IQ-RUDP: Coordinating Application Adaptation with Network Transport

Qi He, Karsten Schwan
qhe, schwan@cc.gatech.edu
Center for Experimental Research in Computer Systems
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332

Abstract

Our research addresses the efficient transfer of large data across wide-area networks, focusing on applications like remote visualization and real-time collaboration. To attain high performance in the real-time exchange of data across collaborating machines and end users, we are developing and evaluating methods and techniques for coordinating application-level with network transport-level adaptations of data communication. Specifically, complementing previous work on TCP-friendly communication and on adaptive transport protocols, our approach is to strongly coordinate application-level with transport-level changes in communication behavior, so as to best meet application needs without violating fairness in network resource usage. The approach is evaluated with the IQ-ECho middleware, which implements the distribution of scientific data to remote collaborators. Using IQ-ECho, application-level adaptations like selective data down-sampling are triggered by transport-level information provided by the instrumented IQ-RUDP protocol underlying IQ-ECho's communications. The application- to network-layer exchange of information necessary for such coordinated adaptations is implemented with ECho attributes, which provide a lightweight way for an application to provide quality of service information and to describe its adaptation to the transport layer, and for IQ-RUDP to share network status information with an application. In addition to triggering application-level adaptations and reacting to certain changes in network state, IQ-RUDP also re-adapts its own communication behavior after an application adaptation has been performed, in part to remain fair to other network flows. Such transport-level reactions can be performed at higher rates and with smaller overheads than possible at application level. The evaluation of IQ-RUDP and of its coordination schemes demonstrates the superiority of asynchronous, coordinated adaptations vs. adaptations performed only at protocol- or application-level. Specifically, coordination avoids conflicts due to mismatched application- vs. transport-level adaptations, and it avoids over-reaction due to changes performed simultaneously at multiple levels. In addition, by permitting IQ-RUDP to adjust its behavior independently, mismatches in the application- vs. network-level granularities of adaptation can be ameliorated. Finally, with IQ-RUDP's coordination, application performance is improved by reducing the impact of obsolete information used for application-level adaptation.

Keywords: large-data transfer, wide-area networks, real-time collaboration, reliable UDP, adaptive transport, adaptive reliability, coordinated adaptation,network-aware adaptation

1 Introduction

Distributed applications 'stressing' wide are networks include telepresence[16], remote collaboration and visualization,remote instrument control[10], and monitoring and surveillance[16]. Problems arise because large amounts of data must be transferred with low latency across wide area networks. These problems are due to multiple factors, including the heterogeneity 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.

Researchers have devised multiple solutions to the substantial delays incurred by the transfer of large data across wide area networks. Recent work on TCP-friendly communication[9] and on dynamic buffer right-sizing[4] attempts to reduce bursty traffic behavior by controlling the ways in which applications or protocols send data across the network. Specifically, while the AIMD (Additive Increase Multiplicative Decrease) algorithm used by the dominating transport protocol TCP is shown to converge to a fair state in terms of bandwidth usage, its adaptation behavior makes TCP traffic bursty in nature and delivers unstable QoS over time, which is particularly problematic for large data transfer[4, 15] and for real-time applications[9]. TCP-friendly congestion control algorithms[9, 14] focus on achieving a relatively stable sending rate that is adapted to the network's level of congestion. Improved bandwidth can also be realized by concurrent use of multiple routes or even sockets[15]. An alternative solution is to use application-level information at the protocol level to alter retransmission behavior, such as implementing partially reliable network transports that are sensitive to application needs[6]. Finally, reacting to calls for new communication protocols, multiple variants of TCP have been designed, in part to maintain the end-to-end congestion control deemed crucial for fairness in network use and to avoid congestion collapse[9].

While building on previous work on communication protocols, our work also leverages the substantial research that has been conducted in the areas of adaptive systems and applications in general[1]. Specifically, our approach is to have the transport level initiate and coordinate changes in communication behavior with the network and the application. Such *coordinated adaptations* (1) permit changes in communication behavior that are based on network measurements naturally accessible at the transport level, and (2) they also initiate responses to application-level changes, such as increasing the (packet-based) window size after an application adaptation that down-samples data, thereby reducing the sizes of application-level messages. We specifically investigate three cases where coordination is important to overall performance:

1. to resolve conflicts between application- and transport-level adaptations,

2. to consider and adjust the cumulative effects of application- and transport-level adaptations, thereby avoiding over-reaction to changes in the communication environment, and

3. to deal with the limited granularity of adaptation available at application level[13], where a transport-initiated adaptation may not be performed sufficiently fast, whereas the transport can react at much higher rates.

Our approach is evaluated with the IQ-ECho middleware, which implements the real-time distribution of scientific data to remote collaborators. Using IQ-ECho, application-level adaptations like selective data down-sampling are triggered by transport-level information that is provided by the instrumented IQ-RUDP protocol that underlies IQ-ECho's communications. The IQ-RUDP approach of coordinated adaptation enhances a system's ability to react to changes in the environment, by combining the transport's ability to react quickly, combining detailed information about network state with the application's ability to understand the complex relationships between its multiple dimensions of quality of service, such as the delay in information transfer weighed against the level of precision of the information being transferred. Specifically, system monitoring costs[11] are reduced by making available to the application only the monitoring data needed to decide upon how it should adapt. The costs of making adaptation decisions are reduced by having both the transport and application levels perform the adaptations for which they are best suited, such as performing high rate or monitoring data-intensive adaptations at the transport level while performing adaptations that consider complex application or system characteristics at the middleware or application levels. The costs of carrying out adaptations (i.e., adaptation enactment) are reduced by 'piggybacking' adaptations onto already existing transport- or application-level actions, such as performing an adaptation upon message retransmission or adapting application behavior prior to sending a message. Finally, the costs of coordination are small since such actions are integrated into adaptation actions at the protocol and application levels.

1.1 Related Work

Previous research on adaptive systems and applications has typically used middleware- or system-based approaches to supporting runtime adaptation[1], in part because multiple resources (e.g., CPU cycles, network bandwidth) must be allocated and controlled across multiple users and machines. In comparison, this paper focuses on the network resource, since that is critical to the large-data real-time collaborations targeted by our work. Moreover, since we seek to perform adaptations with smaller overheads than those experienced by general approaches to application adaptation[13], we integrate adaptation support into the ECho middleware already used for real-time scientific collaboration. While IQ-ECho is the vehicle used in this paper, similar results would be attained if the methods described here were integrated into grid-based software developed for large file transfers or into other frameworks for remote visualization or telepresence.

There has been substantial previous research on transport protocols that adapt to changes in network condition. First, prior work adapts congestion windows or rates[9], to provide certain transport choices to applications[14, 9], to address specific application-level traffic characteristics, or to support certain adaptation strategies. One example is a TCP-friendly protocol that runs a rigorous congestion control algorithm so as to tightly control the traffic pattern it generates. Such tight control will make it unsuitable for large-data applications that not only generate data at their own rates and times, but that also desire certain limits on the end-to-end latencies experienced by end users. Furthermore, the separation of transport and application adaptation evident may be suitable for rate-adapted multimedia applications, but is neither viable for the rich set of application adaptations targeted by our research, nor can it achieve both bandwidth fairness and desired application performance. A second stream of previous research translates application-level QoS requirements to concrete resource demands, based on which transport adaptation is performed, thereby making the transport the active controller of adaptations. An issue with such work is the difficulty of translating QoS expressions from quantities meaningful to applications to those interpretable by the transport layer[7], potentially leading to the use of partial or even incorrect information during adaptation. A third approach to transport adaptation is to simply provide applications with all of the network information they need[11], thereby making them the active controllers[8, 17]. Issues with this approach includes its assumption of correct application behavior and the need for applications to perform adaptations at rates commensurate with the rates of change in network state. Results in this paper demonstrate that large-data applications cannot be assumed to react at such rates.

In summary, previous work has established that (1) a transport protocol that tries to be fair to other flows cannot delegate all adaptation decisions to the application, while (2) delegating all application adaptations to the transport protocol can limit available adaptation choices and lead to complicated QoS translation procedures. In response, IQ-ECho and its underlying adaptive IQ-RUDP transport protocol cooperatively perform adaptations at both the transport and application levels, using coordination schemes suitable for the target applications.

2 IQ-RUDP: Coordinated, Adaptive Reliable UDP

IQ-RUDP is an extension of Reliable UDP(RUDP)[2], the latter being a connection-oriented, datagram-based transport protocol that provides in-order reliable data delivery and flow control. IQ-RUDP implements TCP-like congestion control using an algorithm resembling Loss-Delay Adjustment (LDA)[14]. In addition, it has many of the adaptation support features seen in existing adaptive transport protocols (see [5] for more detail).

2.1 Basic Elements of IQ-RUDP

IQ-RUDP has the following three mechanisms to support application-level adaptation: (1) it exposes certain network performance metrics[1], (2) it supports application-registered callbacks, and (3) it implements application-controlled adaptive reliability. Concerning (1), the application can query for a group of network performance metrics maintained by IQ-RUDP anytime during a connection's lifetime. This is useful for applications that require tight control over their data transmissions, such as interactive audio applications[1]. Concerning (2), an application can register callbacks to be triggered under certain conditions. Such callbacks can be used by self-clocked applications that are able to adjust their rates dynamically, such as streaming layered audio/video applications[1]. Finally, concerning (3), an application can adaptively change the desired reliability of different sub flows and then delegate to IQ-RUDP the realization of such changes.

IQ-RUDP is further distinguished from previous work by two facts. First, it uses *quality attributes*, which are lightweight middleware mechanisms for exchanging performance information between the application and transport layers. Second, its extensive adaptive reliability mechanisms permit both the sender and receiver to adaptively control their desired reliability.

IQ-RUDP provides adaptive reliability at the transport level, implemented within its window-based congestion control algorithm. Compared to previous work[6, 8], it supports *both* receiver loss tolerance and sender packet priority marking. In the remainder of this paper, the term RUDP is used to denote the basic reliable and adaptive transport functionality of IQ-RUDP, whereas the term IQ-RUDP refers to the coordination schemes needed for RUDP's interaction with application-level adaptations.

2.2 Using Quality Attributes

IQ-ECho's quality attributes[12] are the basic mechanism used for IQ-RUDP's application adaptation support. Each attribute is in the form of a `<name, value>` tuple. The registration, update and query of ECho attributes are implemented via a distributed service. Quality attributes are the means of exporting network performance metrics from IQ-RUDP to the application, and they can be used to inform the transport level about application level adaptations. Also, the application registers for call-backs from IQ-RUDP using attributes and specifies its reliability requirement as attributes. Attributes are usually carried either as parameters to IQ-RUDP's API for sending, *CMwritev_attr()*, or as an IQ-RUDP connection state variable shared by the application. For the library-based implementation of IQ-RUDP used in this paper, the costs of updating and querying attributes are negligible even when done frequently. For our future, kernel-level implementation of IQ-RUDP, the use and implementation of quality attributes will leverage our experience with Linux-based experiments that evaluate in detail the costs of realizing quality attributes and call-backs across the middleware-OS kernel interface[12].

2.3 Coordinating Application and Transport Adaptation

The point of this paper is to go beyond the support for transport-level adaptation provided by the basic elements of IQ-RUDP. As a result, for the large-data applications targeted by our work, IQ-RUDP uses quality attributes to *coordinate* transport- with application-level adaptations:

1. Both the transport and the application levels adapt their own behaviors, rather than having either one of them delegate control to the other.

2. IQ-RUDP coordinates the adaptations performed at the transport and application layers.

The remainder of this section describes the motivation and design of the coordination schemes used.

2.3.1 Motivation and Solution Approach

With the basic IQ-RUDP, as with other TCP-friendly transport protocols that support application adaptation, transport adaptation and application adaptation are only loosely coupled. That is, while the application does base its adaptations on information about network state received from the transport layer, neither it nor the transport knows about the actual adaptations they are each carrying out. This independence gives each layer flexibility in adaptation, simplifies the architecture, and reduces overheads. However, it also leads to the following question:

Need for Coordination

> *Should the transport's behavior be modified when the application starts to adapt to a network condition?*

There are several scenarios in which such re-adaptation based on the knowledge of the application's adaptation may be critical to an application's end-to-end performance:

Case 1: Conflicting Interests The transport and the application may have different reasons for adapting, such as considering application performance versus being fair to other network flows. In addition, many applications need to adapt to multiple contexts or coordinate several streams, both of which might conflict with transport adaptation. For instance, an application may adapt to sustained congestion by unmarking more packets but maintaining its frame rate, with the hope of trading reliability for the timeliness of important messages. This hope will not be realized if the transport adapts its congestion window to the available bandwidth.

Case 2: Over-reaction Since both the application and the transport may adapt to a change in network condition, the aggregate effect of their adaptations might be too strong. An example is a reduction of data resolution at the application level coupled with an IQ-RUDP reduction of its congestion window, both trying to match the available network bandwidth. The result is that the application uses less bandwidth than its fair share and experiences both worse quality video and longer delay/jitter. Over-reaction can also lead to performance oscillation and longer convergence time. In wide-area networks, and especially for bulk-data transfers, these impacts are exacerbated since it takes longer for the window size to match the connection's fair share of bandwidth.

IQ-RUDP has to be made aware of *how* application adaptation affects its traffic pattern in order to appropriately adjust its own behavior for the above two cases. The next cases concern the issue of *when* adaptations are performed.

Case 3: Limited Granularity Adaptation granularity refers to how often adaptations can be performed[12], as exemplified by a media application in which an adaptation involves changes in pixel resolution or quantization levels: the sender application cannot adapt until all packets belonging to the same frame or frame group have been sent.

The mismatch in granularity of application- and transport-level adaptations results in two issues, both of which may be addressed by giving IQ-RUDP information about when certain application-level adaptations have taken place and under what assumptions they were performed, using quality attributes as additional parameters to IQ-RUDP calls.

Case 3-1: Asynchronous Adaptation. First, if an application delays its execution of a triggered adaptation, then IQ-RUDP should be made aware of *when* the adaptation is actually performed, so that it can perform its own adaptation prior to that time and then re-adapt to the changed application behavior afterward.

Case 3-2: Obsolete Information. Second, due to delayed adaptation, an application may be using obsolete information for adaptation, such as outdated network status information. The solution is to *coordinate* across the transport and application levels, by (1) permitting the application to determine how to best adapt given its current knowledge of network condition and of its own communication behavior, and (2) permitting IQ-RUDP to re-adapt to adjust for differences in application-level assumptions about current network state vs. its own knowledge about such state. Quality attributes help with the implementation of this scheme, as the application will not only adapt what data it sends, but will also pass along with such data the assumptions it made about network state for that adaptation. This kind of coordination is particularly helpful to applications that do not want to be frequently interrupted for adaptation and thus only adapt on coarse-grained condition changes. With those applications, since changes in network condition can be rather significant without additional application adaptations being triggered, the ability of IQ-RUDP to consider the change in network over the pending period of an adaptation can improve the performance by a large extent.

Keys to the Solution If coordinated adaptation is desired, how and when should it take place? A solution to this problem has three components: (1) the entity to perform coordination, (2) the mechanism for coordination, and (3) the manner in which coordination is carried out.

The keys to the solution, corresponding to each of these components, are three observations. First, the *transport* should be the final point of regulation before data is sent onto the network and is therefore, a natural place for coordination. Second, the information necessary for coordination is comprised of *when and how* an application adaptation is performed and the *network conditions* on which the adaptation is based. This implies that there must be an information flow from the application to IQ-RUDP regarding its adaptation. Third, IQ-RUDP can be informed about application-level adaptations by mapping them to changes of IQ-RUDP's parameters, which permits IQ-RUDP to perform re-adaptation by *re-adjusting* those parameters.

2.3.2 Coordination Mechanism

The key to coordination of application- with transport-level adaptation is an application to transport flow of quality information. In order to be useful for IQ-RUDP's coordination activities, this information has to address the impact of the application adaptation on the traffic it passes onto IQ-RUDP. This can be described in terms of message frequency, size, and reliability requirements, which correspond to application-level changes of frequency, resolution, and reliability, respectively. With a frequency adaptation, the application sends the same amount of data as before in each message but less frequently. With a resolution adaptation, the application sends less data in each message with the previous frequency. Frequency adaptation and resolution adaptation have different implications on the IQ-RUDP window algorithm, as discussed in Section 3.4. A reliability adaptation does not lead to changes in IQ-RUDP's window algorithm, as shown in Section 3.3, but coordination is sometimes necessary in order to achieve the behavior intended by the application.

We use ECho attributes to communicate the three aspects of an application adaptation that are necessary for coordination: the impact on application traffic, the timing, and the network conditions based on which adaptations are performed. Attributes ADAPT_FREQ, ADAPT_MARK and ADAPT_PKTSIZE describe the degrees of adaptations of frequency, reliability, and resolution, respectively. The ADAPT_WHEN attribute indicates whether or when an application will adapt. The ADAPT_COND attribute describes the network conditions on which an adaptation is based, including the error ratio and the average data rate.

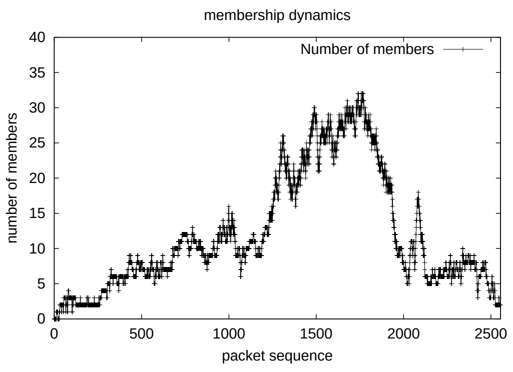

Figure 1: Membership Dynamics

3 Performance Evaluation

3.1 Setup

Experiments demonstrating the importance of coordinated adaptation are performed with the EMULAB test-bed[3], which is a network emulation environment that supports a wide variety of network topologies and link characteristics. All experiments are conducted on emulated 20Mb physical links with a path RTT of 30ms, unless otherwise noted, and a maximum RUDP segment size of 1400 bytes. The first experiment is a baseline comparison of IQ-RUDP and TCP, demonstrating IQ-RUDP's fairness and adaptation support feature. The remaining experiments each evaluate a different coordination scheme, using some application-level and some IQ-RUDP-level set of adaptations. To demonstrate the importance of coordination across these two levels, adaptations performed independently at each level without coordination (RUDP) are compared with adaptation performed in a coordinated fashion (IQ-RUDP).

For each case of coordination, experiments are performed for both the following two settings: 1) the application traffic pattern changes, 2) the available network bandwidth changes. In both settings, the condition that triggers the adaptation is network congestion level, or loss ratio as seen by the end system. In the first case, fluctuating application traffic pattern is implemented by having the application source send frames of different size at certain fixed frame rate. The changing pattern of frame size follows the MBone trace in Figure 1, which in one way emulates a content delivery server that uses multiple unicast streams to multicast. The frame size is the group size multiplied by 3000 bytes. In the second case, a variable bit rate UDP source is used as cross traffic while the application sends out fixed size data packets as fast as allowed by RUDP. The UDP source also has a fixed frame rate (500 frames/sec) and the frame size fluctuation follows the same MBone trace. The frame size is the group size multiplied by 2000. To congest the 20M link, we use the *iperf* [1] tool to generate UDP cross traffic at a fixed rate that differs across experiments, as noted.

3.2 IQ-RUDP Performance

We compare the performance of IQ-RUDP and RUDP in the following subsections. To make such comparisons meaningful in a global context, i.e., relative to other adaptive transport protocols, we will first compare the performance of IQ-RUDP and TCP, thereby demonstrating the effectiveness of application adaptation support in RUDP. To this end, we use the dynamically changing application setup described above coupled with 18M UDP cross-traffic. We run the application described above with different transport/adaptation schemes and measure the duration, throughput, packet inter-arrival and the jitter (deviation) of packet inter-arrival for each case. The application adaptation in the last two experiments is by changing resolution (frame size), the exact algorithm is the same as in section 3.4 and is to be described there. It is sufficient for now to keep in mind that the application reduces its frame size by a degree proportional to the loss ratio and increase the frame size at fixed rate when the loss is below a certain threshold.

In the experiment with application adaptation only, as shown in the third row in Table 3.2, we instrumented IQ-RUDP to disable its adaptive congestion window algorithm, but still provide performance metrics to the application, based on which the application can perform its own adaptations. IQ-RUDP with application adaptation represents the case where IQ-RUDP's congestion control is enabled as well as the application's adaptation. Results of 1 and 2 show that TCP and IQ-RUDP get approximately the same throughput, suggesting that their throughput is approximately the same given certain network condition. However, IQ-RUDP achieves better delay and jitter, due to its smoother changes of congestion window. With application adaptation (3 and 4), the application finishes faster and has a smaller average delay/jitter of packet interarrival. However, application adaptation only (3) has about 8% lower throughput than IQ-RUDP, while application adaptation with IQ-RUDP congestion control (4) reduces the difference to about 2%. The improvement of (4) over (3)

[1]visit http://dast.nlanr.net/Projects/Iperf/.

Transport Tested	Time	Throughput	Inter-arrival	Jitter
TCP(1)	313	94.2K	0.239	0.110
IQ-RUDP(2)	298	98.2K	0.201	0.098
App adaptation only(3)	158	90K	0.114	0.008
IQ-RUDP w/ app adaptation(4)	144	95.6K	0.113	0.058

Table 1: Basic performance comparison

derives from the fact that given the application's adaptation granularity (the thresholds it acts on), there can be a significant mismatch between its rate and the available bandwidth, while IQ-RUDP can catch very fine-grained changes in loss ratio and adapt instantly. The fact that (4) does not achieve the same throughput as IQ-RUDP alone is explained by the over-reaction phenomenon, discussed in more detail in Section 3.4.

To test the fairness of TCP and IQ-RUDP, we change the cross-traffic to a TCP cross-traffic and run the same application (without adaptation). Results in Table 2 show that the throughput of TCP and IQ-RUDP are close, with TCP achieving somewhat higher throughput than IQ-RUDP. We observe that although the average rate of increase is the same for both protocols, TCP adapts at a finer granularity and might take over more bandwidth than the competing IQ-RUDP flow initially. This might make the IQ-RUDP flow converge to a throughput lower than its fair share, but we do not expect this to be the case when there is a sufficient degree of multiplexing on the path.

3.3 Conflicting Interests

This experiment demonstrates how coordination can help resolve conflicts between application- and transport-level adaptations. Consider, for instance, a remote visualization of large data in which some of the data being sent is outside the area on which the end user is currently focusing. The application, then, can use this information to transmit data on which an end user is not currently focusing unreliably, thereby gaining communication bandwidth for essential data transfers. To realize this, the application registers a pair of call-backs to be triggered when the RUDP error ratio exceeds 30% (upper threshold) or drops below 5% (lower threshold). When the upper threshold call-back is triggered with 'error ratio' *eratio*, all application datagrams sent afterward will be either marked or unmarked using the following algorithm: (1) there is a tagged packet every five packets; this represents control information that has to be sent to ensure correct data display; (2) for all other packets, there is a probability of *max(40, (5/4)*eratio)*[2] of being unmarked and therefore, not requiring delivery; this represents raw data of which some fraction may be lost.

In the lower threshold call-back, the unmarking probability is reduced by 20%. In this experiment, the receiver loss tolerance is set to 40%, and a 10Mb *iperf* cross traffic is used. It should be noted that we choose relatively large and coarse-grained loss ratios as adaptation thresholds for the following two reasons: (1) given that we use a controlled test environment where the only competing flow is an aggressive UDP flow, we might not see the statistical nature of losses resulting from multiplexing in an actual packet-switched network, so that an RUDP flow might suffer a high loss ratio within a measuring period; (2) the application being tested displays constant and very fast changes in rate, as can be seen from the trace in Figure 1, and those changes can result in non-negligible losses in consecutive measuring periods, for which a low adaptation threshold will result in overly frequent application adaptations.

With IQ-RUDP, after an application indicates, through a call-back return value, that it adapts by unmarking some packets, RUDP starts to discard unmarked datagrams before sending them onto the network. Without IQ-RUDP, RUDP continues to send *all* application packets at a rate allowed by its congestion window. This means that the application still incurs large delay/jitter for tagged packets, although more untagged packets will be delivered. Notice that this is usually what a TCP-like adaptive transport protocol without coordination will do, even if it does support variable reliability.

Here, the interesting metrics are: (1) time to finish (duration), (2) average inter-arrival of tagged messages (delay), (3) deviation of the inter-arrival of tagged messages (jitter), and (4) the percent messages delivered. The results in Table 4 show that IQ-RUDP reduces the application run time and improves the delay/jitter of tagged packets substantially (about 25%), while the percent of packets undelivered is still within the loss tolerance, although higher than that of RUDP. The average delay/jitter across all pack-

[2]Untagged packets are unmarked with a probability higher than *eratio* so that the overall ratio of unmarked packets is approximately *eratio*.

Transport Tested	Time	Throughput	Inter-arrival	Jitter
TCP	51	118K	0.022	0.0001
IQ-RUDP	60	99K	0.024	0.0001

Table 2: Fairness test

	Duration(sec)	Mesgs Recvd(%)	Tagged Delay(msec)	Tagged Jitter	Delay(msec)	Jitter
IQ-RUDP	60.0	72	58.4	6.6	56.4	6.6
RUDP	80.9	91	66.8	9.1	62.2	7.9

Table 3: Coordination against conflict–changing application

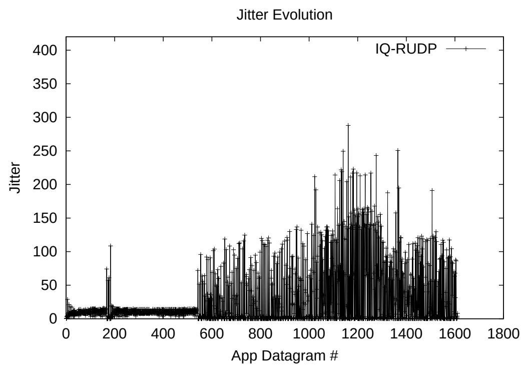

Figure 2: Delay jitter–IQ-RUDP

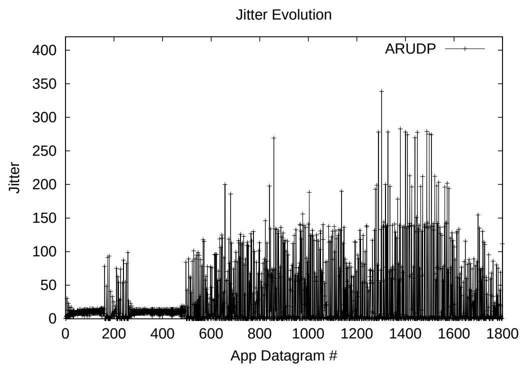

Figure 3: Delay jitter–RUDP

ets is not improved a lot, because with both schemes, the sending window is adapted to the congestion level of the network. But the delay/jitter of tagged packets is improved for tagged packets because many untagged packets in between are not sent onto the network. Also note that the average delay over all packets is smaller than that of tagged packets because tagged packets are spaced out.

Figures 2 and 3 present the different jitters experienced by the receiver application with the two schemes. The sharp increase around the 500th packets reflects the point when the cross traffic starts to have adverse impact on the application. IQ-RUDP results in jitter that is more stable and lower on the average. However, we notice that even with IQ-RUDP, the jitter fluctuates over a congestion window worth of packets. Due to the window-based nature of the IQ-RUDP, a congestion window worth of packets are sent back-to-back under normal operation. They will experience different queuing delays at the bottleneck.

3.4 Over-reaction

coordinating IQ-RUDP improves throughput, latency, delay/jitter. Also, experiments under different levels of network congestion suggest that the improvement over uncoordinating RUDP is pronounced under severe congestions. The following test evaluates the importance of coordination to prevent over-reaction: the application registers callbacks for a pair of error ratio thresholds (upper 15%, lower 1%). The application adaptation instantly reduces packet size by a percentage equal to the error ratio when the upper threshold is exceeded, and increases packet size by 10% when the lower threshold is hit. This experiment emulates an application that tries to achieve shorter duration and delay/jitter, as in Section 3.3, not by permitting packet loss, but instead, by sacrificing data resolution through downsampling.

With IQ-RUDP, after an application indicates an adaptation that reduces frame size by $rate_{chg}$, RUDP increases window size (in units of RUDP packets) to $1/(1-rate_{chg})$ of the current value if the current application frame is smaller than the maximum RUDP segment size. The effect is that the actual bit rate is adapted to the connection's

	Duration(sec)	Mesgs Recvd(%)	Tagged Delay(msec)	Tagged Jitter	Delay(msec)	Jitter
IQ-RUDP	23.9	63	30.2	3.1	29.6	3.1
RUDP	32.5	87.4	38.1	4.3	29.4	3.8

Table 4: Coordination against conflict–changing network

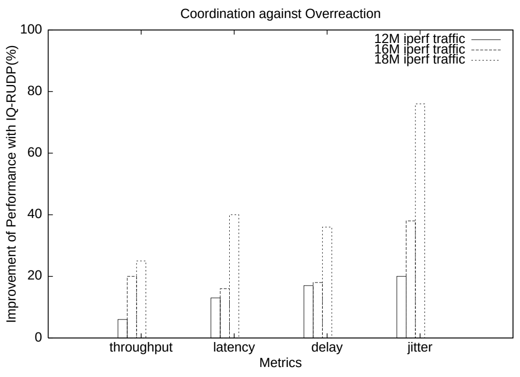

Figure 4: Performance improvement–over-reaction

fair share of bandwidth. With RUDP, the reductions of the application's frame size and RUDP's congestion window jointly contribute to a bit rate that is lower than the available bandwidth.

Notice that for a frequency adaptation, IQ-RUDP does not have to increase the window size since the reduction of application frame frequency has the same effect.

Results in Tables 5 and 6 and the graphical depiction of the results in 6 in Figure 4 show that performance is improved by a large extent with IQ-RUDP. The improvement with different rates of *iperf* background traffic (the VBR UDP cross traffic remains the same across these experiments) in the changing network experiment suggests that coordination is more helpful when the congestion gets more severe. We see an increase in throughput ranging from 6% to 25% and a reduction of jitter ranging from 20% to 76%, for increasing levels of network congestion.

3.5 Limited Granularity

This set of experiments evaluates the extent to which application performance can be improved via coordination that addresses the limited granularity of application adaptation. Here, the application registers the same pair of call-backs as in Section 3.4, but it can only start to adapt at the next application frame with a sequence number divisible by 20, thereby emulating application-level packets of large size. We perform three experiments, each to test a different adaptation support scheme. Assuming an error ratio *eratio*, the upper threshold adaptation and coordination in the three experiments differ in the following way:
(1) RUDP: RUDP starts adaptation on its own after the call-back returns *void*; the application later reduces its frame size by $rate_{chg}$, which equals *eratio* in these experiments;
(2) IQ-RUDP without ADAPT_COND: the call-back returns an attribute to indicate that the adaptation will be delayed. RUDP, in turn, adapts on its own until the application starts to reduce frame size by $rate_{chg}$ the same way as in (1). This change is indicated by an attribute parameter in the application's next call to *CMwritev_attr()*, when it immediately increases its window size to $1/(1 - rate_{chg})$ of the current value;
(3) IQ-RUDP with ADAPT_COND: the IQ-RUDP coordination in (2) is enhanced in the following ways: (a) An additional attribute ADAPT_COND is used in the attribute list parameter to *CMwritev_attr()* to describe the error ratio $eratio$ when the application adaptation is triggered. (b) Assuming that network congestion has changed to $eratio_{new}$ during the delay of the application adaptation, after receiving the information of the delayed adaptation, RUDP changes its window to

$$\frac{(1 - eratio_{new})/(1 - eratio)}{1/(1 - rate_{chg})} \tag{1}$$

where $eratio$ is provided by the application. This change accounts for the network change during the application's delay of adaptation. Notice that this modification to (2) usually only changes the behavior of the changing network test, since for the changing application test, $eratio$ usually does not change a lot. Hence, we test scheme (3) only for the changing network scenario.

Similar processing is performed when a lower threshold call-back is triggered, except that the application increases frame size by 10% in each call. Compared to RUDP, which adapts based on its measurement only, we expect to see better performance in IQ-RUDP with its immediate change of the sending window, especially when the round-trip time is relatively large (see Section 2.3.1). Furthermore, by considering the changes of network conditions for a delayed adaptation, we expect better performance of IQ-RUDP with ADAPT_COND than without.

	Throughput(KB/sec)	Duration(sec)	Delay(msec)	Jitter
IQ-RUDP	380	39	10.4	0.78
RUDP	367	42	15.2	0.83

Table 5: Coordination against overreaction–changing app

iperf traffic	Transport Tested	Throughput(KB/sec)	Duration(sec)	Delay(msec)	Jitter
12Mbps	IQ-RUDP	506	9.5	3.8	0.20
	RUDP	478	10.9	4.6	0.25
16Mbps	IQ-RUDP	131	26.1	10.2	6.4
	RUDP	109	31.0	12.4	10.3
18Mbps	IQ-RUDP	99	51	14	19
	RUDP	79	85	22	80

Table 6: Coordination against overreaction–changing network

Thus, we expect to see increased performance in the order of methods (1), (2), and (3) above. To understand the impact of RTT on the possible performance differences, we deliberately use different physical path conditions for the changing application and changing network tests. In the first case, we use the same network setup as that used in the previous experiments. In the latter case, we use a path with 125ms one-way delay. For this much larger RTT and hence much lower achievable throughput, we change the application such that it has a fixed frame rate instead of sending at its fastest possible rate. Further, the application adapts in order to achieve better throughput while meeting the deadlines of each packet and of the whole task. A 14Mbps *iperf* cross traffic is used in the latter experiment.

Results in Tables 7 and 8 show that IQ-RUDP outperforms RUDP in both cases. The performance differences between them, though, are less noticeable in Table 7 than in Table 8, due to the much larger RTT in the latter experiment (see section 2.3.1). It should also be noted that the metrics that are most improved by IQ-RUDP differ in the two experiments. Improved in the changing application test is delay/jitter, while it is throughput in the other test, which is just as expected from the goals of application adaptation mentioned previously.

In the experiment with a changing network, the performances of the three schemes in the changing network scenario are exactly as expected. IQ-RUDP with ADAPT_COND is able to obtain a throughput improvement over RUDP of about 18%, comparable to that achieved in Section 3.4, virtually eliminating the effect of the limited application adaptation granularity. While improvements in delay and duration are less obvious since this application is rate-based, jitter is improved significantly (about 38% less). However, without ADAPT_COND, the IQ-RUDP improvement over RUDP is much less significant in these experiments than in those in Section 3.4. This is contrary to our expectation since a longer RTT is used in these experiments. An explanation of this fact is the use of obsolete network information, which is exactly what is addressed by scheme (3).

4 Concluding Remarks

This paper presents results that demonstrate the importance of coordinating transport- with application-level runtime configuration for large-data grid applications. Using the IQ-ECho middleware to implement data streams for remote visualizations across wide area networks, we evaluate adaptive behavior in which application-level adaptations that downsample data or that mark it for potential loss interact with transport-level adaptations that change the rates at which data is sent. Substantial performance improvements are attained by coordinating the adaptations performed at both levels via an active transport layer. Activity and adaptation at the transport layer are implemented by the IQ-RUDP protocol specialized for large-data transfers across wide area networks. Coordination across the transport and application layers is implemented using quality attributes, which are lightweight name-value pairs coupled that carry information across the application-system boundary and also provide the asynchronous notification support needed to enable adaptations to proceed at their at each level at their own speeds.

Our future work concerns both specific extensions to the results presented here and the application of IQ-ECho and IQ-RUDP to the large-scale real-time collaborations now being initiated in the U.S. One specific example under study is the Supernova Initiative in which a large number of

Transport Under Test	Results			
	Duration(sec)	Throughput(KB/sec)	Delay(msec)	Jitter
IQ-RUDP w/o ADAPT_COND	140	97	0.097	0.047
RUDP	144	95.6	0.113	0.058

Table 7: Limited application adaptation granularity–changing app

Transport Under Test	Results			
	Duration(sec)	Throughput(KB/sec)	Delay(msec)	Jitter
IQ-RUDP w/ ADAPT_COND	22.1	37.8	6.5	0.8
IQ-RUDP w/o ADAPT_COND	22.7	33.8	6.7	1.1
RUDP	23.2	32.0	6.8	1.3

Table 8: Limited application adaptation granularity–changing network

DOE and university researchers are collaborating to model and evaluate the physical and nuclear processes ongoing in supernovae. Other examples studied concern Grid-FTP, for which we are currently developing the IQ-FTP implementation for selectively lossy file transfers: end users can dynamically select (with user-provided functions) the most critical file contents to be transferred to their local sites.

References

[1] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System Support for Bandwidth Management and Context Adaptation in Internet Applications. In *SOSP*, Oct 2000.

[2] T. Bova and T. Krivoruchka. Reliable UDP Protocol(draft-ietf-sigtran-reliable-udp-00.txt), 2000.

[3] J. Lepreau et. al at the Utah University. The Utah Network Testbed. http://www.emulab.net/.

[4] M. Fisk and W. Feng. Dynamic Right-Sizing in TCP. In *ICCCN*, 2001.

[5] Q. He and K. Schwan. Adaptive Reliable UDP. March 2002.

[6] R. Kravets, K. Calvert, P. Krishnan, and K. Schwan. Adaptive Variation of Reliability. In *HPN*. IEEE, Apr 1997.

[7] R. Kravets, K. Calvert, and K. Schwan. Payoff Adaptation of Communication for Distributed Interactive Applications. *Journal of High Speed Networks*, Jul 1998.

[8] J. Li, D. Dwyer, and V. Bharghavan. A Transport Protocol for Heterogeneous Packet Flows. In *INFOCOM*, Mar 1999.

[9] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control, Jan 1997.

[10] NASA. Using XML and Java for telescope and instrumentation control. In *SPIE Advanced Telescope and Instrumentation Control Software*, 2000.

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

[12] C. Poellabauer, K. Schwan, and R. West. Lightweight Kernel/User Communication for Real-Time and Multimedia Applications. In *NOSSDAV*, Jun 2001.

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

[14] D. Sisalem and H. Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. In *NOSSDAV*, Jul 1998.

[15] H. Sivakumar. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *SuperComputing*, 2000.

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

[17] G. Wong, M. Hiltunen, and R. Schlichting. A configurable and extensible transport protocol. In *IEEE Infocom*, Apr 2001.