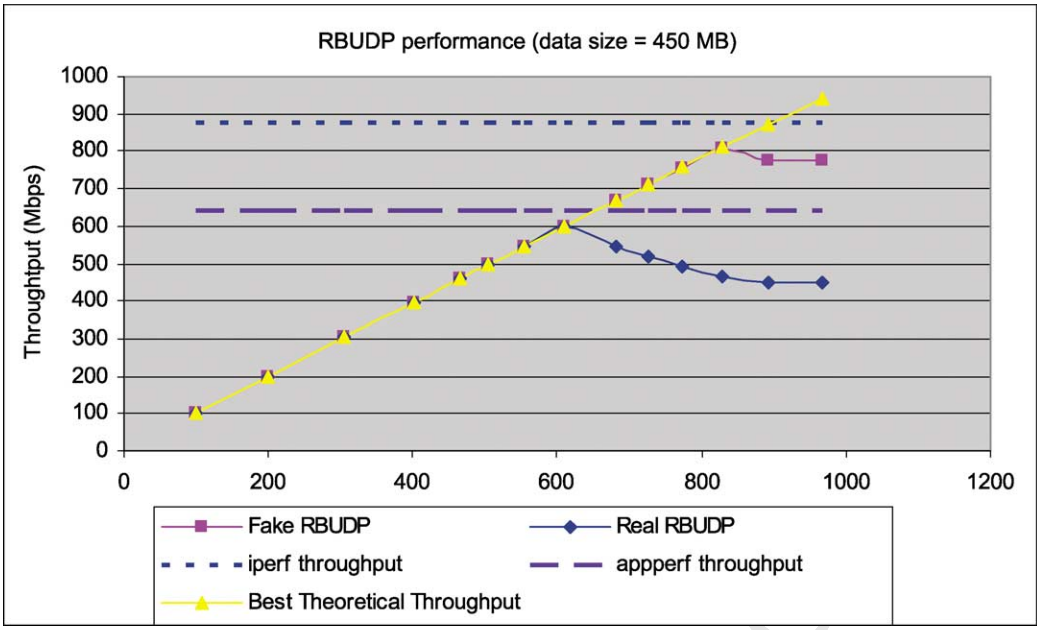


Fig. 5. RBUDP throughput from Chicago to Amsterdam. Payload is 450 MB. Bottleneck is in the receiving host. The lines indicating iperf and app_perf throughput show the maximum performance when the tools are sending at the network's full data rate. app_perf is a more realistic indication of the rate at which an application can absorb incoming data packets as it takes into account the additional overhead involved in most applications that need to take the data off the network and use it.

Chicago to Amsterdam). 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 result was compared against predicted results from our analytical model. A third set of experiments examined RBUDP throughput for different payload sizes.

4.2.1. From the fast PC to the slow PC (Chicago to Amsterdam)—when the bottleneck is in the receiving host computer

In this experiment, iperf measured maximum available bandwidth at 878 Mbps, and app_perf measured maximum possible throughput at 643 Mbps. In Fig. 5, we plot these thresholds as lines across the top of the graph. Plotting the achieved throughput at various sending rates for the fake and real 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 real RBUDP begins to hurt in performance because the underpowered CPU is unable to keep up with handling the incoming packets. However, as real RBUDP is able to match the maximum performance of app_perf, this means that RBUDP is making as much use of the network for useful data transfer as the CPU will allow. Finally, notice

Table 1
Specification of host PCs in the experimental testbed

Host name	CPU	Memory size	System bandwidth
Wgsara2.phys.uu.nl (Amsterdam)	Pentium III 800 MHz	512 MB	238 MB/s
Charybdis.sl.startap.net (Chicago)	XEON 1.8 GHz	512 MB	844 MB/s

that there is a close match between our experimental results and our prediction from Eq. (4) (which estimated RBUDP performance when loss rate is zero).

4.2.2. From the slow PC to the fast PC (Amsterdam to Chicago)—when the bottleneck is in the sending host computer

We repeated the experiment in the opposite direction. This time the bottleneck was in the sending PC rather than in the receiving PC. Fig. 6 shows that when the host computer is fast enough, iperf and app_perf performances match, as do the different implementations of RBUDP. Fake RBUDP is able to reach the maximum performance obtained by iperf; and Real RBUDP is able to reach the maximum performance obtained by app_perf-again confirming RBUDP's ability to maximize bandwidth utilization for useful data delivery.

4.2.3. Effect of payload size on throughput

From the analysis in Section 4.1, 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 Fig. 7, we compare an experimental situation where we send data at 611 Mbps (experiencing no loss) against our theoretical prediction, which assumes no loss (Eq. (3)). Furthermore we compare an experimental situation sending data at 682 Mbps experiencing 12% loss, against our theoretical prediction where we assume a constant 12% loss per iteration.

Firstly, the results show that RBUDP performs best for large payloads. Secondly, the results show that a 12% packet loss does not impact throughput greatly for large payloads. Finally, our analytical models provide good boundaries for our experimental results for 0 and 12% loss.

4.3. Adapting RBUDP for high speed data streaming

Even though the initial motivation of RBUDP is for bulk data transfer over long distance, some applications require high performance reliable streaming transport. In Section 4.2.3, we showed that in order to achieve fairly high throughput, the payload needs to be large. In streaming applications, if the size of

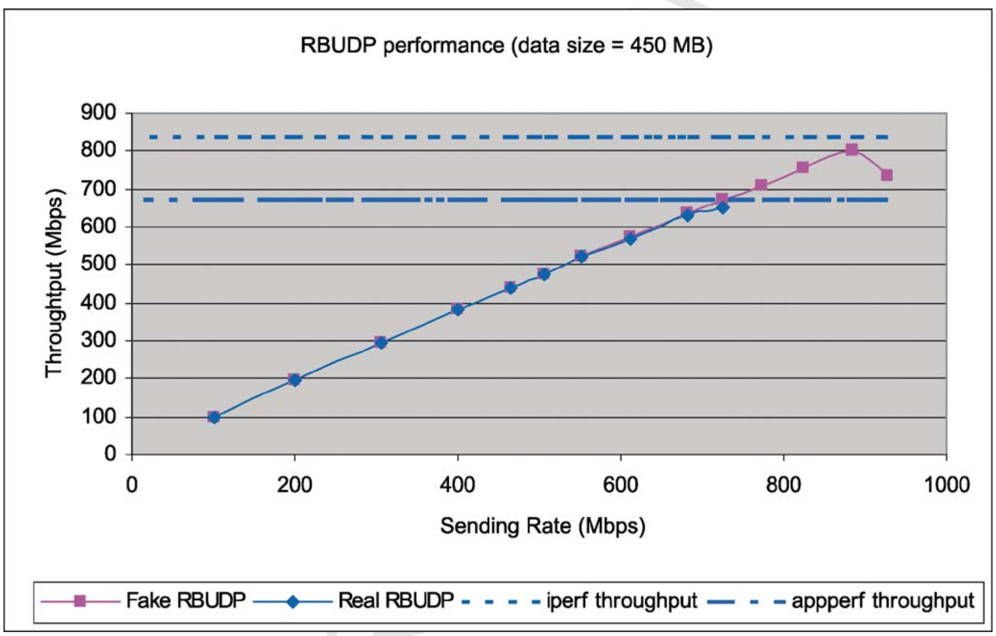


Fig. 6. RBUDP throughput from Amsterdam to Chicago. Payload is 450 MB. Bottleneck is in the sending host. The maximum of the sending rate is 725 Mbps. See Fig. 5 for an explanation of the iperf and app_perf lines in the graph.

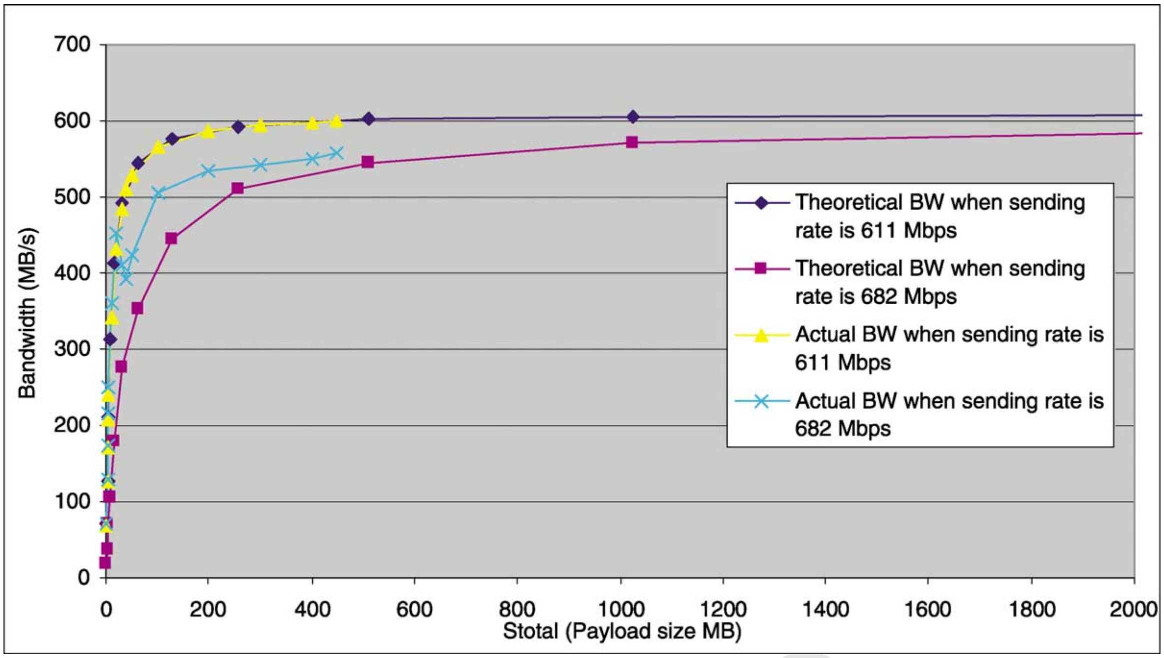


Fig. 7. Throughput vs. payload size. Larger payloads produce better network utilization.

objects to be streamed is small, we combine multiple objects to form a large payload. However this will cause end-to-end latency to increase because of the buffering needed to form the large payloads. Based on our analytical model, we can determine the minimum sending rate needed to ensure a desired object throughput rate, given the maximum delay the application is able to tolerate.

Let S_{obj} is the size of streamed objects, N_{obj} the number of objects per payload, B_{obj} the required throughput of objects (number of objects per second). For example, in the case of graphics streaming, object throughput rate is measured in frames per second, D is the maximum extra delay the application can tolerate.

Then the size of a payload is

$$S_{\text{total}} = S_{\text{obj}} N_{\text{obj}} \tag{8}$$

where

$$N_{\text{obj}} = B_{\text{obj}} D \tag{9}$$

The required raw bandwidth is

$$B_{\text{best}} = B_{\text{obj}} S_{\text{obj}} \tag{10}$$

Assuming we are operating over an over-provisioned network, we plug (8), (9) and (10) back in Eq. (5) to compute the rate at which RBUDP needs to send data to achieve the application's requested throughput:

$$B_{\text{Send}} = \frac{S_{\text{obj}} B_{\text{obj}}}{1 - \text{RTT}/D} \tag{11}$$

Hence, using a graphics streaming application as an example: given that RTT is 100 ms, S_{obj} is $800 \times 600 \times 3$ (assuming image resolution of 800×600 and 3 bytes color information for each pixel) if we want to achieve a frame rate B_{obj} of 20 frames/s, the maximum extra delay introduced will be 0.5 s, the sending rate needs to be at least 288 Mbps and each payload must encapsulate 10 image frames. During IGrid 2002, Luc Renambot applied Quanta's RBUDP to a parallel graphics streaming application called *Griz*. Using our analytical model and the parameters from the above example, we were able to predict the number of animation frames that Griz had to package into a single payload to achieve full utilization of the Amsterdam–Chicago Starlight link [21].

5. Conclusions

We have described the overall architecture and capabilities of Quanta, a cross-platform C++ toolkit for building high performance networking applications. In particular we described in detail, an aggressive bulk data transfer scheme, called RBUDP, which is intended for either dedicated, or QoS-enabled high bandwidth networks. RBUDP 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. This is because with smaller payloads, the time taken for completing the delivery approaches the time taken to acknowledge the payload.

We have provided an analytical model that gives a good prediction of RBUDP's performance. This prediction can be used as a rule of thumb in a manner similar to the *bandwidth* $\times$ *delay* product for TCP. In addition, this prediction can be used to estimate how future ideas for improving the algorithm might impact RBUDP performance. Even though the initial application of RBUDP is bulk data transfer over high-speed networks, this protocol can also be extended for use in streaming applications. Here an application must make a tradeoff between latency and throughput. To achieve higher throughput, latency will increase because more data must be aggregated as a single transmission payload.

Through the combined use of PIN and Quanta, bandwidth intensive Optiputer applications will soon be able to allocate light paths between multiple photonic domains and make full use of the available bandwidth.

Uncited references

[11,15,22,23].

Acknowledgements

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-0115809, ANI-9980480, ANI-0229642, ANI-9730202, ANI-0123399, ANI-0129527 and EAR-0218918, as well as the NSF Information Technology Research (ITR) cooperative agreement (ANI-0225642) to the University of California San Diego (UCSD) for "The OptIPuter" and 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) 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. The CAVE and ImmersaDesk are registered trademarks of the Board of Trustees of the University of Illinois. PARIS, Wanda, CAVELib and ElsieDesk are trademarks of the Board of Trustees of the University of Illinois. STAR TAP and Euro-Link are service marks of the Board of Trustees of the University of Illinois. 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.

References

[1] http://www-fp.mcs.anl.gov/fl/accessgrid/.

[2] W. Allcock, J. Bester, J. Bresnahan, et al., Data management and transfer in high-performance computational grid environments, Parallel Comput. (2001).

[3] C. Cruz-Neira, D. Sandin, T.A. DeFanti, Virtual reality: the design and implementation of the CAVE, in: Proceedings of the SIGGRAPH'93 Computer Graphics Conference, August 1993, ACM SIGGRAPH, pp. 135–142.

[4] http://www.evl.uic.edu/cavern/continuum/indexmain.html.

[5] R. Fang, D. Schonfeld, R. Ansari, J. Leigh, Forward Error Correction for Multimedia and Tele-immersion Streams, 2000. http://www.startap.net/images/PDF/RayFangFEC1999.pdf.

[6] E. Gamma, E. Helm, R. Johnson, J. Vlissides, Design Patterns—Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995, pp. 293–303.

[7] G. Gilder, Fiber Keeps Its Promise: Get Ready, Bandwidth will Triple Each Year for the Next 25, Forbes, April 7, 1997.

[8] S. Harinath, Data management support for distributed data mining of large datasets over high speed wide area networks, PhD Thesis, University of Illinois at Chicago, 2002.

[9] E. He, J. Leigh, O. Yu, T. DeFanti, Reliable blast UDP: predictable high performance bulk data transfer, in: Proceedings of the IEEE Cluster Computing 2002, Chicago, IL, September 2002.

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

[11] A. Johnson, M. Roussos, J. Leigh, C. Barnes, C. Vasilakis, T. Moher, The NICE Project: learning together in a virtual world, in: Proceedings of the IEEE VRAIS'98, Atlanta, GA, March 14–18, 1998, pp. 176–183.

[12] J. Leigh, O. Yu, D. Schonfeld, R. Ansari, et al., Adaptive networking for tele-immersion, in: Proceedings of the Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE), May 16–18, Stuttgart, Germany, 2001.

[13] J. Leigh, G. Dawe, J. Talandis, E. He, S. Venkataraman, J. Ge, D. Sandin, T.A. DeFanti, AGAVE: access grid augmented virtual environment, in: Proceedings of the AccessGrid Retreat, Argonne, IL, January 2001.

[14] J. Leigh, T.A. DeFanti, A. Johnson, M. Brown, D. Sandin, Global tele-immersion: better than being there, in: Proceedings of the Seventh International Conference on Artificial Reality and Tele-Existence, Tokyo, Japan, December 1997, pp. 10–17.

[15] 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: Proceedings of the Supercomputing'97, San Jose, CA, November 15–21, 1997.

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

[17] http://www.evl.uic.edu/activity/template_act_project.php3?indi=147.

[18] http://www.evl.uic.edu/cavern/optiputer.

[19] K. Park, Y. Cho, N. Krishnaprasad, C. Scharver, M. Lewis, J. Leigh, A. Johnson, CAVERNsoft G2: a toolkit for high performance tele-immersive collaboration, in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000, Seoul, South Korea, October 22–25, 2000, pp. 8–15.

[20] http://www.evl.uic.edu/cavern/quanta/documentation.html.

[21] L. Renambot, T.V.D. Schaaf, H.E. Bal, D. Germans, H.J.W. Spoelder, Griz: experience with remote visualization over optical grids, in: Proceedings of the IGrid 2002, October 2002.

[22] D.C. Schmidt, S.D. Huston, C++ Network Programming: Mastering Complexity Using ACE and Patterns, Addison-Wesley, Reading, MA, 2001.

[23] R. Singh, J. Leigh, T.A. DeFanti, F. Karayannis, TeraVision: a high speed, high resolution graphics streaming device for the OptIPuter, in: Proceedings of the IGrid 2002, October 2002.

[24] http://www.startap.net/starlight.

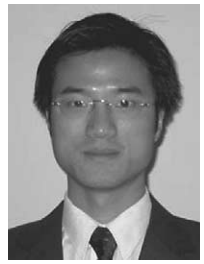
[25] http://www.trecc.org.

[26] W.R. Stevens, TCP/IP Illustrated, vol. 1, Addison-Wesley, Reading, MA, 1994, pp. 344–350.

[27] http://www.indiana.edu/~anml/anmlresearch.html.

[28] http://www.web100.org.

[29] J.S. Olson, L. Covi, E. Rocco, W.J. Miller, P. Allie, A room of your own: what would it take to help remote groups work as well as collocated groups? in: Proceedings of the Conference on Human Factors in Computing Systems (CHI'98), Short Paper, 1998, pp. 279–280.



Eric He received his BS, MS degrees in Optical Engineering from Beijing Institute of Technology in 1994 and 1997, respectively. He is currently working toward a PhD degree in Computer Science at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago. His research interests include High Performance Network Transport Protocols, Optical Networking, Quality-of-Service, Cluster Computing and Virtual Reality.

Javid Alimohideen is a Master's student at Electronic Visualization Laboratory, UIC. His current research work involves development and maintenance of the Quanta toolkit.

Josh Eliason is an MFA student and Research Assistant at Electronic Visualization Laboratory, University of Illinois at Chicago. Research interests include human computer interaction with augmented and virtual reality systems, augmented and virtual reality tracking systems-tracking and controller devices, and distributed heterogeneous networking-mobile computing, device discovery, services, and synchronization.

Naveen K. Krishnaprasad received his Bachelor of Engineering (BE) in Instrumentation and Control from the University of Madras, India and his Master of Science (MS) in Computer Science from the University of Illinois at Chicago. Now he is a PhD student at the Electronic Visualization Laboratory (EVL) at University of Illinois at Chicago (UIC). His research interests include distributed computing, large data visualization, and collaborative environments. Krishnaprasad worked at Christian Michelsen Research AS in Norway, on the design of a virtual reality kernel for tele-immersive applications. His prior work at EVL included development and maintenance of the CAVERNsoft/QUANTA networking library and design of a unified collaborative framework for network analysis.

Jason Leigh is an Associate Professor in the department of Computer Science at the University of Illinois at Chicago; and a Senior Scientist at the Electronic Visualization Laboratory (EVL). Leigh is a co-chair of the Global Grid Forum's Advanced Collaborative Environments research group; and a co-founder of the GeoWall Consortium.

Oliver Yu received his Master's and PhD degrees in Electrical and Computer Engineering from the University of British Columbia. He has over 12 years of industry experiences in computer networking and telecommunications. He held technical leadership positions in Microtel Pacific Research, Nortel and Hughes. He last worked in Motorola as the section manager for the GPRS wireless mobile network development. He is an Assistant Professor in the ECE department at the University of Illinois at Chicago. He has been working on research projects that are funded by the NSF, DOE and NTT Japan. His research interests are in the areas of intelligent optical network signaling, next generation Internet service provisioning, and 3G/4G wireless mobile networks.

Thomas A. DeFanti is the Director of the Electronic Visualization Laboratory (EVL), a distinguished professor in the department of Computer Science, and director of the Software Technologies Research Center at the University of Illinois at Chicago.