IQ-Services: Network-Aware Middleware for Interactive Large-Data Applications

Zhongtang Cai, Greg Eisenhauer, Qi He, Vibhore Kumar, Karsten Schwan, Matthew Wolf
{ztcai,eisen,qhe,vibhore,schwan,mwolf}@cc.gatech.edu
College of Computing
Georgia Institute of Technology

Abstract

IQ-Services are application-specific, resource-aware code modules executed by data transport middleware. They constitute a 'thin' layer between application components and the underlying computational and communication resources that implements the data manipulations necessary to permit wide-area collaborations to proceed smoothly, despite dynamic resource variations. IQ-Services interact with the application and resource layers via dynamic performance attributes, and end-to-end implementations of such attributes also permit clients to interact with data providers. Joint middleware/resource and provider/consumer interactions implement a cooperative approach to data management for the large-data applications targeted by our research. Experimental results in this paper demonstrate substantial performance improvements attained by coordinating network-level with service-level adaptations of the data being transported and by permitting end users to dynamically deploy and use application-specific services for manipulating data in ways suitable for their current needs.

1 Introduction

Motivation. Distributed applications that 'stress' wide area networks include telepresence[25], remote collaboration and visualization [45], remote instrument access and control[29, 13], and real-time monitoring and surveillance[43]. Problems arise both from their large data volumes and from their interactive nature, the latter requiring data to be transferred with low latency and high predictability. Factors contributing to these problems include the heterogeneity and dynamics of underlying networks, the lack of support for QoS at the network level, and TCP's end-to-end congestion control that results in bursty network traffic coupled with the delivery of unstable QoS over time [34]. In addition, it is difficult to measure available network bandwidth, especially when collaborators or remote users must utilize shared substrates like the Internet [49].

IQ-Services: resource-aware middleware. This paper describes and evaluates middleware-based methods of providing to interactive wide area applications the data they need at acceptable levels of quality. The idea of the IQ-Services resource-aware middleware is to have middleware execute application-defined services for data filtering, transformation, and scheduling, and to continuously adjust service behavior in accordance with current resource availability and client needs. This paper's specific focus is on network behavior, which is captured with dynamic bandwidth measurement [18, 26] techniques. Network-aware middleware services are installed at runtime and under application control, initiated by applications and on the machines used by them. An example is a data filtering service installed by a client on a machine acting as its data source, thereby giving the client complete control of the data sent to it. A concrete instance is a data downsampling filter used by a visualization client, where the filter continuously varies data resolution in order to maintain acceptable data transmission rates, despite variations in available network bandwidth [17, 15]. Other examples include data reordering, prioritization, and elimination [46, 15], the use of application-specific compression methods [48], and data transformations that implement tradeoffs in the amounts of processing vs. data volumes in the overlay networks middleware uses for service execution [4, 6].

The IQ-Services model is implemented with the IQ-ECho middleware, along with a service-aware communication protocol underlying such middleware. This protocol, termed IQ-RUDP [15], provides instrumentation less general than

[1]This work is supported in part by the NSF ANIR program and by the DOE MICS High Performance Networks program.

but akin to the instrumented Linux protocol stacks developed in the Web100 project [26]. In contrast to such work, however, IQ-RUDP uses dynamic performance attributes associated with communications to convey monitoring and control information across the middleware/protocol boundary. Per message monitoring information includes current RTT (Round Trip Time) and loss rate. Control information includes desired loss tolerance and packet markups with priorities. Additional resource information is available from system-level monitoring mechanisms described in [19].

The ability to 'drive' dynamic changes in middleware services with system- and protocol-level information is demonstrated with an interactive high performance application, termed SmartPointer [48]. This application permits remote users to collaboratively view and manipulate data produced by a molecular dynamics simulation [48]. Users can dynamically specialize data by using middleware-level transformation and filtering services. These services are dynamically adjusted in accordance with network-level changes in behavior.

Experimental evaluations are performed across wide-area network links and on the NetLab network emulation facility [22]. Additional measurements now underway involve the use of IQ-ECho to transport large-scale interactive data across a 10Gbps link connecting Georgia Tech with machines located at Oak Ridge National Laboratories. The goal is to enable wide-area collaborations like those required in DOE's Supernova Initiative, where multiple remote collaborators seek access to output data produced at rates approximating 1GB/sec for future large-scale simulation scenarios.

Contributions. The contributions of this paper are threefold. First, we experimentally validate earlier simulation- and emulation-based results [15] in which we show that coordinated application/middleware- and network-level adaptations can outperform application-only adaptation methods. Second, we experiment with the concurrent adaptation of multiple communication links, to adjust their joint behavior to underlying network capabilities. Third, we demonstrate the generality of the IQ-Services architecture by applying it to grid applications other than those written by our group, by creating and evaluating a network-aware version of GridFTP, termed IQ-GridFTP. IQ-GridFTP dynamically adjusts file contents during transfer (via user-supplied data manipulation services) in order to maintain transfer rates under varying network conditions. Adjustments may result in the partial file transfers specified as part of the GridFTP standard or they may be size-sensitive changes in file formats and data contents, as when a data provider chooses to share only the low-resolution data contained in a file with a partner.

We leverage substantial previous work on adaptive systems. In comparison to our own work with application-level actions like runtime service replication, relocation, or tuning to meet dynamic bandwidth constraints [17, 33], the IQ-Services software architecture focuses on 'lightweight' services interposed via middleware 'between' applications and underlying operating systems. The intent is to permit end users to write their application-level components in any way suitable for their domain, rather than dictating the use of certain application programming paradigms. IQ-Services simply implement the additional adaptive behaviors relevant to such application components. The role of middleware is to facilitate the execution of such behaviors. In contrast to previous CORBA-based middleware [51], however, the IQ-Services architecture and implementation are focused on the high performance/grid domain, and the adaptation methods employed specifically address this domain's need to manipulate and transfer large data volumes across wide area network infrastructures. Our work also borrows concepts and ideas from prior work on the adaptive transfer and manipulation of multimedia or sensor data, which has already demonstrated that in dynamic systems, application-level constraints like end-to-end delays can be met only if applications and/or underlying system-level mechanisms [27] dynamically adapt to runtime changes in resource availability [38, 40]. Finally, we note that industry is currently pursuing goals similar to those of our group, including in efforts like IBM's autonomic computing initiative [20].

Overview. The remainder of this paper is organized as follows. In Section 2, sample distributed applications that use IQ-Services are outlined, including a description of their performance requirements and of the IQ-Services associated with their use. In Section 3, the architecture of IQ-Services is described. The intent is to demonstrate the general nature of our approach, including its applicability to ongoing OGSA-based standards efforts for grid software. Experimental results attained on wide area networked machines and on the NetLab emulation facility appear in Section 4. Conclusions and future research are outlined in Section 5.

2 Wide-area Scientific Collaboration

There are multiple, ongoing, wide-area collaborations in high performance computing, ranging from high profile efforts like Ligo [23, 13] and DOE's Terascale Supernova Initiative (TSI) [7] to adhoc collaborations between specific sets of researchers. Collaboration tools like Access Grid [2] and grid portal efforts [31] are evidence of the importance of remote collaboration to the broader CS and scientific communities.

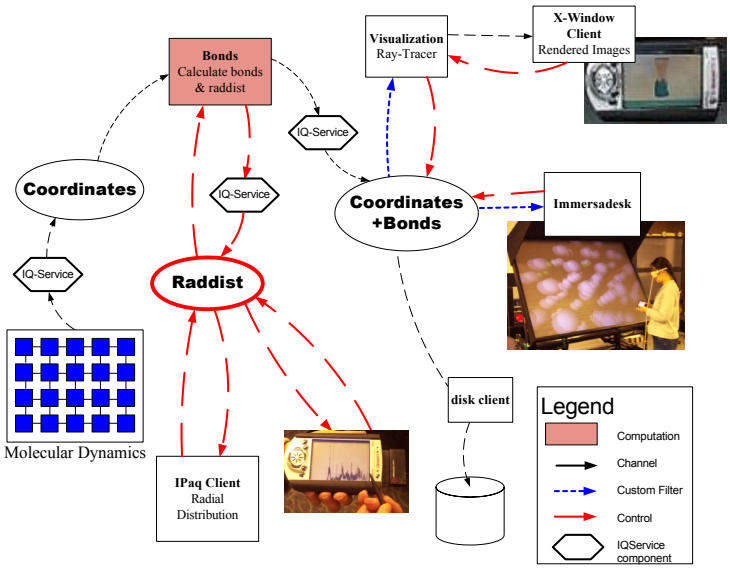

Figure 1: SmartPointer System Overview

This paper experiments with two tools commonly used in such collaborations: (1) real-time collaborations that include remote data visualization, and (2) mechanisms for large file transfers. In both cases:

- the underlying computing/network infrastructure is heterogeneous, linking high end machines with workstations and even portable devices, with networks ranging from high speed LANs to wide area to wireless connections; and
- the amounts of data produced, transported, and consumed are large, expected to be up to 100Gb/sec in applications like TSI, but in our current work, we use more moderate data volumes.

2.1 Real-time Collaboration

For real-time scientific collaboration, in addition to large data volumes shared across heterogeneous infrastructures, an important attribute of theirs is that 'fresh' or new information may be more relevant to a collaborator than complete detail. This implies that they share real-time requirements with applications like multimedia and video services [44, 46, 21, 16].

Figure 1 shows a prototypical, distributed collaborative visualization [1, 48]. It implements a many-to-many dataflow, and the heterogeneous underlying computing/communication platform ranges from low end, ill-connected clients to high end, well-connected server machines. Collaborative environments like these pose problems to developers both because of the heterogeneous platforms and because of inherent capabilities expected by end users. For example, data delivery should be coordinated for effective collaboration, if the information for one collaborator consistently lags that of the others, collaboration activities may be compromised. The resulting soft real-time constraints on data delivery imply a need for consistent frame rates (to insure data freshness), perhaps in preference over always receiving data at the highest level of resolution. Another issue is fairness across multiple connections, which is particularly important when large data transfers share network links with remote control loops. Sample loops in our application include those that control a remote data filter and those that implement dynamic annotations on clients' displays.

Our work addresses some of issues, including:

- information freshness [35],
- end-to-end information quality, and
- fairness, particularly across the multiple connections maintained in collaborative applications.

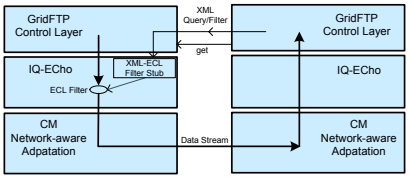

Figure 2: IQ-GridFTP Overview

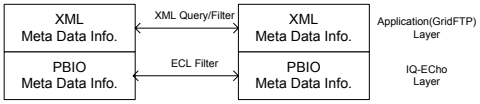

Figure 3: IQ-GridFTP MetaData Mapping

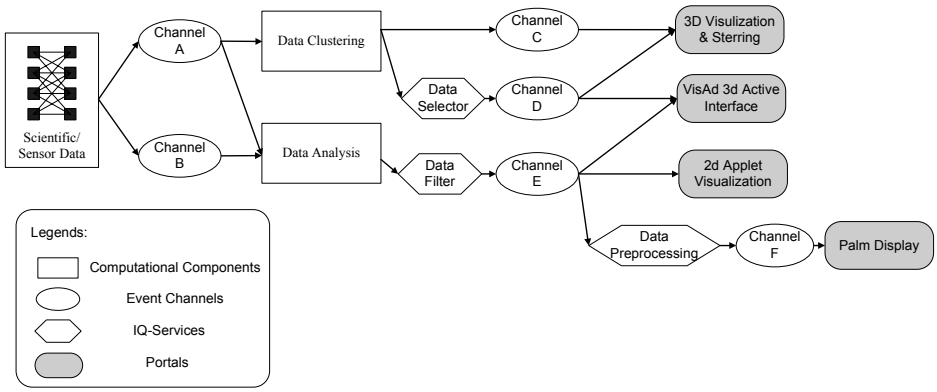

Figure 4: Typical Application Scenario

2.2 Large-Data Transfers via IQ-GridFTP

The second application evaluated in this paper is termed IQ-GridFTP. IQ-GridFTP extends the Globus GridFTP implementation [11] by realizing its goal of partial file transfers, that is, to transfer both entire files and also certain regions of files. The GridFTP standard represents this functionality with ERET (Extended Retrieve) and ESTO (Extended Store) file access instructions, which include an offset size parameter to fetch relevant file portions. In the future, we expect file manipulations and transfers to be able to utilize the DFDL (Data Format Description Language) currently under development. This language will provide the high-level descriptions of file contents needed for realizing finer grain content filtering, down to the level of individual file data attributes.

Our intent is to utilize partial file transfers to limit network usage from a server to specific clients, by deploying customized filters at the server-resident data source. Figure 2 demonstrates such functionality by placing a client-specific XML-based, SQL-like query with the server, which then ensures that only relevant file portions and/or file attribute data are transferred. We exploit the fact that data sharing in scientific applications means transferring data on the order of GigaBytes, from one location to another, even when all the data attributes and rows might not be needed or when certain attributes can be combined to reduce the number of bytes being transferred per row [33]. Finally, as with the real-time collaborations outlined above, the amounts of data transmitted by filters can then be adjusted to match available network bandwidth.

3 IQ-Services: Software Architecture

3.1 Architecture Overview

IQ-Services. IQ-Services are application-specific code modules associated with data transport middleware. Figure 4 shows a prototypical real-time collaboration in which application-level data is distributed to remote collaborators and is manipulated by computational components like data clustering or data analysis[32]. The role of IQ-Services in this scenario is illustrated by the additional data filters and selectors associated as ‘handlers’ with the publish/subscribe event channels used for data distribution. As evident from the figure, IQ-Services do not replace application-level computations like data analysis or clustering. Instead, they form a ‘thin’, efficient layer of application-provided functionality that is placed ‘into’ middleware by applications, the purpose of which is to allow middleware to manipulate

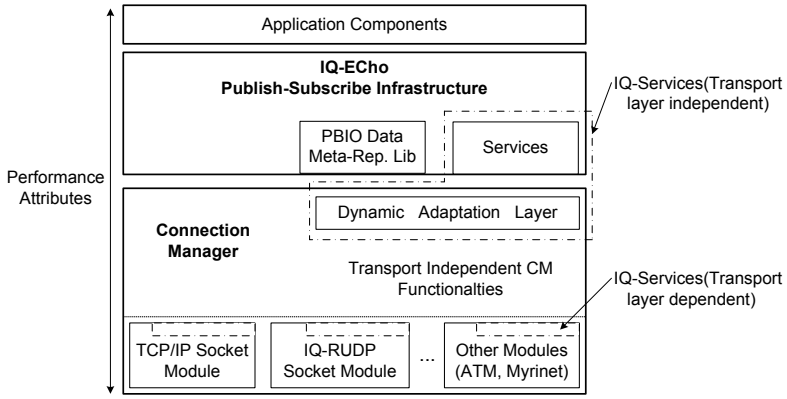

Figure 5: System Architecture Overview

data on its path from providers to consumers and to do so in conjunction with information about network behavior provided by communication protocols. In the IQ-ECho publish/subscribe infrastructure used in our implementation, this layer is comprised of dynamically created and deployed 'event handlers'.

IQ-Services may be associated with clients, servers, or intermediate nodes, thereby forming an overlay network. Typical configurations of overlays used in interactive applications are described in [32, 5]. We are not concerned with how to best map overlays to communication/processor networks, but note that the dynamic behavior of end users in wide area collaborations implies that it must be possible to create overlays dynamically, when desired by applications or when indicated by substantial changes in network or machine resources. IQ-ECho supports this by permitting the dynamic creation of overlay nodes and the runtime installation of services on those nodes, using dynamic binary code generation techniques [8].

Performance Attributes. Another element of IQ-Services are 'performance attributes', which implement IQ-Service/protocol interactions, by traversing multiple layers of the protocol stack (a) to provide application-level information like packet priorities or importance to communication protocols, and (b) to provide network-level monitoring information like available bandwidth or current packet loss to middleware-level services. Attributes may also be used to implement end-to-end performance-relevant interactions between data providers and consumers. Examples of such interactions include a client's use of performance attributes to set parameters in a server's data filter, and a server-side instruction of the middleware handler to upsample the data sent since additional network bandwidth is now available. End-to-end and cross-layer interactions via performance attributes are depicted by the solid, vertical arrow in Figure 5.

Transport Independence. Figure 5 also shows the multiple levels of abstractions used in IQ-ECho to map an application-level message submitted to an event channel to a message sent to the underlying protocol stack and communication socket used by a specific source-sink pair: after event submission, the event is mapped to a lower-level facility called the 'Communication Manager', which then sends the event to one of multiple communication protocols. This functionality permits IQ-ECho to run on top of multiple transport protocols, e.g., the standard TCP protocol and an instrumented version of RUDP developed in our research. This 'IQ-RUDP' protocol employs performance attributes (1) to provide feedback to the application-level handler also shown in the figure and (2) to adjust its data transmission behavior (e.g., using attributes to not retransmit certain data upon detected loss). In comparison, for TCP, for example, network measurement capabilities may be directly associated with communication services, as depicted in Figure 5 by the 'measurement' methods layered between event management and the communication manager. By placing such methods directly 'into' the communication stream, measurements can be performed using the application's native communications, rather than additional packet trains generated for such purposes. This is useful for the streaming data applications targeted by our work, but should be complemented by active bandwidth measurements for applications with intermittent or bursty communication behaviors.

For brevity, we do not describe the full set of layers depicted in Figure 5. Instead, we next briefly outline the IQ-RUDP transport layer that implements coordinated network and application-layer adaptation. Experiments with this layer demonstrate the utility of this approach in maintaining application performance in the face of changing network conditions.

3.2 IQ-RUDP and the Dynamic Adaptation Layer

The IQ-RUDP Instrumented Communication Protocol. IQ-RUDP is an open transport protocol intended to facilitate end system and application-layer adaptations to network conditions. To attain TCP-friendly behavior [24, 41], it also runs its own congestion control algorithm. The distinctive features of IQ-RUDP are:

- *Exposing performance metrics*. IQ-RUDP exposes certain transport-layer performance metrics to the higher layer by means of performance attributes. Metrics like bandwidth, RTT, and loss ratio can be exploited by IQ-Services when adapting the data being transported.
- *Callbacks*. Applications can register callbacks along with the conditions under which they should be triggered.
- *Adaptive and application-controlled reliability*. A typical adaptation adopted by applications is to lower reliability requirements in favor of increased transmission rates. IQ-RUDP provides prioritized reliability control for each application-level message.
- *Coordinated adaptations between the application and the transport*. IQ-RUDP first proposed the idea of coordinating between application-level adaptations and changes in transport-level behavior[15]. The advantages of coordinated adaptation are particularly apparent for high frequency adaptations that are driven by the protocol level, or when coordination can prevent conflicting adaptation decisions across different layers of a protocol stack.

The Dynamic Adaptation Layer (DAL). The primitives provided by IQ-RUDP comprise a basis for runtime traffic adaptation. However, it can be difficult to translate between application-level information and transport-level metrics, which limits the direct use of IQ-RUDP primitives by applications. Moderating between lower-level (e.g., IQ-RUDP) and application-level semantics and desires is the role of the Dynamic Adaptation Layer (DAL) in the IQ-Services architecture. The DAL's monitoring methods expose selected performance metrics to upper layers, including RTT, lossrate, throughput, etc. The DAL can also implement service-specific performance models. Model outputs exported via performance attributes then 'drive' adaptation methods realized in the DAL, which in turn drive the actual adaptations performed by IQ-Services. Current methods include packet reliability control and coordination across multiple, concurrent transport-level connections. Two performance 'models' utilized in our work employ (1) linear bandwidth regression and (2) an end-to-end method for measuring available bandwidth. In either case, the output of such dynamic measurement methods is made available to upper layers (i.e., to IQ-Services) via performance attributes or models. Conversely, applications or IQ-Services can use attributes to dynamically adjust certain parameters of bandwidth measurement algorithms, such as frequency of measurement and accuracy/overhead tradeoffs. Finally, the DAL can use additional threads to execute adaptation methods and/or periodically run network measurement tools like Pathload [18] or Netlets [37].

The following adaptation methods are currently integrated into the DAL:

- *Coordinated Packet Reliability Control* permits applications to define different levels of packet reliability requirements, which are then handled in a coordinated fashion by the combined actions of the IQ-Service/DAL/IQ-RUDP layers. In Section 4, per packet reliability indications are used to send high priority packets reliably, whereas low priority packets are sent unreliably and only when there are sufficient network resources.
- *Congestion Avoidance* implemented in the DAL uses application-level methods for controlling the traffic volume imposed on the underlying network. In this paper's experiments, the DAL interacts with the IQ-RUDP protocol to control the transport layer's sending rate and congestion control behavior, so as to avoid possible congestion, recover faster from existing congestion, and improve throughput while still maintaining TCP-like fairness properties. Specifically, the DAL periodically polls IQ-RUDP for available bandwidth and loss rate measurements and based on these measurements, adjusts the sending rate of the application messages 'pushed into' the IQ-RUDP layer. Since loss rates are updated more frequently than measurements of available bandwidth, they are used as 'quick' traffic adjustments in between bandwidth measurements.
- *Connection Coordination.* For multiple connections to/from a single address space, the DAL can implement coordination schemes for the traffic imposed on those connections. Consider a real-time collaboration in which

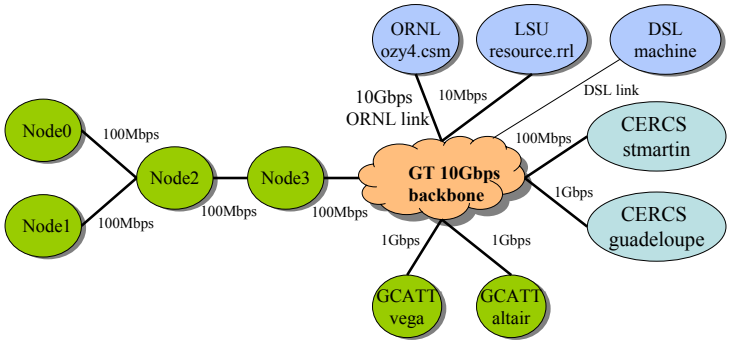

Figure 6: Testbed Configuration

Table 1: IQ-ECho Microbenchmarking

Data size(bytes)	Send side cost(ms)	receive side cost(ms)
100K	0.0088	0.0107
10K	0.0073	0.0098
1K	0.0066	0.0092

several clients communicating with a server. Since the clients' network connections differ, they will exhibit different RTTs and therefore, will not be treated fairly by TCP protocol when they share a common network bottleneck: clients with larger RTTs to the server will receive a smaller fraction of the shared bottleneck than those with smaller RTTs.The DAL can use application-level heuristics to implement collaboration-friendly traffic coordination schemes. This paper's simple 'fairness' scheme compensates for the unfairly treated client and improves synchronization among collaborators, by pacing the packets in the flow that is favored by the network.

4 Experimental Evaluation

This section presents experimental results that demonstrate the ability of the IQ-Services architecture and implementation to dynamically manage traffic behavior in response to network resource availability:

- demonstrations that coordinated network- and middleware-level adaptation can substantially improve the performance of wide area applications;
- sample adaptations performed for interactive high performance applications, focusing on online collaboration via large data sets; and
- comparisons of results attained with different network measurement methods.

4.1 Testbed and Microbenchmarks

The testbed used to evaluate coordinated network- and middleware-level adaptation consists of multiple wide- and local-area network links, as depicted in Figure 6 with link capacity labeled. The two CERCS nodes **stmartin** and **guadeloupe** are connected to an intranet backbone with 100Mbps and 1Gbps links, respectively. In another building (GCATT), **vega** and **altair** both have 1Gbps links up to the backbone. The RTTs between stmartin and guadeloupe are typically in the range of 0.06ms to 0.13ms. The RTTs between the GCATT and CERCS nodes are slightly larger (0.2ms to 0.4ms). The topology, bandwidth, and delay of **Node0** to **Node3** are configured through the Netlab(an Emulab site[22]) in Georgia Tech. These nodes are actually part of another cluster in CoC, and the RTTs between Node$x$ and CERCS nodes typically range from 0.3 to 0.5ms, while the RTTs between Node$x$ and GCATT nodes range from 0.5 to 0.8ms. The node at Oak Ridge National Laboratories has a 1Gbps uplink to a 10Gbps link to Atlanta, which in turn has multiple 1Gbps uplinks to Georgia Tech. Future experiments will use these links to evaluate operation across high end wide area links. The node at Louisiana State University represents a lower end Internet-connected collaborator, using a 10Mbps network link. Finally, a 'home' DSL link represents ill-connected collaborators.

Microbenchmarks. The microbenchmarks reported in Table 1 establish that our implementation of IQ-Services is capable of supporting the large-data applications they target. The table shows the send and receive-side costs introduced by IQ-ECho, using the testbed machines **isleroyale** and **guadeloupe**. The testbed machine **isleroyale** is a dual processor Intel XEON at 2.8GHz with 2GB of memory while **guadeloupe** is a dual processor Intel XEON at 2.0Ghz with 1GB of memory; the OS on both the machines being Redhat Linux 9. Only one processor of each testbed machine is used for microbenchmarking. In Table 1, Send side cost is the time between an application submitting data for transmission to the time at which the infrastructure invokes the underlying network 'send()' operation. Receive side costs represent the time between the end of the 'receive()' operation and the point at which the application receives the data. Since these costs are in the range of 0.006ms to 0.0107ms, the resulting overheads are small compared to the typical round trip delays experienced in local area networks (0.1ms-0.3ms in our environment) and negligible for typical wide area round trip delays (50ms-100ms). Additional performance information comparing IQ-ECho's performance to that of other high performance communication infrastructures (e.g., MPICH) appear in [8].

Over our targeted wide area network, these overhead which are less than 0.011 are quite trivial. Even for a high-speed local area network, e.g. the local network between isleoryal and guadeloupe which are connected with a high performance CISCO Catalyst 6513 switch, the typical round trip delay is 0.1ms-0.3ms, and 0.01ms cost introduced by IQ-ECho is also comparably small. [8] also gives costs breakdown comparision between ECho, COBRA, MPICH and XML, and shows that ECho provides efficient data transmission with significantly lower overhead.

4.2 Experimental Results with a Real-time Collaboration

Coordinated Packet Reliability Control. Our first experiment shows the effectiveness of coordinating reliability control between IQ-RUDP and IQ-services, for collaborative applications that use application-specific methods of dealing with data loss. In this example, a bondserver (as described above in Section 2.1) application component sends data over a lossy link (from a home DSL machine to a campus machine, **guadaloupe**). There are three types of application messages, and in the order of importance, they are: messages with atoms and bonds information [AB], messages with atoms information only [A], and messages with bonds information only [B]. In addition, different types of messages are grouped together according to the pattern (AB, B, B, A, B). There is often redundant information among the messages in one group. Overall, AB messages have to be delivered, both type A and B messages can be lost, where type B messages have lower priorities than those of type A.

The coordinated reliability control algorithm implemented jointly by the DAL and by IQ-RUDP dynamically labels application-level messages with certain priorities and then uses them to differentiate message transmissions, as follows:

- discards the lowest priority messages without transmitting them;
- attempts to transmit the second priority messages, but will not try to retransmit them if they are lost; and
- guarantees the reliable delivery of the highest priority messages.

The experiments shown set a target rate for type AB messages. When there are significant packet losses and the target rate is not achieved (as detected by IQ-RUDP), the DAL layer adaptation algorithm adjusts message priorities, by setting some portion of type B messages to the lowest priority, and by setting type A messages to the second priority. The idea is to give the application-level (i.e., the IQ-Services layer) the ability to distinguish which data are most important, thereby permitting the network layer to focus on transporting those parts.

Results in Figure 7 show that coordinated packet reliability control successfully achieves the different target rates set a priori, at the cost of an increased loss ratio of type B messages. Type A messages, which are also much smaller than the other two types, have very few losses.

Coordinating the Transfer of Two IQ-RUDP Connections. In this experiment, the DAL layer coordinates message transmission across two IQ-RUDP connections used by a single application, one example being the co-existence of a control and a data connection in a remote collaboration, another being the co-existence of a large-data connection (e.g., visualization data) with a video connection (e.g., for video conferencing). The specific experiment conducted emulates two remote clients that receive the same visualization data stream. Since these clients are engaged in a real-time collaboration, their desire is to receive and display the same data at the same time, without undue buffering costs.

For simplicity, the experiments uses two connections that share a bottleneck link but have different RTTs and therefore, have different throughput. To better synchronize the two connections, message transmission behavior to

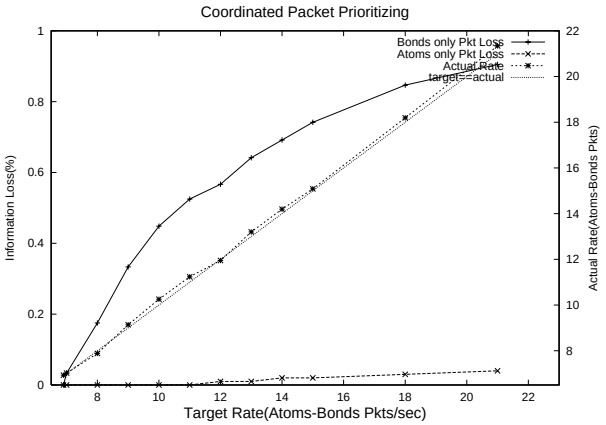

Figure 7: Prioritized Reliability Control

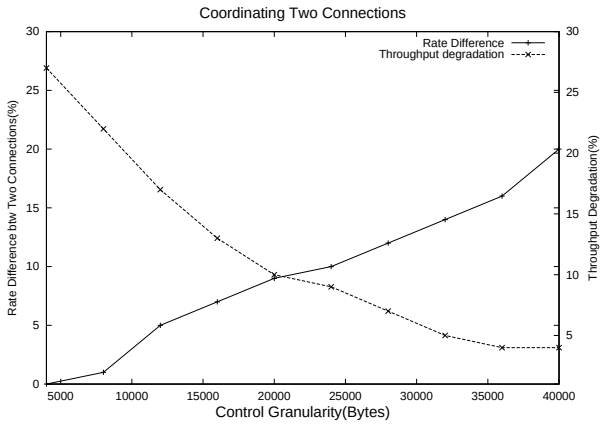

Figure 8: Coordination between Connections

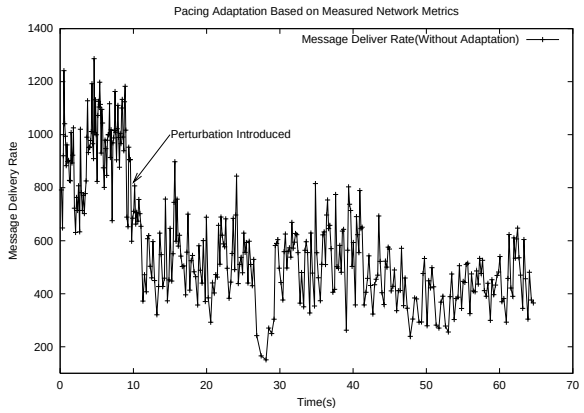

Figure 9: Without Packet Pacing Adaptation

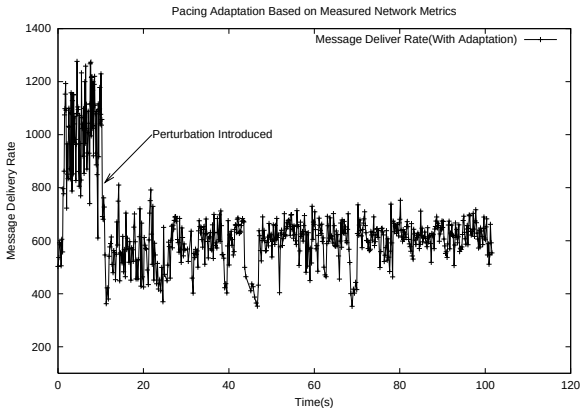

Figure 10: With Packet Pacing Adaptation

each client is adjusted at the IQ-Services layer in response to network measurements at the IQ-RUDP layer: when the application starts, it establishes each IQ-RUDP connection and measures the available bandwidth; IQ-RUDP makes available to the DAL dynamic throughput information about both connections, which is used to slow down the faster connection in proportion to the ratio of the two throughput measurements.

The goals of this experiment are twofold. First, it demonstrates the viability of application-level, network-aware control over multiple connections. Second, it evaluates 'granularity' issues concerning such control, the purpose being to better understand potential control limitations due to application constraints like delays due to differences in the sizes of application-level messages being sent. In Figure 8, the X-value is the size of the data unit for which the sending rate is controlled. The left Y-value is the receiving rate difference between the two receivers. The right Y-value is the throughput degradation of the faster connection. We can see that finer grained control achieves better synchronization, at the cost of slowing down the faster connection, i.e., not fully utilizing the network bandwidth.

In separate experiment, we use the DAL layer to make two connections share a bottleneck link fairly, using the bandwidth and RTT measurements at the connection layer. The application adjusts the sending rate based on the RTTs of the two connections. The adjustment algorithm is based on the TCP throughput formula, which shows that two connections sharing the same bottleneck link have throughputs that are inversely proportional to RTT^2. The results attained are very similar to those in Figure 8, although different IQ-RUDP layer measurements are used.

Table 2: Packets Pacing Adaptation - Performance Comparison

	Message Delivery Rate(f/s)	Normalized Standard Deviation	Jitter(ms)
Without Adaptation	443.99	0.27	2.41
With Adaptation	599.39	0.17	1.76

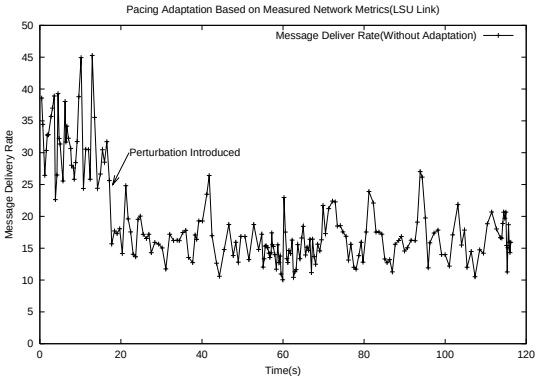

Figure 11: Without Packet Pacing Adaptation (LSU Link)

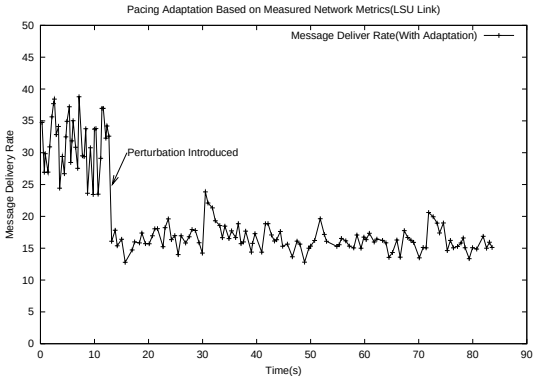

Figure 12: With Packet Pacing Adaptation (LSU Link)

Table 3: Packet Pacing Adaptation - Performance Comparison(LSU Link)

		Message Delivery Rate(f/s)	Normalized Standard Deviation	Jitter(ms)
Exp1(Figure 11, Figure 12)	Without Adaptation	16.11	0.20	61.9
	With Adaptation	16.45	0.17	55.6
Average Over 5 Experiments	Without Adaptation	16.05	0.18	60.2
	With Adaptation	16.25	0.17	54.9

Dynamic Packet Pacing. This experiment shows that a 'pacing adaptation' based on measured network behavior can both improve and smooth throughput when there is cross traffic. When the bond server transfers large amounts of data to some specific client, we inject cross-traffic into the network, thereby decreasing available network bandwidth. With IQ-Services, network metrics are exposed through performance attributes to middleware and to application modules, thereby making both 'aware' of changes in network status and permitting them to adjust their behavior accordingly. In this experiment, they adjust the way in which packets are sent out (packet pacing). The packet size used is 66KB, and the injected cross-traffic over the 1Gbps link is 650Mbps, generated by iperf[9].

Figure 9 illustrates the measured message delivery rate at the client side from a server that is unaware of the cross-traffic and cannot adapt to changes in network behavior. Figure 10 shows much better performance when the cross-traffic is injected, since in this case, the server is made aware of the measured loss rate and the available bandwidth and then paces its packets accordingly (at a better rate). The idea is to ensure that its offered throughput does not exceed available bandwidth, thereby resulting in reduced congestion and packet losses. The message delivery rate improves from 434f/s to 599f/s, and it is much smoother: the normalized standard deviation of the message delivery rate decreases from 0.27 to 0.17, and jitter decreases from 2.41ms to 1.76ms. These result are attained with TCP-based bandwidth measurement techniques in [36] and loss rate measurement. The idea is for the server to adapt quickly to network changes by use of loss rate measurement and to avoid oscillations by use of bandwidth prediction.

A similar experiment is performed between **stmartin.cc.gatech.edu** and **resource.rrl.lsu.edu**, as shown in Figure 11 and Figure 12. This link has higher fluctuation in terms of available bandwidth and RTT, since there is significant and often unpredictable traffic that also impacts the link besides the traffic we generate. Figure 11 and Figure 12 demonstrate how both throughput and jitter are improved in a typical run of the experiment. Table 3 shows the average performance improvement over 5 experiments.

Adaptive Downsampling in Congested Networks. As demonstrated via emulation in [15] and experimentally in this technical report, IQ-Services can regulate the traffic imposed on the underlying network by 'pacing' application-level messages. Packet pacing can effectively reduce congestion and maintain better message delivery rates. However, its benefits are limited when the available bandwidth is already below the threshold at which it becomes impossible to deliver messages as fast as required by an ongoing real-time collaboration. This issue is addressed by deploying application-level data filters as IQ-Services and then using them to downsample the actual data being sent prior to submitting it to the network transport.

Experimental results shown in Figures 13 and 14 demonstrate the necessity and effectiveness of IQ-Services-level

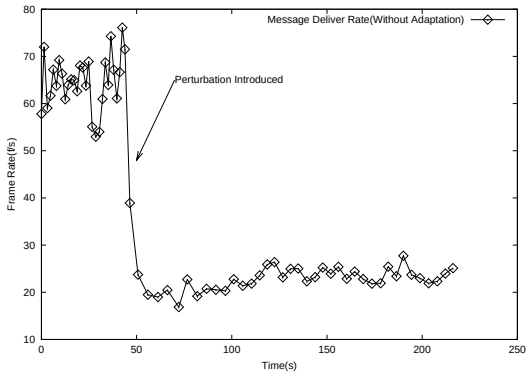

Figure 13: Without IQ-Service Metadata-Based Filtering

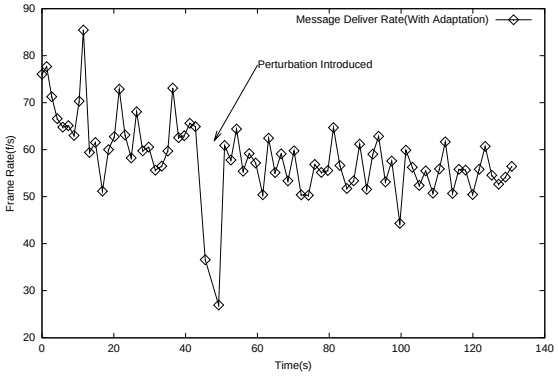

Figure 14: With IQ-Service Metadata-Based Filtering

adaptation through dynamic data downsampling. Here, large cross traffic (850Mbps) is injected into the network. Using a simple data pacing algorithm, the server can only deliver 23.3 f/s(frames per second, average over 5 experiments) to the client. However, when permitting the client to characterize the subset of data most important to it, when congestion occurs, it can install an IQ-Service data filter at the server side and get essential data at satisfactory speed. The data downsampler used removes data relating to visual objects that are not in the user's immediate field of view. That is, the client transfers the current position and view point of the user to the filter, which computes at the server side what data set the user is watching and then transfers exactly those data sets. Such data selection actions are taken in conjunction with the network layer, in order to adjust the data amounts sent to match available network bandwidth.

4.3 Experimental Results with IQ-GridFTP

The implementation of IQ-GridFTP evaluated in this section replaces the transport layer of GridFTP with IQ-ECho, thereby making it easy for end users to deploy desired IQ-Services 'into' FTP transfers. As with real-time collaborations, such filters may be deployed statically, or via dynamic linking methods, or written with E-Code, a highly portable subset of C, which is dynamically executed at the source. A return value of one from the filter function causes the current filtered row to be transmitted over the network, while a return value of zero causes it to be discarded.

We use XML to represent the structure of data contained in the files transported by IQ-GridFTP, specifically, whenever a client needs to do a partial retrieve on a structured file, say weather.dat, a format description file, say weather.dat.xml (we assume that such a file exists for large structured files), is first fetched from the source. Next, the requesting client creates an IQ-ECho structured data channel using the information from the xml file, and then generates E-Code for the filter using the information supplied. Using IQ-ECho's facilities for dynamic deployment, the IQ-Service that realizes this filter can dynamically: (1) select the specific file attributes that need to be transferred, (2) select the rows required, and (3) perform data reduction operation like averaging, etc., at the data source, thereby reducing the data volumes transmitted.

We compare the performance of the IQ-GridFTP with basic GridFTP in several ways. First, we compare the two GridFTP implementations' effective throughput with and without cross traffic, and 50Mbps cross traffic, injected via iperf into a 100Mbps network that connects two machines running FTP. Figure 15 depicts the throughput time series for a single experiment, and Table 4 depicts the average attained throughput and its normalized standard deviation for 5 experiments. These basic experiments demonstrate that IQ-GridFTP achieves throughput similar to that of GridFTP. Second and more interestingly, Figure 16 and Table 5 demonstrate the advantage of utilizing IQ-Services in conjunction with GridFTP. Here, a large number of data files are transferred from an FTP server to a client. In IQ-GridFTP, when the client finds the frame rate to be lower than required, it creates a data downsampling filter (an image resolution reduction filter when data is being visualized) and deploys it on the server, then uses the filter to reduce data volume, when necessary.

Results demonstrate the importance of domain-specific data downsampling even for general mechanisms like

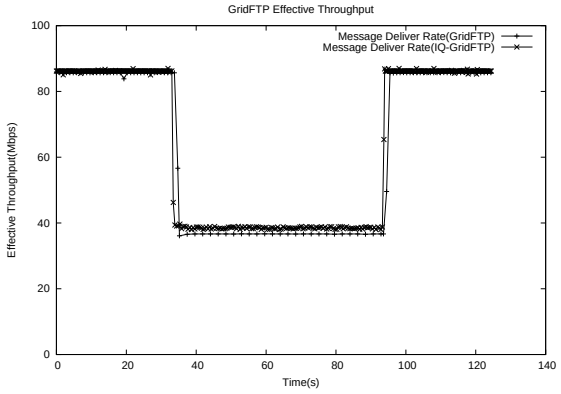

Figure 15: IQ-GridFTP Performance Analysis

Table 4: GridFTP Performance Analysis

Cross Traffic	Transport Protocol	Average Throughput(Mbps)	Normalized Standard Deviation
No	TCP	85.62	0.002470
No	IQ-RUDP	86.18	0.002459
Yes	TCP	36.64	0.006064
Yes	IQ-RUDP	37.52	0.006602

GridFTP. When there is 60Mbps cross traffic in the link, the frame rate at which the GridFTP server can transfer data to the client drops to approximately 15 frames per second, from 62 frames without network congestion. In contrast, IQ-GridTP can transfer downsampled, reduced-size images at approximately 60 frames per second. Table 5 lists the average frame rate and average normalized standard deviation of GridFTP and IQ-GridFTP over 5 experiments. With adaptation, the frame rate is improved from 14.66 frames per second to 59.50 frames per second and the normalized standard deviation is improved from 0.045 to 0.017. The response time of such an adaptation (from the time the available bandwidth is reduced until the client starts to receive adapted image files) is also small, ranging from 0.12 to 0.25 seconds.

Figure 17 compares the performance of GridFTP and IQ-GridFTP with dynamic cross traffic derived from an real trace [1]. Cross traffic is injected via UDP, thereby reasonably emulating actual network behavior. The data the server transfers to the client is climate data (the size of each record/frame is 172.8K bytes). The client sends a filter to the server to specify whether it wants IQ-GridFTP to adjust data precision when congestion is noticed and if so, what percentage of the data will be adjusted. From Table 6, it is clear that the frame rate(averaged over 5 experiments) is improved from 27.34f/s to 30.03f/s, where the normalized standard deviation of frame rate is reduced from 0.11 to 0.04.

Additional capabilities of IQ-GridFTP now under consideration by our group include per connection adaptations like dynamic window size adjustments and TCP buffer size auto-tuning, as well as fairness improvements or stream synchronization when multiple connections are used for single, large file transfers. The latter is important for storage systems (e.g., DPSS, HPSS) that utilize parallel data transfers and data striping across multiple servers to improve performance.

5 Discussion and Related Work

A characteristic differentiating IQ-Services from previous research is its ability to coordinate system- or network-level reactions to resource changes with application-level adaptations. This extends prior work on application-level flow control or data striping [3, 42], as well as approaches that focus on the system-level provision of network status in-

[1]Trace file BWY-1063337799-1.asc from NLANR trace repository(pma.nlanr.net), collected at Columbia Univ (BWY site) on Sep 12th, 2003.

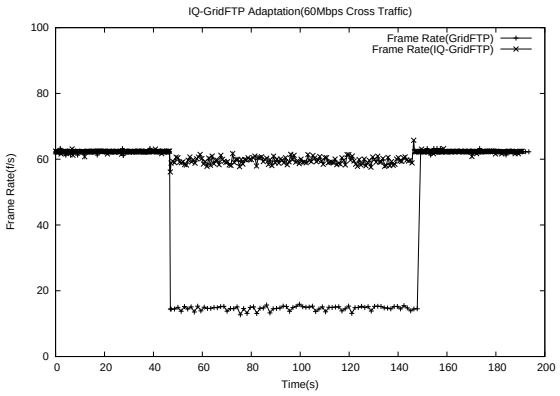

Figure 16: IQ-GridFTP Adaptation(60Mbps Cross Traffic)

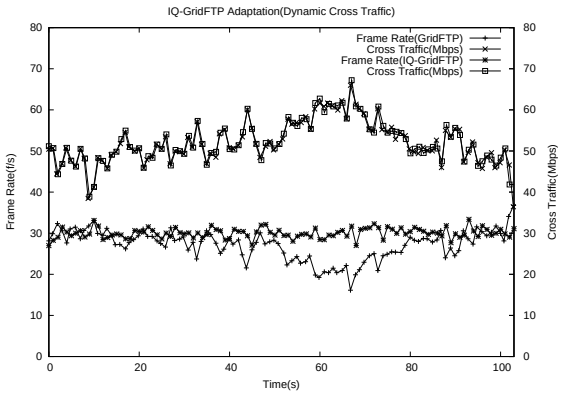

Figure 17: IQ-GridFTP Adaptation(Dynamic Cross Traffic)

Table 5: IQ-GridFTP Adaptation(60Mbps Cross Traffic), Average over 5 Experiments.

Cross Traffic	Implementation	Average Frame Rate(f/s)	Normalized Standard Deviation
No	GridFTP	62.23	0.004017
No	IQ-GridFTP	62.30	0.003871
Yes	GridFTP	14.66	0.04510
Yes	IQ-GridFTP	59.50	0.01665

formation [30]. It also distinguishes our work from the many, middleware-level adaptation infrastructures developed in previous research, including BBN's Quo [51] or other object-based infrastructures [50]. Furthermore, IQ-Services does not require adaptations to be performed in certain ways or at certain system levels. Instead, they can be 'driven' from the system and/or application levels. One consequence is that reactions to changes in available network bandwidth, for example, can occur at packet boundaries rather than at the boundaries defined by application-level message sizes and/or by applications' time slices. The resulting 'faster' and bounded reaction times can reduce jitter and improve system predictability [15, 38].

An important characteristic of our middleware is its support of user-defined 'handlers' – IQ-Services – that implement the actual data adaptations suitable for specific applications. As a result, in comparison to previous domain-specific solutions (e.g., for the multimedia domain [41, 24]), the IQ-Services architecture can be used to implement adaptive methods for a variety of target applications. We demonstrate this capability by implementing an IQ-Services version of GridFTP, termed IQ-GridFTP. We note that 'handlers' like those used in IQ-Services are similar to earlier work in object-oriented operating systems, as with the application-specific policy objects used in [10] or the subcontracts used in [14], the latter also present in the implementation of RMI in Java. In fact, in recent work, we have associated policy objects and data handlers with SOAP-based communications, in order to dynamically adjust data transmissions to changes in network resources or client needs [39].

6 Conclusions and Future Work

The software architecture of IQ-Services shown in Figure 5 offers developers the ability to insert application-specific, lightweight services into data exchange middleware and mechanisms. In this paper, IQ-Services are used to support

Table 6: IQ-GridFTP Adaptation(Dynamic Cross Traffic), Average over 5 Experiments

Implementation	Average Frame Rate(f/s)	Normalized Standard Deviation
GridFTP	30.03	0.04076
IQ-GridFTP	27.34	0.11218

the efficient exchange of scientific data in real-time collaborations, by dynamically adjusting the data sent from information providers to information consumers [23, 13, 7]. IQ-Services are also used to implement adaptive file transfers via an IQ-version of GridFTP. In concurrent work, we are applying the architecture to other communication paradigms defined by the grid community, such as the M-by-N data exchanges used in remote storage, monitoring, or visualization systems [12]. In addition, we have used IQ-Services to create resource-aware communication services that apply general compression methods to data being exchanged across wide area networks [47]. Finally, in [48] and in ongoing extensions of that work, we are using IQ-Services to create adaptive remote graphical displays, for the high end 3D visual depictions required by applications like molecular dynamics.

The performance improvements attained by use of IQ-Services can be substantial, including up to 25% improvements in message delivery rates when information sources 'pace' the data offered in conjunction with available network bandwidth, and almost threefold improvements in message rates when a client-specific data downsampling service is used to control the amounts of data sent from data server to client.

Future work will compare service-level adaptations that utilize different network-level techniques for assessing current network bandwidth [36, 28]. It will also utilize overlay networks to combine the lightweight data filtering and downsampling methods used in this paper with heavier-weight methods for data transformation and personalization executed by additional machines interposed into the path between data providers and consumers [4, 32]. Such work will dynamically deploy lightweight IQ-Services to utilize alternative network and machine paths from data providers to consumers.

Acknowledgments

We acknowledge the help of Constantinos Dovrolis and Nagi Rao in defining the network testbeds used in this paper's experimentation. Nagi Rao also made available the Internet-connected machine used for wide-area measurements, and his network measurement methods have been integrated into the current version of IQ-ECho. Neil Bright and the Utah Emulab team spent many hours to set up Georgia Tech's NetLab facility.

References

[1] M. Aeschlimann, P. Dinda, L. Kallivokas, J. Lopez, B. Lowekamp, and D. O'Hallaron. Preliminary Report on the Design of a Framework for Distributed Visualization. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 1833–1839, Las Vegas, NV, June 1999.

[2] AG. Access Grid. `http://www-fp.mcs.anl.gov/fl/accessgrid`.

[3] M. Allman, H. Kruse, and S. Ostermann. An Application-level Solution to TCP's Satellite Inefficiencies. In *Proceedings of the International Workshop on Satellite-Based Information Services(WOSBIS)*, Nov. 1996.

[4] F. Bustamante, G. Eisenhauer, P. Widener, K. Schwan, and C. Pu. Active Streams: An Approach to Adaptive Distributed Systems. In *Proceedings of the 8th Workshop in Operating Systems (HotOS-VIII)*, Elmau/Oberbayern, Germany, May 2001.

[5] P. Chandra, A. Fisher, C. Kosak, and P. Steenkiste. Network Support for Application-Oriented Quality of Service. In *Proceedings of IEEE/IFIP International Workshop on Quality of Service*, May 1998.

[6] Y. Chen, K. Schwan, and D. Zhou. Opportunistic Channels: Mobility-Aware Event Delivery. In *Proceedings of ACM/USENIX International Middleware Conference*, 2003.

[7] DOE-TSI. TeraScale Supernova Initiative. `http://www.phy.ornl.gov/tsi`.

[8] G. Eisenhauer. The ECho Event Delivery System. Technical Report GIT-CC-99-08, Georgia Tech, Aug. 1999.

[9] N. L. for Applied Network Research". Iperf - The TCP/UDP Bandwidth Measurement Tool. `http://dast.nlanr.net/Projects/Iperf/`.

[10] A. Gheith and K. Schwan. Real-Time Objects and Atomicity for Multiprocessors. *Advances in Real-Time Systems*, 1994.

[11] Globus. GridFTP. http://www-fp.globus.org/datagrid/gridftp.html.

[12] Globus. http://www.globus.org/ogsa/. An Open Grid Services Architecture, 2003.

[13] GriPhyN. The Grid Physics Network. http://www.griphyn.org.

[14] G. Hamilton, M. Powell, and J. Mitchell. Subcontract: A Flexible Base for Distributed Programming. In *Proceedings of ACM Symposium on Operating Systems Principles*, pages 69–79, 1993.

[15] Q. He and K. Schwan. IQ-RUDP: Coordinating Application Adaptation with Network Transport. In *High Performance Distributed Computing*, July 2002.

[16] H.-Y. Hsieh and R. Sivakumar. A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts. In *Proceedings of ACM/IEEE MOBICOM*, September 2002.

[17] C. Isert and K. Schwan. ACDS: Adapting Computational Data Streams for High Performance. In *Proceedings of IPDPS*, May 2000.

[18] M. Jain and C. Dovrolis. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proceedings of ACM SIGCOMM*, Aug. 2002.

[19] J. Jancic, C. Poellabauer, K. Schwan, M. Wolf, and N. Bright. dproc - Extensible Run-Time Resource Monitoring for Cluster Applications. In *Proceedings of International Conference on Computational Science*, 2002.

[20] J. Kephart and D. Chess. The Vision of Autonomic Computing. *Computer Magazine*, Jan. 2003.

[21] T. Kim and M. Ammar. Optimal Quality Adaptation for MPEG-4 Fine-Grained Scalable Video. In *Proceedings of IEEE INFOCOM*, Apr. 2003.

[22] J. Lepreau. The Utah Network Testbed. http://www.emulab.net/. University of Utah.

[23] LSC. http://www.ligo.org/. LIGO Scientific Collaboration, 2003.

[24] J. Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control, Jan. 1997.

[25] G. M. Mair. Telepresence - The Technology and Its Economic and Social Implications. In *Proceedings of IEEE International Symposium on Technology and Society*, 1997.

[26] M. Mathis. Web100 and the End-to-End Problem. http://www.web100.org/docs/jtech/.

[27] D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, A. Goel, P. Wagle, C. Consel, G. Muller, and R. Marlet. Specialization Tools and Techniques for Systematic Optimization of System Software. *ACM Transactions on Computer Systems*, 19(2):217–251, May 2001.

[28] M.Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions in Networking*, Aug., 2003.

[29] NASA. Using XML and Java for Telescope and Instrumentation Control. In *Proceedings of SPIE Advanced Telescope and Instrumentation Control Software*, 2000.

[30] Net100. http://www.net100.org/. The Net100 Project-Development of Network-Aware Operating Systems, 2001.

[31] NPACI. Grid Portals. http://gridport.npaci.edu.

[32] B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From Interactive Applications to Distributed Laboratories. *IEEE Concurrency*, 6(3), 1998.

[33] B. Plale, P. Widener, and K. Schwan. Taking the Step From Meta-information to Communication Middleware in Computational Data Streams. In *Proceedings of IEEE Heterogeneous Computing Workshop*, 2001.

[34] P.Tinnakornsrisuphap, W. Feng, and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2000.

[35] C. Pu. The InfoSphere Project. `http://www.cc.gatech.edu/projects/infosphere`.

[36] N. S. Rao, Y.-C. Bang, S. Radhakrisnan, Q. Wu, S. S. Iyengar, and H. Choo. NetLets: Measurement-based Routing Daemons for Low End-to-End Delay over Networks. *Computer Communications*, to be published.

[37] N. S. V. Rao, S. Radhakrishnan, and B. Y. Cheol. NetLets: Measurement-based Routing for End-to-End Performance over the Internet. In *Proceedings of International Conference on Networking*, 2001.

[38] D. Rosu and K. Schwan. FARACost: An Adaptation Cost Model Aware of Pending Constraints. In *Proceedings of IEEE RTSS*, Dec. 1999.

[39] B. Seshasayee, K. Schwan, and P. Widener. SOAP-binQ: High-Performance SOAP with Continuous Quality Management. In *Proceedings of ICDCS*, Mar. 2004.

[40] L. Sha, X. Liu, and T. Abdelzaher. Queuing Model Based Network Server Performance Control. In *Proceedings of Real-Time Systems Symposium*, Dec. 2002.

[41] D. Sisalem and H. Schulzrinne. The Loss-delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme. In *Proceedings of NOSSDAV*, Jul 1998.

[42] H. Sivakumar. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *Proceedings of IEEE/ACM Supercomputing Conference*, Nov. 2000.

[43] M. Trivedi, B. Hall, G. Kogut, and S. Roche. Web-Based Teleautonomy and Telepresence. In *Proceedings of SPIE Optical Science and Technology Conference*, 2000.

[44] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. In *Proceedings of SPIE Applied Imagery Pattern Recognition Workshop*, Washington, DC, Oct. 1997.

[45] R. F. Walters, B. B. Douglas, T. C. Leamy, and W. Yaksick. RC (Remote Collaboration): A Tool for Multimedia, Multilingual Collaboration, 2000.

[46] R. West and C. Poellabauer. Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams. In *Proceedings of IEEE Real-Time Systems Symposium*, 2000.

[47] Y. Wiseman and K. Schwan. Efficient End to End Data Exchange Using Configurable Compression. In *Proceedings of ICDCS*, Mar. 2004.

[48] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smart Pointers: Personalized Scientific Data Portals in Your Hand. In *Proceedings of IEEE/ACM Supercomputing Conference*, Nov. 2002.

[49] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[50] D. Xu and K. Nahrstedt. Supporting Multimedia Service Polymorphism in Dynamic and Heterogeneous Environments. Technical report, 2000.

[51] J. A. Zinky, D. E. Bakken, and R. E. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1), 1997.