IQ-Paths: Self-regulating Data Streams across Network Overlays

Zhongtang Cai, Vibhore Kumar, Karsten Schwan
{ztcai, vibhore, schwan}@cc.gatech.edu
College of Computing
Georgia Institute of Technology

***Abstract*— Overlay networks have been shown useful for improving the delivery of network and processing resources to applications, in part due to their ability to use alternate or parallel network paths and computational resources. This paper presents IQ-Paths, a set of techniques and their middleware realization that implement self-regulating data streams for data-intensive distributed applications. Self-regulation is based on (1) the dynamic and continuous assessment of the quality of each overlay path, (2) the use of online network monitoring and statistical analyses that provide probabilistic guarantees about available path bandwidth, loss rate, and RTT, and (3) a packet routing and scheduling algorithm that dynamically schedules data packets to different overlay paths in accordance with their available bandwidths. Additional aspects of IQ-Paths are its predictive statistical bandwidth guarantees and the fact that packet scheduling across different overlay paths is governed by application-level specifications of stream utility. An example is to send control data across links that offer strong guarantees for future bandwidth vs. mapping other data across less guaranteed paths. Experimental results presented in this paper use IQ-Paths to better handle the different kinds of data produced by (1) distributed multimedia applications with desired QoS guarantees and (2) data-driven or interactive high performance codes with user-defined utility requirements.**

I. Introduction

Data-driven distributed applications are important to many constituencies, including corporations in applications like real-time data mining or data integration [4,28], common end users in telepresence [27], and scientists or engineers in applications like remote data visualization [45] or instrument access [32]. A common characteristic of such applications is their need to meet quality of service (QoS) guarantees and/or offer utility-based services to end users (i.e., meet certain service-level objectives (SLOs)). However, excepting datacenter-based solutions [4] and the few dedicated, high end links existing between select centers of excellence (e.g., via DOE's UltraScience Net [10] or the National Lambda Rail [24] (NLR)), such guarantees must be provided across shared network infrastructures, where dynamic network behavior and multiple available network paths make it imperative for middleware to assist end user applications in best utilizing available network resources. More specifically, when transporting and manipulating their data, applications should receive utility-based guarantees from the overlay networks used by middleware, accommodating dynamic variations in network behavior: (1) media or other real-time applications require consistent levels of end-to-end performance, such as limited delays or small jitter [11], (2) enterprise applications couple data transport and manipulation with application-level expressions of utility or cost [18, 23], and (3) both application classes can utilize guarantees that differentiate across different traffic types, such as offering stronger guarantees for control vs. data traffic [35].

Previous work on middleware for data-intensive distributed applications has addressed limitations and runtime variations in network bandwidth with adaptive approaches to matching desired to available network resources. Examples include dynamically adjusting data transfer rates [44, 47], varying compression levels in response to monitored changes in network bandwidth [20, 48], or changing the nature of the data being sent [7, 19, 49]. Other research has sought to use alternative network connections or new network infrastructures to compensate for problematic connection behaviors [5,34,36].

This paper presents the IQ-Paths approach to self-regulating data streaming with defined quality requirements across wide area networks. IQ-Paths offers novel functionality that enhances and complements existing adaptive data streaming techniques. First, IQ-Paths dynamically measures [17, 29] and then, also predicts the available bandwidth profiles on network links. Second, it extends such online monitoring and prediction to the multi-link paths in the overlay networks used by modern applications and middleware. Third, it offers automated methods for moving data traffic across overlay paths. These include splitting a single data stream across multiple paths to improve performance through concurrency and to improve desired end-to-end behavior by dynamically differentiating the amounts and kinds of data traffic imposed onto different paths. Such self-regulating data movement and differentiation utilizes a dynamic packet scheduling algorithm that automatically maps packets to paths to match application-level utility specifications. Finally, an important attribute of IQ-Paths is that unlike other methods for bandwidth prediction based on measurements of average bandwidth, it uses statistical techniques to capture the dynamic or noisy nature of available network bandwidth across overlay paths. This enables it to better map data with different desired utility – service guarantees – to the underlying best effort network infrastructure.

Our research uses IQ-Paths for both scientific and multimedia applications. In the scientific domain, real-time remote data visualization for a molecular dynamics (MD) code benefits from IQ-Paths' ability to better meet its dynamic end

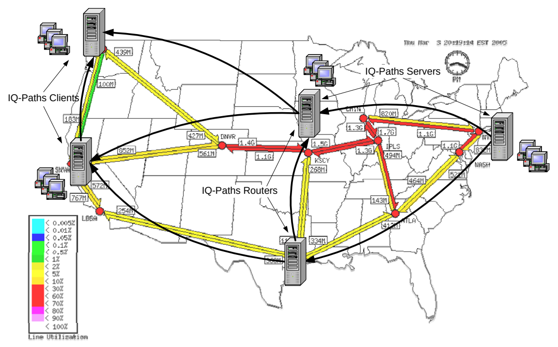


Figure 1: IQ-Paths overlay network: servers, routers, and clients continually assess the qualities of their logical links, admit and map data streams with different desired utility using a self-regulating packet routing and scheduling algorithm.

user requirements. A specific example is to differentiate the transport of various elements of the application's data streams, in this case being the atoms vs. bond forces visually depicted for each timestep of the MD application. Another example is to use network paths with more stable bandwidths for the critical 'control' traffic in the remote visualization software and also for the most time-sensitive data sets in large volume parallel data transfers. Stability is dynamically diagnosed and predicted via the aforementioned statistical techniques. In the multimedia domain, IQ-Paths is shown to deliver improved performance for different encoding levels of MPEG-4 video streams.

Results in Section VI also demonstrate the advantages derived from IQ-Paths' statistical guarantees. Specifically, we demonstrate distinct improvements over earlier work on adaptive methods that provide QoS over wide-area networks by predicting future average network behavior from past history [26]. With such methods, quantities like RTT can be predicted well, but average available bandwidth or packet loss rate are not easily captured (e.g., using predictors like MA, AR, or more elaborate methods like ARMA and ARIMA) [50]). This is because noise is a large portion of the signal in the time series of available bandwidth or packet loss rate. As a result, the values for predicted average bandwidths will have large prediction errors. For example, the results reported in [50], based on measurements at over 49 well-connected academic and research institutions, have prediction errors larger than 20% for more than 40% of the predicted values (i.e., $|predictedvalue/actualvalue| > 1.2$), and for 10% of the values, prediction error is larger than 50%. In comparison, IQ-Paths can provide an application with strong guarantees, stating that it will receive its required bandwidth 99% of the time or experience a deadline miss rate (i.e., jitter) of less than 0.1%, for example. Finally, other methods apply low frequency filters [8] to measured values, to reduce prediction error, but unfortunately, this means that they essentially eliminate the noisy nature of (i.e., dynamic variations experienced over) certain network paths. The outcome is that applications cannot adjust to or deal with such variations, by mapping less important or less delay-sensitive data to noisier connections, for example.

Figure 1 illustrates an example of an IQ-Paths overlay, which utilizes automatic network resource profiling, admission control, and self-regulating data routing and scheduling to guarantee different streams' desired utility requirements. The overlay implemented by IQ-Paths has multiple layers of abstraction. First, its *middleware underlay* – a middleware extension of the network underlay proposed in [31]) – implements the execution layer for overlay services. The underlay is comprised of processes running on the machines available to IQ-paths, connected by logical links and/or via intermediate processes acting as router nodes. Second, underlay nodes continually assess the qualities of their logical links as well as the available resources of the machines on which they reside. The service guarantees provided to applications are based on such dynamic resource measurements, on runtime admission control, resource mapping, and on a self-regulating packet routing and scheduling algorithm. This algorithm, termed PGOS (Predictive Guarantee Overlay Scheduling), provides probabilistic guarantees for the available bandwidth, packet loss rate, and RTT attainable across the best-effort network links in the underlay.

Key technical advantages of IQ-Paths and its PGOS algorithm include the following:

- *Probabilistic and 'violation bound' guarantees*: since the PGOS algorithm uses bandwidth distribution analysis and prediction to capture network dynamics, it can make service guarantees and provide prediction accuracies superior to those provided by prediction methods based on average network behavior: (1) it can ensure that applications receive the bandwidths they require with high levels of assurance (e.g., it can guarantee that an application receives its required bandwidth 99% of the time or that its deadline miss rate is less than 0.1%)); (2) in addition, PGOS can also provide deadline violation guarantees that bound the average number of packets that miss their guaranteed QoS (e.g., their deadlines).
- *Reduced jitter*: by reducing jitter in applications like remote data acquisition or display, buffering needs are reduced. This is particularly important for high volume data transfers in time- or delay-sensitive applications.
- *Differentiated streaming services*: different streams can receive different levels of guarantees. As a result, when applications use close to the total available bandwidths of all overlay paths, PGOS can ensure that high utility streams receive stronger service guarantees that others.
- *Full bandwidth utilization*: providing guarantees does not imply sacrificing the bandwidths available to applications (e.g., by purposely under-utilizing some link). Instead, PGOS has sufficiently low runtime overheads to satisfy the needs of even high bandwidth wide area network links.

The remainder of this paper is organized as follows. The next section describes related work, to provide a better perspective on the technical contributions of the IQ-Paths approach. We then outline the software architecture of IQ-Paths, followed by descriptions of its bandwidth prediction methods and of the PGOS algorithm using these methods. Experimental evaluations on an emulated network testbed appear before the paper's conclusions.

II. Related Work

While the PGOS packet scheduling algorithm is inspired by the DWCS packet scheduling algorithm described in [47], its use for efficient multimedia data streaming across the Internet leverages substantial prior work on improving the quality of network video streaming [19, 37, 38]. Here, early work established the utility of adding and dropping different encoding layers of video streams for longer term coarse-grain stream adaptation [44]. Improvements like those in [37] also use a TCP-friendly control mechanism to react to congestion on shorter timescales, with mismatches between the two timescales absorbed by buffering at the receiver. The specific control mechanisms we use for multimedia data streaming are based on the work described in [19], which proposes an adaptive layered video streaming algorithm for MPEG-4 with limited buffer size. Priorities are used in the VOP (video object plane) to select or discard each VOP element based on average bandwidth prediction, to control the fashion in which fine-grain scalable coding allocates bandwidth to different encoding layers. In contrast, IQ-Paths uses statistical bandwidth measurement and prediction to capture network link qualities, and its PGOS self-regulating data routing and scheduling algorithm can utilize both multiple or alternate overlay paths to satisfy different video layers' utility requirements. The outcome is improved smoothness of video playback, despite the variable-bit-rate nature of layered video. The additional techniques described in [11] can be used to further smooth such variable-bit-rates, thereby attaining a constant transfer rate for each time interval in the transmission process.

A useful extension of our research for video streaming in peer-to-peer networks might use the techniques described in [33], which suggests the use of 'Hill-building' algorithms to deal with source disconnection and with substantial changes in client download rates. While earlier video portions are being played back, these on-line algorithms continuously pre-fetch video in small variable quality 'chunks' to best use currently available bandwidth, minimizing the sum of the squares of the number of layers not used in video playback.

OverQoS [42] describes the general idea of using overlays and admission control to deliver video across the Internet. OverQoS uses a Controlled-Loss Virtual Link (CLVL) abstraction to bound the loss rate observed by a traffic aggregate. Performance gains are achieved by FEC (Forward Error Correction) and conditional packet retransmission in the form of ARQs (Automatic Repeat reQuests). In each CLVL, bandwidth less than the total available bandwidth can be achieved for a subset of the OverQoS flows, with high probability, but potentially at the expense of other flows. In contrast, the PGOS algorithm controls path usage with a more general link abstraction that is able to provide statistical guarantees for both single and multiple streams across both single and multiple paths across the overlay.

Both IQ-Paths and OverQos assume that overlay routing nodes can be placed such that the paths between different pairs of routing nodes do not share common bottlenecks. In practice, such placements require knowledge of the network, by using methods of detecting shared congestion across flows [39], or by using more direct ways of detecting network topologies [40]. A general way to implement information exchanges between middleware and networks is described in [31], with a design of a network underlay that extracts and aggregates topology information from the underlying Internet. Overlay networks query the underlay when making application-specific routing decisions. The current implementation of IQ-Paths could take advantage of underlays, with network monitoring functions embedded in the communication layers of the middleware providing inputs to overlay construction and to its dynamic management. In addition, we could utilize the results of recent work on a 'map of the Internet' described in [41], which annotates it with properties that include connectivity, geography, routing policy, patterns of loss, congestion, failure and growth, etc.

Our general approach of using overlay networks to adapt to network dynamics is shown feasible in [8], which compares the performance of an End System Multicast architecture to that of IP Multicast. The paper also notes that noisy link measurements coupled with aggressive adaptation can cause overlay instability, while conservative adaptations may experience low performance. The proposed solution is to use exponential smoothing to capture the long term performance of a link, thereby distinguishing persistent from temporary changes. Our approach differs in that it exploits knowledge about noise rather than suppressing it, for example, by mapping critical data flows to less noisy links.

The ability of overlay networks to provide differentiated data delivery services requires certain levels of independence in underlying network links' packet losses, changes in bandwidth, etc. For the Internet, [2] shows that there is a reasonable degree of loss and failure independence across different links. Measurements on Planetlab [9] and our own analyses of Planetlab active probing trace and NLANR passive measurement traces show that there are reasonable degrees of bandwidth independence for different Internet links.

A basic contribution of the PGOS algorithm is its ability to predict future network behavior. [50] points out the difficulty of predicting bandwidth in wide area networks, studying the likelihood of observed bandwidth remaining in a region for which the ratio between the maximum and minimum observed values is less than a factor of ρ. We adopt a similar approach, assuming that it is difficult to predict the exact value of throughput in the next time interval (e.g., in the next second) and instead, providing statistical guarantees for predicting the distribution of throughput in the near future. Interestingly, as shown in [34], it is easier to make guarantees about RTT. Finally, we also leverage the substantial research on measuring available bandwidth described in [17,29]. Of specific relevance to this paper is recent work presenting more accurate metrics and algorithms to measure the variation of end-to-end available bandwidth [30].

III. Software Architecture of the IQ-Paths Middleware

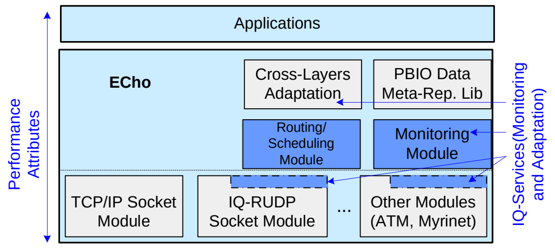


Figure 2: Middleware Architecture.

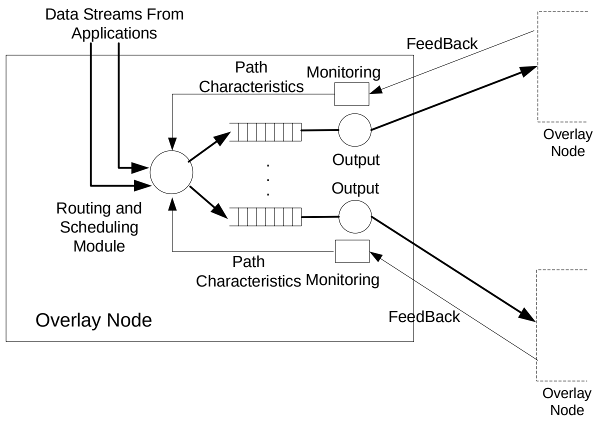


Figure 3: Structure of IQ-Paths Overlay Node.

The software architecture of the IQ-Paths middleware is depicted in Figure 2. It is derived from our substantial experiences with the IQ-ECho [6, 7, 15] high performance publish/subscribe infrastructure implementing channel-based information subscriptions. IQ-Paths leverages IQ-ECho's support for multiple transport protocols (e.g., TCP, RUDP, SCTP) and its monitoring modules for measuring desired network metrics from middleware and in cooperation with certain transport modules (e.g., RUDP). PGOS routing/scheduling module aggregates such runtime measurements in order to schedule application packets across multiple overlay paths. Unlike ECho, however, IQ-Paths is realized at a layer 'below' the publish/subscribe model of communication. Namely, IQ-Paths manipulates arbitrary application-level messages flowing from data sources to data sinks. Whether such messages are described as pub/sub events or in other forms is immaterial to the research described here. Similarly, IQ-Paths is not concerned with how source-to-sink links are established. It supports both direct source-to-sink links and more complex linkages that utilize overlay networks to route messages and process them 'in-flight' on their paths from sources to sinks. One way for end users to establish such linkages is via IQ-ECho's 'derived channel' abstraction [15]. Another way is to use the deployment features implemented as part of the 'in-transit' information flow infrastructure described in [21]. A third way is to directly use IQ-Paths as the transport layer for applications, as with the IQ^{PG}-GridFTP implementation used in the evaluation section of this paper.

The goal of IQ-Paths is to provide a general framework for routing, scheduling, and processing streams of application-level messages. Generality is established by layering IQ-Paths 'beneath' the different messaging models used by end users, including the IQ-ECho and in-transit models developed in our own research. A specific example is the IQ^{PG}-GridFTP described in this paper, which (1) replaces its transport level with IQ-Paths and (2) interposes the IQ-Paths message routing and scheduling algorithm between GridFTP's parallel link layer and lower level message transports. As a result, IQ-GridFTP (1) retains its ability to exploit parallelism in data transport by simultaneously using multiple network links, while more importantly, (2) gaining the ability to adjust the volumes of data being transferred to the current behavior of each single network link between source and sink, and (3) using overlay paths and path bandwidth-sensitive message routing and scheduling to better control how data is streamed across multiple links from source to sink.

Important components of the IQ-Paths middleware described in this paper are its Statistical Monitoring techniques and its Routing/Scheduling algorithms. Figure 3 illustrates the structure of each IQ-Paths overlay node and the dynamic interactions of these software components. Specifically, the Statistical Monitoring component monitors the bandwidth characteristics (i.e., bandwidth distribution) of each overlay path and shares this information with the Routing/Scheduling component. The latter routes applications' data streams and sub-streams to the appropriate overlay paths and in addition, for each path, it schedules the data packets mapped to it. The goal, of course, is to route and schedule application-level messages to continuously match the network loads imposed by the middleware to the available network bandwidths present in overlay paths, such that application-level metrics of stream utility are met (e.g., probabilistic guarantees on the timeliness of data delivery).

The remainder of this paper ignores other components of the IQ-Paths middleware, referring the reader to a more complete description of the system in [7]. We next describe the manner in which bandwidth guarantees are attained.

IV. Statistical Bandwidth Prediction and Guarantees

The PGOS algorithm presented in Section V provides to an end users predictive guarantees that with some large probability, application-level messages will experience certain levels of bandwidth across certain overlay paths. Toward this end, for each overlay path, IQ-Paths network monitoring (1) tracks the past distribution of path bandwidth, and (2) uses the percentile points in that distribution as the bandwidth predictor. The PGOS algorithm then uses these predictions to judiciously map application-level messages across overlay paths. This section explains (1) and (2) in more detail. The validity of our approach is demonstrated in Section VI-A, which shows that the efficacy of the PGOS statistical bandwidth predictor is much higher than that of predictors for average bandwidth.

Comparison of average with statistical bandwidth prediction*:* For each specific overlay path, frequent bandwidth variation makes it difficult to predict the exact values of average available bandwidth in the near future, both for very short timescales like milliseconds and for the second timescales at which IQ-Paths operates. The idea of statistical prediction is to leverage rather than suppress such variations, in order to provide to applications probabilistic bandwidth guarantees like the following: for some large value of P_0, we can find the value of bw_0, such that the probability $P(bw \geq bw_0) \geq P_0$. While predicting the exact value of future bandwidth is hard, statistical prediction relaxes the prediction requirement by asking if we can obtain certain amount of bandwidth with high probability. Because of the IID nature of available bandwidth, statistial prediction has much smaller prediction error than average bandwidth prediction. Furthermore, statistical prediction also captures the service-level objectives of many applications, including multimedia and scientific codes, answering questions like: 'can some stream obtain specific amounts of bandwidth most of the time?'

Another useful attribute of statistical prediction is its ability to retain certain information of potential value to end user applications. Specifically, when predicting average bandwidth, using a larger timescale for average time may reduce prediction error in some cases, since this effectively applies a smoothing filter to the available bandwidth time series. Statistical bandwidth prediction, in contrast, retains information about bandwidth variation, so that end users can be told, for instance, that some non-congested path can provide stable available bandwidth, whereas another path provides the same level of bandwidth with much less stability (i.e., with higher potential losses). IQ-Paths recognizes this beneficial property of statistical vs. average bandwidth prediction, exploiting it to better map different application-level data streams across overlay paths with stable vs. less stable bandwidth properties. A concrete example is to use a stable path for critical data in remote program steering [43] and a less stable path for additional information about the state of the remote application being steered.

V. The PGOS Overlay Path Guarantee and Scheduling/Routing Algorithm

This section describes the Predictive Guarantee Overlay Scheduling (PGOS) algorithm, first discussing the general algorithm framework, then clarifying the concept of predictive guarantees and describing the algorithm itself.

A. General Framework

An overlay network like the one in Figure 1 may be represented as a graph $G = (V, E)$ with n overlay nodes and m edges. An overlay node may be a server (i.e., data source) running on some host, a client (i.e., data sink), or a daemon for data routing. There may exist multiple distinct paths $P^j, j = 1, 2, ...L$, between each server and client, where $P^j = (V^j, E^j)$, $V^j = \{v_0, v_1, ...v_k, v_p \neq v_q \text{ if } p \neq q\}$ and $E^j = \{v_0v_1, ..., v_{k-1}v_k$, where $v_pv_{p+1} \in E$, for all $0 \leq p \leq k-1\}$. As in [42], we make no assumptions about the placement of overlay nodes in the network. Rather, we assume that the middleware has determined some suitable placement [22].

For each overlay link, since network bandwidth varies over time, the service time of each application-level message is not known a priori and varies over time. The specific problem addressed by the PGOS algorithm is further illustrated in Figure 4, where multiple streams $S^j, j = 1, 2, ...N$ must be transmitted from Server s to Client c with 'best' predictive performance guarantees. Figure 5 illustrates a server that deliver multiple streams (in Queue 1, 2, ...) to a client via overlay paths 1, 2, etc. In this model, there is one scheduler and L path services (each service corresponds to one overlay path used to deliver packets, with service rate $r^j(t)$).

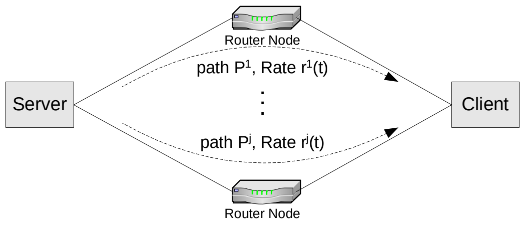


Figure 4: Overlay Routing and Scheduling Algorithm Framework.

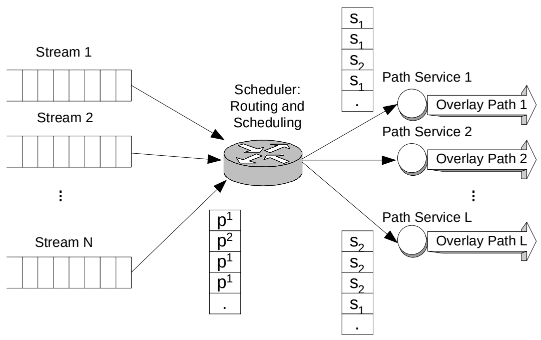


Figure 5: Routing and Scheduling on the Server.

Applications specify stream utility in terms of the minimum bandwidths they require, or using Window-Constraints [46] requirement. A Window-Constraint is specified by the values x_i, and y_i, where y_i is the number of consecutive packet arrivals from stream S_i for every fixed window, and x_i is the minimum number of packets in the same stream that must be serviced in the window. The idea, of course, is to guarantee to an application that at least some minimum number of packets in each time window will be serviced. Utility formulations like these have been shown useful widely useful, ranging from scientific applications (to limit buffering needs), to multimedia applications (to reduce jitter), to real-time applications (to limit numbers of missed deadlines). In the high performance domain, such guarantees are useful for control traffic (to limit the number of late control messages), for data traffic (to reduce buffering needs for high end data streams), and to distinguish the criticality of data transmission across multiple data streams.

The dynamics of the underlying network make it difficult to satisfy the minimum bandwidth guarantees required by the utility specifications described above, including for the guarantees associated with each scheduling window t_w. We address this issue by asking applications to specify additional requirements of the following nature: ensure that a window constraint is met with some large probability P(e.g. 95%, 99%). This also means that 95% of the time, the user's minimum bandwidth requirement will be satisfied. Given these specifications, assuming a packet size of s, and denoting the available bandwidth over a given path by $b^j(t)$, or simply b, (1) the available bandwidth distribution is described as the cumulative distribution function $F^j(b) = P\{avail\_bw \in (0,b)\}$, and (2) the service rate of the path service j is described as $r^j = r^j(t)$, where r^j varies over time.

B. Predictive Guarantee Overlay Scheduling/Routing Algorithm

The Predictive Guarantee Overlay Scheduling/Routing Algorithm (PGOS) supports two types of guarantees for stream utility specifications: probabilistic and 'violation bounded'. The former states that with some large probability P, stream S_i will receive the required bandwidth on the selected path. It also means that the stream S_i will receive the required bandwidth for at least $100P\%$ of the time. The latter states that the average number of packets that miss their constraint during each scheduling window can be bounded. In this section, we first define a single path selection algorithm for predictive guarantees and we then extend it to a scheduling algorithm that operates across multiple overlay paths.

1) Single path guarantee: The idea of single path selection is to choose the best path among all candidate paths for stream S_i, with some desired guarantee. Single path selection is important because there exist streams that are not easily mapped across multiple paths, an example being a stream with tight deadline/bandwidth requirements which would have to cope with synchronization issues and out of order arrivals when mapped across multiple paths.

*a) **Probabilistic guarantee**:* The following is the probabilistic guarantee provided by the PGOS algorithm. For brevity, all proofs appear in the appendix:

Lemma 1: Suppose during time $(t, t+t_w)$, where t_w is the length of the scheduling window, the available bandwidth distribution of server j is $F^j(b^j)$. Then, with probability $P = 1 - F^j(x_i s/t_w)$, it is guaranteed that x_i packets will be served during the scheduling window t_w.
Note that this guarantee essentially bounds the probability of insufficient throughput by $F^j(x_i s/t_w)$

*b) **'Violation bound' guarantees**:* Another useful application-level utility specification is to bound some violation, such as the deadline miss rate. The following is the deadline 'violation bound' guarantee provided by PGOS, where Z is the number of packets that miss their deadlines during one scheduling window, given the rate distribution $G^j(r^j)$ in this scheduling window:

Table I: Precedence among packets in different streams.

	Packet Ordering
1.	pkts scheduled on current path.
2.	pkts scheduled on other path:
2.1	Earliest deadline first.
2.2	Equal deadlines, highest window constraint first.
3.	pkts not scheduled:
3.1	Earliest deadline first.
3.2	Equal deadlines, highest window constraint first.

Lemma 2: Given available bandwidth distribution $F^j(b)$, $E[Z]$ is bounded by $x_i \cdot F^j(b_0) - \frac{t_w}{s} \cdot M[b_0]$, where b_0 is the required bandwidth of Stream S_i, $b_0 = x_i s/t_w$, and $M[b_0]$ is the mean of b for all $b \leq b_0$. Both $F^j(b_0)$ and $M[b_0]$ can be easily computed from the available bandwidth distribution.

2) Guarantees for multiple overlay paths: By combining the properties of multiple paths, PGOS can provide better guarantees to applications than those achievable on single paths. This is particularly relevant to large data transfers, where the parallelism achievable across multiple paths can be used to speed up data transfers as well as desired 'in flight' processing.

In Section V-B.1, two types of guarantees are developed for each a single path. As a result, we can quantitatively evaluate each of the overlay paths connecting the same server/client pair and choose the best overlay path to deliver a particular stream S_i. Based on these guarantees, we now describe an overlay routing and scheduling algorithm that maps multiple streams across multiple paths, as depicted in Figure 4. The algorithm schedules all packets of streams $S_i, i = 1, 2, ..., N$ such that the best guarantee is provided for the timely delivery of high utility streams, while other streams are delivered with less stringent guarantees. The PGOS algorithm, therefore, consists of two parts: (1) utility-based resource mapping and (2) path routing and packet scheduling.

*a) **Utility-based Resource Mapping**:* The resource mapping part of the PGOS scheduling algorithm (see Figure 6) finds the best proportion of stream S_i to be delivered via path P^j(**resource mapping**). The result is the generation of a *scheduling vector*, which is then used for routing and scheduling stream packets across multiple paths. The resource mapping step is executed when a new stream joins (or an existing stream terminates) or when the CDF of some path changes dramatically. A single resource mapping typically persists across many scheduling windows.

During each scheduling window, PGOS schedules packets based on the current scheduling vector and the stream precedence listed in Table I. This table maintains the statistically optimal stream division scheme, while also utilizing additional available bandwidth whenever possible. For example, given two overlay paths' available bandwidth distribution $G^j(j = 1, 2)$, and two streams $S_i(i = 1, 2)$, the table is set up to

```
1.   updateCDF(); /*update CDF using bandwidth/lossrate measurement in last scheduling window*/
2.   if(previous scheduling vectors doesn't satisfy currentCDF){
       /*this happens when new stream joins or CDF changes dramatically*/
3.     Find best scheduling share $Tp_i^j$ ;
       /*$Tp_i^j$ is the number of packets belonging to stream $i$ and scheduled to be sent on path $j$*/
       /*now rebuild scheduling vectors:*/
4.     for( $i = 1$; $i \leq N$; $i++$){
5.       for( $j = 1$; $j \leq L$; $j++$){
6.         $Tp^j += Tp_i^j$; /*for path lookup vector*/
           /*rebuild path lookup vector:*/
           /*Insert deadlines corresponding to Tpi(j) into $VD^j$*/
7.          UpdatePathDeadlineVector($VD^j$, $Tp_i^j$).
8.       }
9      }
       /*build path lookup vector*/
10.    $VP$=PathSchedVector($Tp^j$);
       /*convert deadlines to stream scheduling vector*/
11.    $VS$ []=StreamSchedVector($VD$ []);
     }/*end of scheduling vectors update(when necessary)*/

12.  while(in current scheduling window){
13.    path=GetNextFreePath(V^p); /*get next path according to V^p and path's backoff status*/
       /* get next packet to send based on V_s[p]*/
14.    if(getNextScheduledpkt(V_s[p]))
15.      sendpkt(path, pkt);
16.    else if(pkt=getNextUnscheduledPkt(V_s)){
         /*other unscheduled pkt. Precedence rule 2 and 3.*/
17.      sendpkt(path, pkt);
18.    }
19   }
```

Figure 6: Scheduling Algorithm.

divide each stream S_i into two sub-streams $S_i^1 + S_i^2$, where S_i^1 will be sent via path 1 and S_i^2 will be sent via path 2, such that their required performance guarantees are met. Note that S_i^j could be a null sub-stream, if necessary. We will send S_1^1 and S_2^1 via path 1, and send S_1^2 and S_2^2 through path 2.

Stream precedence is determined by the probabilities with which different streams' bandwidth requirements must be met. If streams S_i desire to receive their required bandwidths $100P_i\%$ of the time, then PGOS first finds the path that can satisfy the requirement of the most important stream (with highest P_i), then finds the path for the second most important stream, and so on. If there does not exist a single path that can satisfy stream S_i's requirement, then the stream S_i is divided into multiple parts S_i^j if this can satisfy stream S_i's requirement. If this still fails due to limited bandwidth, an upcall is made to inform the application that it is not possible to schedule this particular stream. The application can reduce its bandwidth requirement (e.g., from 95% to 90%) or try to adjust its behavior to the limited available bandwidth [7].

When a deadline violation bound guarantee is desired, PGOS works in a fashion similar to the probabilistic guarantees described above. When one or multiple streams join, PGOS begins with new stream with the highest deadline guarantee (i.e., with $\underset{i}{Minimum}[E[Z_i]]$), and attempts to find a path to meet its guarantee. If such a path does not exist, PGOS divides stream S_i(with x_i packets) into multiple parts S_i^j(with x_i^j packets) such that $\sum_{j=1}^{P} E[Z_i^j]\frac{x_i^j}{x^j} \leq E[Z_i]$, where $x_i = \sum_{j=1}^{P} x_i^j$ and $x^j = \sum_{i=1}^{N} x_i^j$. An alternative approach is to find a feasible division scheme without considering the ordering of $E[Z_i]$ and solve a mixed integer linear programming problem(MILP). However, this is not desirable since it may divide some important stream (e.g., a control stream) into multiple sub-streams, thereby causing synchronization and delays due to potential packet re-ordering across multiple overlay paths. It is also an N-P hard problem. A detailed analysis of alternative approaches and their comparison are beyond the scope of this paper.

*b) **Path Routing and Packet Scheduling**:* While it may be computationally complex to find the best possible resource mapping, this fact that does affect fast path PGOS performance. Here, during each scheduling window, PGOS needs to schedule packets according to the resource mappings encoded in scheduling vectors and according to the precedence table (see Table I). The efficient data structures used by PGOS are depicted in Figure 6: the scheduler has a path routing vector VP, and each path service has one stream scheduling vector VS. The scheduling vector V encodes the currently best resource mapping scheme derived by the resource mapping step. The lookup vector VP is the vector the scheduler uses to switch between the different overlay paths. As derived in the resource mapping step, path j is assigned x^j packets, so path j is assigned x^j virtual deadlines $D_p[k] = t_w/x^j \cdot (k-1)$. Virtual deadlines are used to maintain the desired resource mapping proportion. That is, VP contains the ordering to be used for visiting each path, based on virtual deadlines.

To illustrate, consider a concrete example with two streams and two overlay paths. Stream S_1 has 5 packets in one scheduling window that are mapped to path 1. Stream S_2 has 10 packets in one schedule window, where 4 of them are mapped to path 1, while another 6 packets are mapped to path 2. In this example, path 1 has 9 packets to deliver, and path 2 has 6 packets to deliver. Thus, VP=[1,2,1,2,1,1,2,1,2,1,1,2,1,2,1]. When the scheduler switches between the overlay paths, the path lookup vector ensures that three fifths of the time, it will visit path 1, and two fifths of the time, it will visit path 2. Stated more generally, when the scheduler visits path j, it uses the stream scheduling lookup vectors VS^j to select the streams to which to send packets.(VS is essentially a lookup table where each row corresponds to one path). The lookup vectors VS^j are based on the deadlines of all of the packets (from multiple streams) to be sent on path j. In the example, path 1 has nine packets, and the deadlines of these 9 packets are for S_1, S_2, S_1, S_2, S_1, S_2, S_1, S_2, and S_1 respectively. Thus, $VS^1 = [1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1]$.

While using VS for path mapping, PGOS schedules packets based on both VP and VS. That is, once it has selected path j, PGOS sends packets over it according to VS. Specifically, it selects a packet to send based on the stream scheduling lookup vector VS^j and the precedence table (Table I). First, it sends the packet scheduled on the current path $j$some other path that has the earliest deadline. Equal deadlines are broken by the window constraint x/y (highest window constraint first) and further ties are broken arbitrarily. When all scheduled packets have been sent out and there are still free paths to utilize, PGOS sends out other unscheduled packets according to their deadlines and window constraints. Whenever a path is blocked, the scheduler switches to the next path immediately, in order to best utilize other available resources. Because of the high cost of blocking, timeouts and exponential backoff are used to avoid sending multiple packets to a blocked path.

The following theorem states more precisely the guarantees provided by PGOS:

Theorem 1: If there is a feasible schedule for PGOS to deliver streams $S_i, i = 1, 2, ..., N$ over paths $P^j, j = 1, 2, ...L$ during scheduling window $(t, t+t_w)$, then stream S_i's window constraint will be met with probability P_i.

C. PGOS Buffer Size Analysis

By providing stable bandwidth to applications, PGOS also reduces the server buffer size, which is particularly desirable for heavily loaded streaming servers. Further, it reduces the client buffer size required for smooth playback, which is important not only for client delays experienced in real-time streaming, but also to reduce waiting time before playback for stored video streaming. Intuitively, all these are made possible by consistently delivering data from server to client at some required bandwidth, resulting in only a small number of packets delayed in the server buffer or in the network. Formally, considering a single stream's buffers at server and client, let these buffer sizes be $B_s(t)$ and $B_r(t)$ respectively, the sending rate at the server side and the playing rate at the client side be r_s and r_p, and the network transfer rate be $r(t)$. Note that $r(t)$ is the maximum rate the stream can achieve given background traffic and other competing streams. In addition, the actual rate at the stream is sent out($r^{'}(t)$) is subject to the actual number of packets available during a small time slot $(t, t+t_w)$. Thus the actual number of packets sent out during time $(t, t + t_w)$ is $MIN\{B_s(t^-)/s + t_w r_s(t), t_w r(t)\}$ and the actual rate the receiver receives is

$$r^{'}(t, t + t_w) = MIN\{\frac{B_s(t^-)}{s\delta} + r_s(t), r(t)\} \qquad (1)$$

If the sender buffer is limited to a maximum buffer size B_{s_MAX}, then the value of $B_s(t)$ at any time $t + t_w$ is given by:

$$B_s(t + t_w) = MIN\{(r_s - r^{'}(t))t_w + B_s(t^-), B_{s\_MAX}\} \quad (2)$$

If $B_s(t)$ is unlimited, then the value of $B_s(t)$ at t is given by

$$B_s(t^-) = \int_0^{t^-} (r_s - r^{'}(t))dt \qquad (3)$$

By formalizing this process as a M/G/1 queuing system and using the Pollaczek-Khintchine formula for the average time each packet spends in the queue, the average sender buffer size can be obtained as:

$$E[B_s(t)] = \frac{r_s^{-2} \cdot E[S^2]}{2(1 - E[S]/r_s)}, \qquad (4)$$

where $S = \frac{1}{r'}$, and $E[S^2] = Var(S) + (E[S])^2$.

From formula 4, it is clear that given two possible paths or scheduling schemes with the same average available bandwidth, the routing/scheduling scheme that minimize the variation of S, $Var(S)$, will ensure the minimum average sender buffer size $E[B_s(t)]$. Intuitively, packets are accumulated in the sender buffer when the transfer rate $r(t)$ is lower than the sending rate r_s, and PGOS reduces this buffer size by finding the best routing/scheduling scheme that can provide the required sending rate most of the time. When the used sender buffer size is low, it also means that the packets are not delayed in the buffer, and the receiving rate r_p is close to the sending rate r_s without bursty transfer, so a minimum number of packets are required to be buffered before playback. In addition, undesirable delays in playback are less possible. In comparison, if we only considered average bandwidth measurement and prediction, without considering the statistical structure of available bandwidth, we cannot distinguish different scheduling schemes leading to different levels of bursty transfer and different server/client buffer size requirements.

VI. Experimental Evaluation

This section evaluates IQ-Paths with three types of applications: (1) the SmartPointer system [49] for distributed collaboration and interactive program steering, (2) the GridFTP [14] implementation for reliable parallel data transmission in wide area networks, and (3) a prototype implementation of MPEG-4 Fine-Grained Scalable video streaming. Our testbed emulates a realistic wide area setting, using the EmuLab facility [25]. NLANR traces are used to inject representative cross-traffic [12].

The first experiment shown below evaluates our methods for statistical prediction of network bandwidth. Next, the PGOS algorithm is used to map application-level data streams across multiple overlay links, and its performance is compared to that of the widely used fair queuing scheduling algorithms (WFQ and MSFQ). We also compare the performance of parallel file transfer using GridFTP vs. IQ^{PG}-GridFTP, which is our implementation of GridFTP based on the original GridFTP and PGOS.

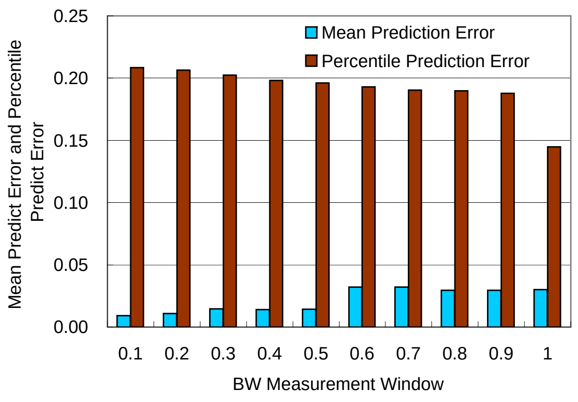


Figure 7: Bandwidth Prediction.

A. Bandwidth Prediction Analysis

Figure 7 illustrates the results of predicting average bandwidths vs. the statistical predictions used in our approach. Here, we analyze more than 8GB of IP header trace files from the National Laboratory for Applied Network Research (NLANR). The traces were collected at a number of locations of the Abilene (Internet2) and the Auckland networks. The mean prediction error is the average relative error ($|(predicted\ value - actual\ value)/actual\ value|$) of several widely used average bandwidth predictors (i.e., MA, EWMA and SMA). From Figure 7, the common average bandwidth predictors have a roughly 20% of prediction error. Similar error ranges are also reported in [50]. On the contrary, our statistical prediction method (percentile prediction) achieves less than a 4% prediction failure rate. The percentile prediction failure rate is the number of prediction failures divided by the total number of predictions. For these experiments, we first calculate the distribution of N (e.g., 500 and 1000) samples, where each sample is the bandwidth measured in 0.1 to 1 second. Then, since we are particularly interested in whether a path can guarantee certain throughput for 90% of the time (or for 80%, 70%, etc), we find distribution D's 10th percentile as X(Mbps), and test whether the next n (n=5 to 10) samples are larger than X. If they are, a successful prediction occurs, and if not, a prediction failure occurs.

From these experiments and for representative distributed applications, we determine two facts. First, in practice, an application is typically less interested in the average bandwidth it receives over a long period than in the fact that it can receive its required bandwidth consistently most of the time. This is precisely the question that is answered by our probabilistic bandwidth prediction methods. Second, such statistical guarantees are easier to make than guarantees on available average bandwidth, because the majority of available bandwidth or maximum throughput on Internet paths is IID [50]. As a result, the exact value of average bandwidth in the near future is hard to predict, but the statistical structure of bandwidth can be predicted well. Simply speaking, if in the last 5 mins., the 10th percentile of bandwidth is 10Mbps, then with a large probability, the bandwidth in the next 1 second will be higher than 10Mbps. The measured low prediction failure rate directly justifies our usage of percentile prediction.

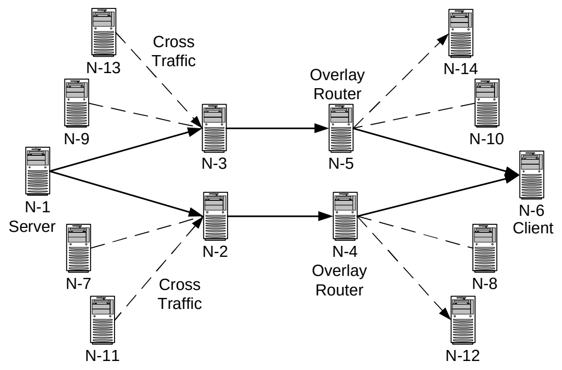


Figure 8: Testbed. The link connecting each pair of nodes is fast ethernet. Cross traffic is injected by Node N-9 to N-14. Overlay routers are placed at Node N-4 and N-5, so that overlay paths and cross traffic paths share the same bottleneck (N-3 to N-5 and N-2 to N-4).

Table II: Importance of choosing the right path to meet service-level objectives. '95%' means the 95th percentile point.

	Throughput		Jitter(ms)		Interval(ms)	
	95%	99%	mean	std	mean	std
Atom-pathb	2.4748	1.927	1.7	6.1	3.97	9.9
Atom-patha	3.2336	3.2322	0.82	1.3	3.99	0.9
Bond1-pathb	17.2284	13.0244	1.7	6.1	3.97	9.9
Bond1-patha	22.0442	22.0303	0.83	1.3	3.99	0.9

B. SmartPointer Experiments

The PGOS algorithm embedded in the IQ-Paths middleware is used with three applications: the SmartPointer system for distributed collaboration and steering of computational applications, GridFTP, a high-performance and reliable data transfer protocol widely used in the Grid community, and MPEG-4 Fine-Grained Scalable video streaming. We configure an Emulab [25] testbed in which the overlay server N-1 has two overlay paths to reach the client N-6(Figure 8). Background traffic is generated from NLANR traces. The background traffic and data traffic share the common link between N-3 and

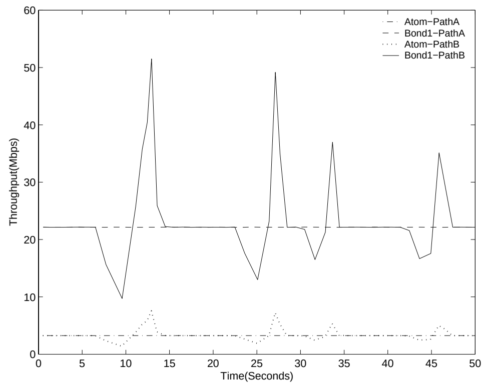


Figure 9: Throughput on each path. Although path A has less mean available bandwidth than path B, it is preferable for streams ‘atom’ and ‘bond1’

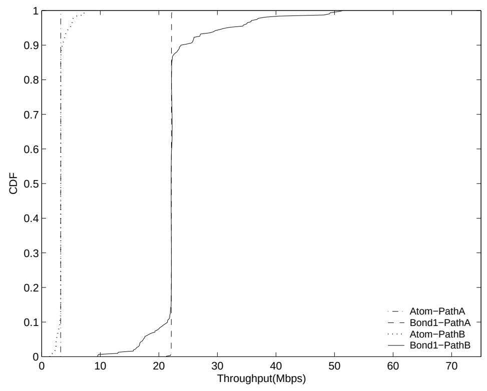


Figure 10: Throughput CDF on each path. Bandwidth on path B is more dynamic than bandwidth on path A.

N-5, and the link between N-2 and N-4. All link capacities are 100Mbps, which is the current up-limit of Emulab.

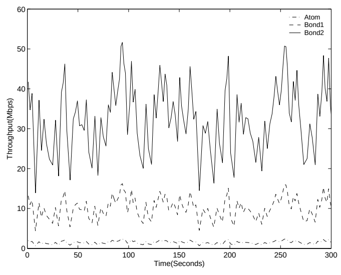
1) Importance of choosing the right path: We first evaluate the importance of choosing the ‘right’ path for an application’s data streams. In this evaluation, cross traffic based on trace files obtained from NLANR is injected into two overlay paths. The average available bandwidth on Path B is higher than that on Path A, but it has larger variation compared to Path A. Two streams of the SmartPointer (streams ‘Atom’ and ‘Bond1’) are transferred from Node1 to Node6 over either of these two overlay paths. Results are depicted in Figure 9. In this figure, the time series of Atom-PathA and Bond1-PathA are the throughputs of streams Atom and Bond achieved if we utilize Path A, and the time series of Atom-PathB and Bond1-PathB are throughputs achieved by the two streams if we use Path B. Although Path B has higher average bandwidth, its higher variation causes unstable bandwidth with wide fluctuation. The cumulative distributions of the throughput of the two streams over two paths are given in Figure 10. Obviously, Path A can provide much more stable bandwidth than Path B.

The importance of choosing the right path based on the distribution of bandwidths instead of simply average bandwidth is summarized in Table II. If Path A is chosen to transfer the two streams simultaneously, for 91% of the time, the stream Atom can obtain at least its required bandwidth, and for 99% of the time, it can obtain at least 99.7% of its required bandwidth. At the same time, stream Bond1 is able to obtain at least its required bandwidth for 90% of the time and at least 99.69% of its required bandwidth for 99% of the time. For comparison, if we choose path B, which has higher average available bandwidth than Path A, stream Atom can obtain its required bandwidth for only 44% of the time, and for 99% of the time, it can only obtain 59.46% of its required bandwidth. Stream Bond1 can obtain its required bandwidth for only 59% of the time, and for 99% of the time, it can only obtain 58.94% of its required bandwidth. This means that the two streams can’t obtain their required bandwidths most of the time. This results in bursty transmission behavior, large queue lengths on the server side, and higher jitter, none of which are desirable for this remote collaboration application. Note that low jitter is particularly important for real-time applications like remote scientific visualization, as it provides smoother data streaming and also reduces total buffering. As shown in Table II, the two streams can achieve much lower average jitter and lower standard deviation in jitter if PathA is chosen instead of PathB (0.82 vs. 1.7 for average jitter and 1.3 vs. 6.1 for standard deviation.)

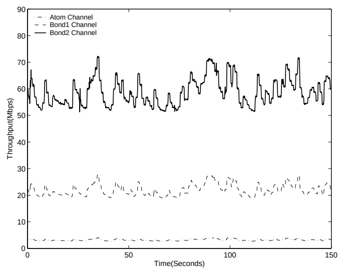
This experiment demonstrates the importance of assessing the distribution of available bandwidth to meet application-level service requirement vs. assessing average bandwidth values. When we transfer large data volumes, average bandwidth is one important factor, but it is not a sufficient one. Specifically, by using the distribution of available bandwidth, PGOS can find the path to transfer application data that has the best predictive guarantee. We next discuss further improvements in attainable end-to-end bandwidth, by using PGOS to schedule traffic across concurrent network paths.

2) PGOS evaluation: The second experiment evaluates PGOS with multi-path message routing and scheduling. The purpose is to see how the algorithm can guarantee some critical stream’s required throughput while also providing high throughput to non-critical streams. Consider the SmartPointer server issuing three streams (Atom, Bond1, and Bond2) to remote clients. Streams Atom and Bond1 are data about all atoms and those bonds that are in the observer’s immediate graphical view volume, whereas stream Bond2 contains the bonds outside the observer’s current view. Therefore, Streams Atom and Bond1 are important and must be delivered in real-time (25 frame/sec) for effective collaboration, but stream Bond2 is less critical (e.g., it may be important when the observer rapidly changes his/her viewing angle.)

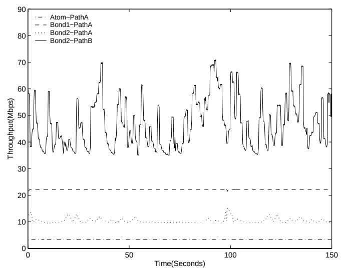
Three on-line message transfer algorithms are evaluated and compared to meet this application’s needs: (1) transfer all messages over one single path based on normal Fair Queuing (WFQ), (2) transfer messages over two paths with fair queuing, and (3) transfer messages over two paths using the proposed PGOS routing and scheduling algorithm. For (2), we use the multi-server Fair Queuing (MSFQ) algorithm [3]. The input (utility requirements) to PGOS are 3.249Mbps with 95% predictive guarantee for stream Atom and 22.148Mbps with 95% predictive guarantee for stream Bond1.



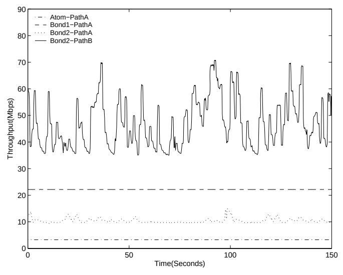
(a) ***Non-Overlay Fair Queuing***



(b) ***Multi-Server Fair Queuing***

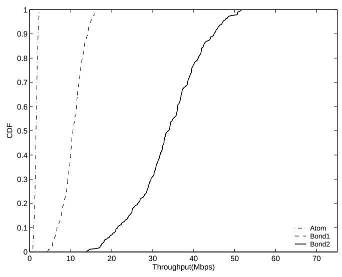


(c) ***PGOS Algorithm***

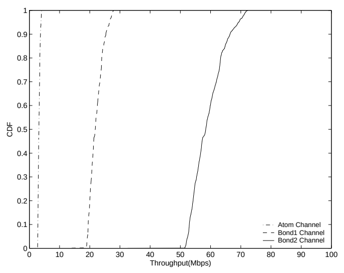


(d) ***OptSched Algorithm***

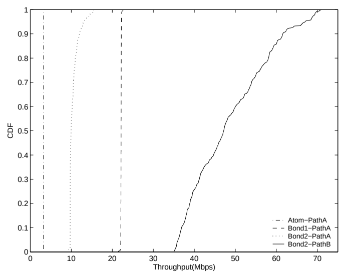
Figure 11: Throughput Time Series Comparison of Three Algorithms.



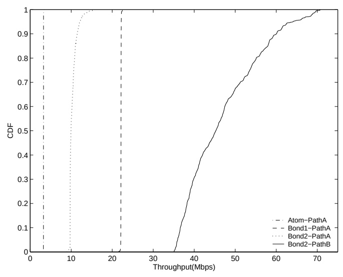
(a) ***Non-Overlay Fair Queuing***



(b) ***Multi-Servers Fair Queuing***

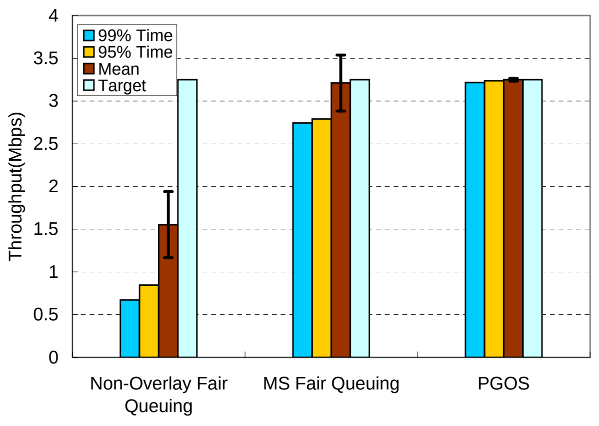


(c) ***PGOS Algorithm***

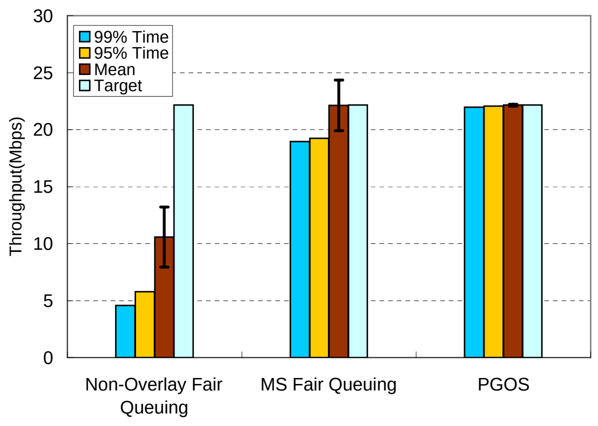


(d) ***OptSched Algorithm***

Figure 12: Throughput CDF Comparison of Three Algorithms.



(a) ***Stream 1(Atom): Targeted Throughput: 3.249Mbps***



(b) ***Stream 2(Bond): Targeted Throughput: 22.148Mbps***

Figure 13: Throughput Achieved by Three Algorithms: Target, Mean, 95% of the time, 99% of the time, and Standard Deviation(represented by the vertical bars).

We also compare these results with a near-optimal off-line algorithm, termed OptSched, which assumes that we know available bandwidth a priori. Although this off-line algorithm cannot be used in practice, it can be used to gauge the absolute performance of PGOS. OptSched operates as follows. First, it maintains a vector of each path's available bandwidth. Second, since it knows future bandwidth a priori, it schedules packets at the beginning of each scheduling window such that the application's requirement of bandwidth can be ensured (e.g., 3.249Mbps for at least 95 percent of the time). Note that even if available bandwidth is known a priori and data is sent at the appropriate rate (e.g., at a speed of 3.249Mbps), the receiving rate could still be slightly less or higher than the sending rate because of competition among multiple flows. OptSched deals with this via a simple feedback control loop: if the the actual receiving rate is not equal to the required receiving rate in the past, it will send this stream at $3.249 + \delta$ Mbps, where δ is a dynamic variable based on past history. When the previous scheduling window's receiving rate was less than 3.249Mbps, δ equals two times of the absolute value of the difference between the receiving rate and 3.249Mbps. If the receiving rate was higher than $3.249 + \beta$Mbps, where β is a very small threshold to stabilize this control loop, delta is set to half of the difference between the receiving rate and 3.249Mbps, multiplied by -1. Because this off-line optimal algorithm requires precise clock synchronization of the streaming server, the client, and the server introducing cross traffic, it is evaluated via ns2 simulation.

The results of using these four algorithms are depicted in Figure 11. Figure 11a depicts the throughput of 3 streams attained by the WFQ algorithm on Path A, which has higher available bandwidth than Path B with larger variance. Multi-Server Fair Queuing (MSFQ) can maintain the proportion of throughput shared by the three streams quite well (see Figure 11b), but because of its inaccurate average bandwidth prediction, it fails to provide the required throughput to the two critical streams Atom and Bond1. Both streams exhibit substantial throughput fluctuation. In comparison, the PGOS algorithm successfully provides very stable throughput to these two critical streams. Furthermore, note that in Figure 11c, the throughput of stream Bond2 is not compromised. This stream is divided by PGOS into two substreams (Bond2-PathA and Bond2-PathB), and the average throughput of stream Bond2 is almost the same as that achieved by MSFQ.

The cumulative distributions of throughput of the three streams under the three algorithms are given in Figure 12, and a summarized comparison of these three algorithm appears in Table III. Note that PGOS provides the two critical streams at least 99.5% of their required bandwidth (denoted by 'Target' in Table III) for 95% of the time. MSFQ can only provide about 87% of their required bandwidth for 95% of the time. For example, stream Bond1 requires 22.148Mbps, and the actual 95th percentile of the achieved bandwidth is 22.068Mbps under PGOS, but it is only 19.248Mbps under MSFQ. The standard deviations of bandwidth experienced by the two critical streams appear in Table III and Figure 13. Although stream Bond2 has slightly larger standard deviation with PGOS, the two critical streams Atom and Bond1 experience much lower standard deviations.

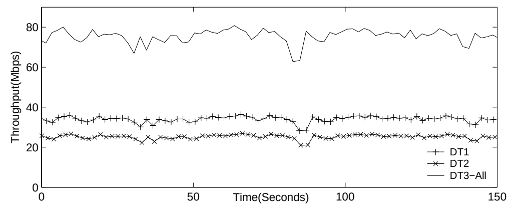
Both Fair Queuing and Multi-Server Fair Queuing try to allocate bandwidth in a proportional based manner according to predicted bandwidth, but they require exact values of end-to-end bandwidth, which are hard to attain. Further, although both of these two algorithms can successfully maintain the proportion of the bandwidth allocated to multiple streams, they cannot provide specific bandwidth to a particular stream. In comparison, PGOS relaxes the prediction assumption, only asking if we can obtain certain bandwidth with some high probability. This is not only easier to predict, but also directly provides the functionality needed by applications.

All three algorithms experience certain overheads when routing single streams over multiple paths, because of packet reordering and possible delays of head-of-line packets. PGOS reduces this overhead by using a single path for one stream whenever possible, especially for streams with higher priorities. Simply speaking, unlike MSFQ which provides the two critical streams less than required bandwidths when the network is congested and more than required bandwidths when the network is free of congestion, PGOS routes and schedules packets such that the two important streams obtain stable required bandwidths no matter whether or not one path is congested. As a result, the application frame jitter is also reduced from 2.0ms (with MSFQ) to 1.4ms (with PGOS).

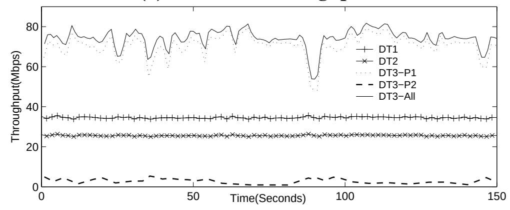
In summary, these experiments show that with PGOS routing/scheduling, critical streams' required throughput can be guaranteed most of the time. This is done without compromising the average throughput experienced by non-critical streams. A case in point in our experiments is non-critical stream Bond2, which still receives almost the same average throughput under PGOS as under MSFQ.

Table III: Routing/Scheduling Algorithms Comparison.

	Throughput(Mbps)					Jitter
	95 Perct.	99 Perct.	Mean	Std.	Target	(ms)
Atom	0.846	0.672	1.5524	0.3863	3.249	5.0
Atom^F	2.789	2.744	3.2111	0.3273	3.249	2.0
Atom^P	3.236	3.216	3.2487	0.0150	3.249	1.4
Atom^O	3.240	3.239	3.2489	0.0058	3.249	1.2
Bond1	5.768	4.569	10.5843	2.6348	22.148	5.0
Bond1^F	19.248	18.946	22.1300	2.2321	22.148	2.0
Bond1^P	22.068	21.959	22.1476	0.0790	22.148	1.4
Bond1^O	22.138	22.139	22.1477	0.0273	22.148	1.3
Bond2	18.297	14.486	33.7021	8.3949	70.340	5.0
Bond2^F	52.446	51.76	59.0786	8.0397	70.340	2.0
Bond2^P	45.782	45.038	59.0588	9.5765	70.340	1.5
Bond2^O	45.757	45.019	59.0583	9.5587	70.340	1.5



(a) ***GridFTP Throughput.***



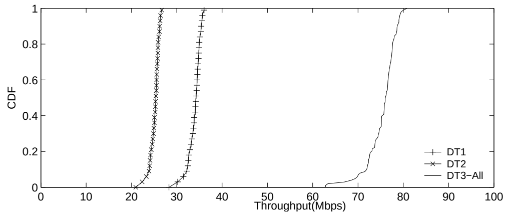
(b) ***IQ^{PG}-GridFTP Throughput. Line DT3-All is the throughput achieved by stream DT3(sum of throughput on two paths: DT3-P1 and DT3-P2).***

Figure 14: Throughput Achieved by GridFTP and IQ^{PG}-GridFTP

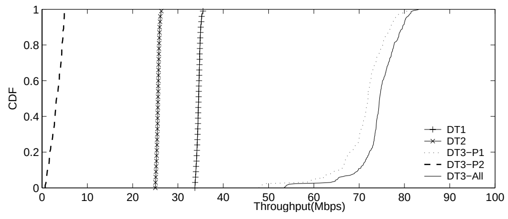
C. GridFTP Experiments

GridFTP [14] is widely accepted as one of the common data transfer services available for high performance applications. One important extension to the FTP protocol implemented in GridFTP is its support for parallel data-transfer. The GridFTP standard represents this functionality by supporting SPAS (Striped Passive)/SPOR(Striped Data Port) instructions to establish multiple data connections. Parallel file transfer then occurs using the parallelism options (like parallel-opts, layout-opts) to the RETR (Retrieve) command. These options essentially determine the number of parallel connections to be established and the layout with which the data is to be distributed across these connections.

In this subsection, we present our experiences with IQ^{PG}-GridFTP, which strengthens our previous work [7] by including support for PGOS-enabled parallel file transfers. The



(a) ***GridFTP Throughput CDF.***



(b) ***IQ^{PG}-GridFTP Throughput CDF.***

Figure 15: GridFTP and IQ^{PG}-GridFTP Throughput CDF Comparison

Table IV: GridFTP.

Stream Name	Throughput(Mbps)			
	Target	95 Perct.	Mean	Std.
DT1^O	34.56	31.431	33.9411	1.4297
DT1^P	34.56	33.869	34.5505	0.4040
DT2^O	25.60	23.282	25.1415	1.0590
DT2^P	25.60	25.094	25.5990	0.2993
DT3^O	76.80	69.393	75.4246	3.1770
DT3^P	76.80	65.287	74.3577	4.7668

IQ^{PG}-GridFTP implementation generalizes the publicly available wu-ftpd [16] server to support the GridFTP protocol extensions for parallel transfers and implements the Partitioned and Blocked data layout options to distribute file contents across the connections in addition to the PGOS layout. A partitioned data layout is one where contiguous chunks of file are distributed evenly across all the connections for transfer, while a blocked data layout is one where data blocks (each of size block-size) are distributed in a round-robin fashion.

We use a climate database in our experiment as simulation of the Earth System Grid II [13]. Each record in this database has three data components: (1) the numeric data (approximately 172.8KB and denoted by 'DT1'), and (2) and (3) are low resolution images (128KB, and denoted by 'DT2') and high resolution images (384KB, and denoted by 'DT3'), respectively. GridFTP and IQ^{PG}-GridFTP are configured to concurrently transfer file records across two TCP connections over two overlay paths. For such transfers, we want to ensure that the numeric data and low resolution images receive their required bandwidths of at least 25 records/second for real-time data streaming. In addition, we also want to fully utilize bandwidth to transfer high-resolution data.

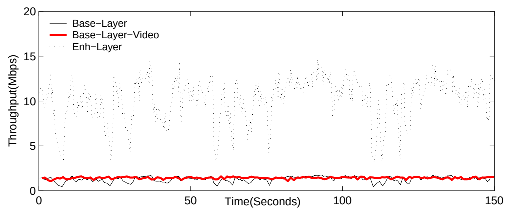
Experimental results are depicted in Figures 14 and 15. From these measurements, it is apparent that IQ^{PG}-GridFTP can ensure that the streams DT1 and DT2 receive their required bandwidths consistently, while stream DT3 is transferred as fast as possible. In comparison, standard GridFTP splits the dataset into blocks allocated to the multiple connections for transfer, but when the available bandwidth of any path is low, all types of data have to compete with each other. This causes the important data streams to not receive their required bandwidths during these periods. Specifically, consider the two curves for DT1 and DT2 in Figure 14a, which have much wider fluctuations compared to curves DT1 and DT2 in Figure 14b. Quantitatively, stream DT1 achieves 33.94Mbps average throughput using GridFTP with a large standard deviation (1.4297), while using IQ^{PG}-GridFTP, it achieves 34.55Mbps average throughput with a small standard deviation (0.4040). Similar results are observed for stream DT2. Note that here the network can provide almost the total throughput required by the application. If the network cannot provide such throughput, then the two streams DT1 and DT2 obtain even less bandwidth using GridFTP, as they have to compete with stream DT3 for the same limited bandwidth. In summary, with PGOS, IQ^{PG}-GridFTP can protect more important streams from competing with other less important streams, while also scheduling less important streams to be delivered when there exists sufficient bandwidth. Applications have full control over deciding how much bandwidth will be allocated for a particular stream and what kind of guarantee is for each stream.

D. Multimedia Streaming Experiments

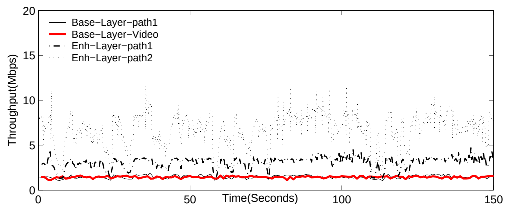
Video and audio streaming over the Internet are known to be important applications. Because the dynamic behavior of the Internet makes it difficult to provide consistently good quality of streaming video/audio, layered coding and multiple description (MD) provide layers of encoded video. Both layered and MD coding can leverage the QoS enhancements offered by PGOS, and in this section, we evaluate the performance of PGOS used with MPEG-4 Fine-Grained Scalable (MPEG-4 FGS) layered video coding [1].

The MPEG-4 FGS framework consists of a base layer and one or two enhancement layer. The base layer is generated by motion estimation/motion compensation and entropy coding with fixed quantization step size. The SNR FGS enhancement layer adds DCT coefficients with reduced quantization step size and leads to more accurate DCT coefficients and higher video quality. The Temporal FGS enhancement layer improves temporal resolution by providing a higher frame rate and smooth motion. The base layer is the most important set of data, and its bandwidth requirement should be consistently provided for smooth playback. Receivers can subscribe to as many enhancement layers as possible to maximum video quality, but these layers are less important and may be dropped at when there in insufficient available bandwidth.

The base layer and the enhancement layer require 1.4820Mbps and 11.2901Mbps average bandwidths, respectively. Since encoded video exhibits variable throughput, the input parameter for PGOS is a 95% prediction guarantee of 1.22Mbps for the base layer, which corresponds roughly the 95 percentile of the actual bitrate of the base layer. There is no requirement for the enhancement layer, i.e., we would like to transmit the enhancement layer using the remaining available bandwidth.

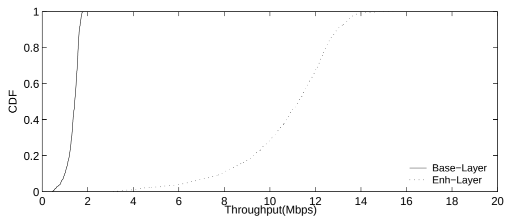


(a) ***Multi-Servers Fair Queuing(MSFQ)***

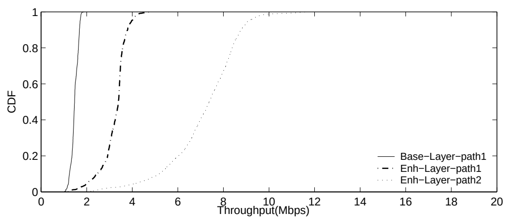


(b) ***PGOS***

Figure 16: Throughput Time Series of MSFQ and PGOS Algorithms.



(a) ***Multi-Servers Fair Queuing(MSFQ)***



(b) ***PGOS***

Figure 17: Throughput CDF Comparison of Three Algorithms.

Experimental results appear in Figures 16 and Figures 17. In Figure 16, the thick red line is the bandwidth of the encoded base layer, and the solid black line is the delivered bandwidth of the base layer. Comparing these two graphs, PGOS can deliver about the bandwidth required by the base layer. In comparison, since mean bandwidth cannot be predicted well, with MSFQ, for some time, the achieved bandwidth of the base layer is significantly less than the required video bitrate. More precisely, PGOS provides 1.20Mbps for 95 percent of the time while MSFQ provides only 0.81Mbps for 95 percent of the time (see Table V). Further, PGOS provides at least 1.22Mbps for 93 percent of time which is very close to our bandwidth guarantee requirement (i.e., at least 1.22Mbps for 95 percent of the time). In comparison, MSFQ provides at least the required bandwidth for only 78 percent of the time.

Table V: MPEG-4 FGS.

	Throughput(Mbps)		
	95 Perct.	Mean	Avg. Req.
Base Layer^F	0.81722	1.3693	1.4820
Base Layer^P	1.2026	1.4761	1.4820
Enh. Layer^F	6.5250	10.7690	11.2901
Enh. Layer^P	6.2931	10.3910	11.2901

VII. Conclusions and Future Work

This paper presents IQ-Paths, a data streaming middleware that uses overlay networks to better serve the needs of distributed applications running on wide area networks. IQ-Paths employs statistical methods to provide to applications predictive guarantees for the bandwidths available to them from the underlying network, for all paths in the overlay connecting data sources to sinks. In addition, its PGOS scheduling algorithm both suitably routes packets across overlay paths and schedules them across single and multiple (concurrent) paths, coupling parallelism in data transfer with statistical bandwidth guarantees.

The statistical prediction technique used in IQ-Paths not only measures average available bandwidth, but also captures the dynamic or noisy nature of the bandwidth available on overlay paths. As a result, IQ-Paths can provide to applications both probabilistic and 'bounded violation' delivery guarantees. The former state that with some large probability P, stream S_i will receive the required bandwidth on the selected path. The latter state that the average number of messages that violate some constraint (e.g., miss their deadlines) during each scheduling window is bounded.

This paper uses IQ-Paths to meet the needs of three representative distributed applications: (1) the SmartPointer real-time collaboration system, (2) multimedia data streaming, and (3) GridFTP. The integration of IQ-Paths into these applications is facilitated by its design as a 'model-neutral' data streaming layer underlying the application-specific communication models offered by higher middleware layers, including the publish/subscribe model used by SmartPointer, the simple data transfer model used by GridFTP, and the data streaming model used by the multimedia application.

Several extensions of the proposed IQ-Paths framework are of future interest. The path characteristics collected by IQ-Paths can benefit not only the applications shown in this paper, but may also be used by other multimedia or high performance data transfer methods. In addition, it would be interesting to extend this work to content delivery systems that use overlay multicast techniques. For enterprise applications, our current research is developing runtime methods for fault tolerance. Here, an intersting use of IQ-Paths is to differentiate data traffic required for replication from other traffic, perhaps dynamically varying reliability/performance tradeoffs with selective replication techniques. Another interesting topic is to use IQ-Paths to isolate the effects of fault tolerance or recovery traffic from regular data traffic, perhaps to avoid the additional disturbances arising during recovery. Finally, we will generalize to additional service objectives, such as message loss rate service guarantees.

VIII. Acknowledgements

We gratefully acknowledge input from Constantinos Dovrolis on the methods for network bandwidth measurement and packet scheduling used and developed for this paper.

References

[1] I. 14496-2/FPDAM4. Coding of audio-visual objects, part-2 visual, amedment 4: Streaming video profile, July 2000.

[2] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-Path vs. Multi-Path Overlay Routin. In *proc. of IMC*, 2003.

[3] J. M. Blanquer and B. Ozden. Fair queuing for aggregated multiple links. In *Proc. of ACM SIGCOMM*, 2001.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] J. W. Byers, J. Considine, and M. Mitzenmacher. Informed Content Delivery Across Adaptive Overlay Networks . *IEEE/ACM Transactions in Networking*, 2004.

[6] Z. Cai, G. Eisenhauer, Q. He, V. Kumar, K. Schwan, and M. Wolf. IQ-Services: Network-Aware Middleware for Interactive Large-Data Applications. In *Proc. of MGC*, 2004.

[7] Z. Cai, G. Eisenhauer, Q. He, V. Kumar, K. Schwan, and M. Wolf. Iq-services: Network-aware middleware for interactive large-data applications. *Concurrency & Computation. Practice and Exprience Journal*, 2005.

[8] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *Proc. of SIGCOMM*, 2001.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowmant. Planetlab: An overlay testbed for broad-coverage services. *ACM Computer Communication Review*, 33(3), July 2003.

[10] DOE. UltraScience Net. `http://www.csm.ornl.gov/ultranet/`.

[11] W.-C. Feng and J. Rexford. Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video. *IEEE Trans. on Multimedia*, Sept 1999.

[12] N. L. for Applied Network Research". Internet measurement and Internet analysis. `http://moat.nlanr.net/`.

[13] I. Foster, E. Alpert, A. Chervenak, B. Drach, C. Kesselman, V. Nefedova, Middleton, S. D., A. A., Sim, and D. Williams. The Earth System Grid II: Turning Climate Datasets Into Community Resources. In *Annual Meeting of the American Meteorological Society*, 2002.

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

[15] F. B. Greg Eisenhauer and K. Schwan. Event Services for High Performance Computing. In *Proc. of High Performance Distributed Computing*, 2000.

[16] W. D. Group. WU-FTP. `http://www.wu-ftpd.org`.

[17] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, Aug. 2003.

[18] M. Karlsson and C. Karamanolis. Choosing Replica Placement Heuristics for Wide-Area Systems. In *proc. of ICDCS*, 2004.

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

[20] C. Krasic, K. Li, and J. Walpole. The Case for Streaming Multimedia with TCP. *Lecture Notes in Computer Science*, 2158, 2001.

[21] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-Aware Distributed Stream Management using Dynamic Overlays. In *Proc. of ICDCS*, 2005.

[22] V. Kumar, B. F. Cooper, Z. Cai, G. E. B. Seshasayee, P. Widener, and K. Schwan. A Self-Adaptive Infrastructure for Composing and Managing Distributed Information Flows. In *submitted to Eurosys-2006*, 2005.

[23] V. Kumar, B. F. Cooper, and K. Schwan. Distributed Stream Management using Utility-Driven Seld-Adaptive Middleware. In *proc. of IEEE International Conference on Autonomic Computing*, 2005.

[24] N. LambdaRail. `http://www.nlr.net/`.

[25] J. Lepreau and et. al. The Utah Network Testbed. `http://www.emulab.net/`. University of Utah.

[26] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Enabling Platforms*, Sept. 1999.

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

[28] P. Mehra. Information, The Final Frontier. In *proc. of High Performance Interconnects for Distributed Computing Workshop(in conjunction with the 14th International Symposium on High Performance Distributed Computing)*, 2005.

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

[30] M.Jain and C. Dovrolis. End-to-end Estimation of the Available Bandwidth Variation Range. In *Proc. of ACM SIGMETRICS*, June 2005.

[31] A. Nakao, L. Peterson, and A. Bavierr. A Routing Underlay for Overlay Networks. In *Proc. of ACM SIGCOMM*, 2003.

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

[33] R. K. Rajendran and D. Rubenstein. Optimizing the Quality of Scalable Video Streams on P2P Networks. In *proc. of IEEE Globecom 2004*, 2004.

[34] N. S. V. Rao. Overlay Networks of In-Situ Instruments for Probabilistic Guarantees on Message Delays in Wide-Area Networks. *IEEE Journal on Selected Areas of Communications*, 22(1), 2004.

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

[36] P. Ratanchandani and R. Kravets. A Hybrid Approach for Internet Connectivity for Mobile Ad Hoc Networks. In *Proc. of IEEE WCNC*, 2003.

[37] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Playback Video over the Internet. In *Proc. of ACM SIGCOMM*, 1999.

[38] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy Caching Mechanism for Multimedia Playback Streams in the Internet. In *proc. 4th International Web Caching Workshop*, 1999.

[39] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. *IEEE/ACM Transactions on Networking*, 10(3), June 2002.

[40] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of ACM SIGCOMM*, 2002.

[41] N. Spring, D. Wetherall, and T. Anderson. Reverse-Engineering the Internet. In *proc. of HotNets-II*, 2003.

[42] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS. In *Proc. of 1st Symposium on Networked Systems Design and Implementation(NSDI)*, Mar. 2004.

[43] J. Vetter and K. Schwan. Techniques for High-Performance Computational Steering. *IEEE Concurrency*, 7(4), 1999.

[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 *Proc. 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 *Proc. of IEEE Real-Time Systems Symposium*, 2000.

[47] R. West and K. Schwan. Quality Events: A Flexible Mechanism for Quality of Service Management. In *Proc. of RTAS*, 2001.

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

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

[50] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proc. of the ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

Appendix

A. Proof of Lemma 1

Proof: Let the service rate over path j at time t be $r^j(t)$. Then the service rate cumulative distribution $G^j(r^j)$ is:

$$\begin{aligned} G^j(r^j) &= P\{r \leq r^j)\} = P\{rs \leq r^j s\} \\ &= P\{b \leq r^j s\} = F^j(r^j s). \end{aligned} \tag{5}$$

The probability that x_i packets will be served during the scheduling window t_w is

$$\begin{aligned} P &= P\{x_i \leq rt_w\} = P\{x_i/t_w \leq r\} \\ &= 1 - P\{r \leq x_i/t_w\} \\ &= 1 - G^j(x_i/t_w) = 1 - F^j(x_i s/t_w). \end{aligned}$$

□

Note that this is essentially bounding the probability of throughput violations.

B. Proof of Lemma 2

Proof:

$$Z = \begin{cases} x_i - r^j t_w & \text{if } x_i > r^j t_w \\ 0 & \text{if } x_i \leq r^j t_w \end{cases} \tag{6}$$

Let f_B be the pdf of available bandwidth b, then we have

$$\begin{aligned} E[Z] &= \int_{-\infty}^{+\infty} Z \cdot f_B(b) d(b) \\ &= \int_0^{x_i s/t_w} (x_i - bt_w/s) \cdot f_B(b) d(b) \\ &= x_i \cdot \int_{-\infty}^{b_0} f_B(b) d(b) - \frac{t_w}{s} \cdot \int_{-\infty}^{b_0} b f_B(b) d(b) \\ &= x_i \cdot F^j(b_0) - \frac{t_w}{s} \cdot M[b_0] \end{aligned} \tag{7}$$

□

C. Proof of Theorem 1

Proof: Let the service rate stream S_i receives on path j be r_i^j, and stream S_i will obtain service rate r_i^j with probability P_i^j, $\sum_{j=1}^{L} P_i^j = P_i$. X_i is the actual number of packets delivered for stream S_i during the scheduling window t_w, and among these packets, X_i^j packets are delivered through path j.

The probability that x_i packets will be served during the scheduling window for stream S_i is:

$$\begin{aligned} P &= P\{x_i \leq X_i\} = P\{x_i^1 \leq X_i^1, ..., x_i^L \leq X_i^L\} \\ &= \sum_{j=1}^{L} P\{x_i^j/t_w \leq X_i^j/t_w\} \\ &= \sum_{j=1}^{L} P\{x_i^j/t_w \leq r_i^j\} = \sum_{j=1}^{L} P_i^j = P_i \cdot \end{aligned} \tag{8}$$

□