

A Multi-path Routing Service for Immersive Environments *

Sherlia Shi, Lili Wang, Kenneth L. Calvert, and James N. Griffioen

Laboratory for Advanced Networking
University of Kentucky
Lexington, KY 40506

Abstract

The Metaverse project aims to develop technology for low-cost, high-resolution networked immersive display environments that can be used for distributed collaboration, exploration of 3D data models, scientific visualization, and other grid-related applications. Such applications often deal with massive data set and need a high-capacity, low-latency transport service to effectively connect distant locations across the wide-area (best-effort) Internet. This paper presents the initial design of such an end-to-end transport service for Metaverse applications, along with results of a simulation study evaluating its effectiveness. The transport service features an application programming interface providing enhanced control over the way resources are allocated to data objects, and uses multiple overlay-based end-to-end paths to increase bandwidth delivered to the application.

1 Introduction

The Internet, Web, and more recently, “Grids,” have fundamentally changed the ways in which people communicate, learn, interact, share information, and perform large-scale computations. Despite these impressive scientific and technological advances, however, the primary modes of computer-based communication and collaboration remain largely unchanged. Most communication still occurs using decades-old mechanisms such as electronic mail or chat programs. Even multi-party video conferencing and collaboration software (whiteboards and application-sharing) have had a limited effect [1, 2, 3, 4]. Video conferencing with postage-stamp-sized images, low polygon-count 3D models, and strict limitations on the number of participants leave much to be desired.

*This work supported in part by NSF Grants EIA-0101242 and ANI-0121438, DARPA agreement number F30602-99-1-0514, and by Intel and MCSI Corporations.

As part of the *Metaverse Project* [5], we have been developing new paradigms for users to interact with their data and with one another over the network. Part of the project focuses on development of high-resolution, low-cost, easy-to-install immersive environments constructed from commodity PCs and projectors. Figure 1 illustrates a tiled projector immersive environment. Such environments enable a wide range of new human-computer interaction paradigms including immersive exploration of 3D models (e.g., virtual spaces, scientific models, volumetric medical data (MRIs)), visualization of large, high-resolution images (e.g., digital library documents, sculpture, and historical artifacts), hands-on training models (e.g., combat training or medical training), and live video interaction or group collaboration.



Figure 1. (a) arbitrarily placed overlapping projectors are (b) automatically calibrated and blended together so show a 3D model of Divinci's Workshop.

Although similar interactive paradigms have been available in high-end, expensive systems (e.g., CAVEs) [6, 7, 8], the ability to deploy these low-end systems in environments ranging from offices, to conference rooms, to dedicated training facilities, makes them accessible to a wide range of users and will place new demands on the system, particularly on the underlying network infrastructure. Because immersive environments can be deployed in the average user's office, there is an increased need for remote (interactive) access to large data sets/models as well as live data feeds from other immersive spaces. For many applications, the (3D) data models to be rendered can easily be 10's of giga-

bytes in size, consisting of a time series of large polygonal meshes, massive volumetric data, and texture maps. Consequently, designing a network substrate capable of meeting the (soft) real-time transmission requirements of these immersive systems is a significant challenge.

In this paper we consider the design of an end-to-end transport service to support such immersive display applications. We begin by describing a receiver-driven *service abstraction* that provides applications (i.e. immersive environments) with enhanced control over the allocation of transmission resources to data objects, and also provides timely estimates of (best-effort) future performance. Such ability can be useful in determining which objects to request next. We then describe an algorithm for selecting multiple parallel data paths across an overlay network to improve throughput between endpoints. We present three different approaches to the problem of route selection in such a network, followed by results from simulation experiments to evaluate their effectiveness under a range of conditions.

2 End-to-End Transport for Immersive Environments

Although the Metaverse’s immersive displays offer users an excellent environment in which to view their data on a large screen with high resolution, retrieving such data from a server over a wide-area network presents significant challenges for the transport protocol. The typical data set is composed of multiple components (polygonal meshes, texture maps, volumetric data, etc), possibly with a time component (e.g., the 3D model’s evolution over time). Given the resulting massive size of the data, a key goal of the transport service is high throughput. Unfortunately, high throughput alone is neither necessary nor sufficient. For our intended applications, the viewing experience is an interactive one, in which the participant controls the information being displayed in (more or less) real time. Although only a portion of the total data set/model is needed by the display system at any given time, that part can still be large and may be more than the network can transfer in a timely fashion. In this case, the application needs to work together with the transport service to achieve the best viewing experience possible given the current network conditions. This means that the API to the transport service must be able to provide information about the level of performance it is currently able to offer, and it must allow the application to prioritize data sent across the channel to ensure the most important data needed for the display arrives.

Assuming the application can learn the current bandwidth and delay offered by the transport service, the application can then select a rendering approach. For example, given a connection with limited bandwidth but low latency, the application may choose to transmit the 3D polygonal

mesh at low resolution and visually compensate for the lack of depth with a high resolution texture map. Figure 2 illustrates two 3D models consisting of many hi-res texture maps, millions of polygons, metadata, as well as volumetric data, that can be rendered in a wide range of ways, depending on the characteristics of the underlying network transport service.



Figure 2. Example 3D models.

One possible way to design the network abstraction for immersive environments is to leverage existing transport services (e.g., multiple TCP streams) in conjunction with the Internet Protocol’s *differentiated services* architecture. This places the work of managing multiple connections squarely on the application, but has the advantage of allowing the application to specify the network service each component requires. However, because the service is built solely on the existing IP infrastructure, the service has no control over the path (route) packets take through the network. Consequently packets belonging to the same component, may take widely different routes, resulting in unpredictable performance. Moreover, this model assumes that differentiated (or integrated) services is fully deployed in the Internet; which to-date has not occurred.

Given our concerns with existing transport services, we identified specific needs of a metaverse transport service, and then designed an abstraction and API to meet those needs. In particular, the transport service must offer the following features:

- The abstraction should be that of a single end-to-end communication channel from source to destination (called the session channel) together with a single feedback channel. The application should not be aware of, or need to manage, multiple connections.
- The session channel should be reliable.
- The application should be able to specify the priority of its data. Higher priority data may slow down or preempt completely the delivery of lower priority data. A session’s priority specification should not affect inter-session flow scheduling.
- The application should be able to query the service for an estimation of bandwidth and delay characteristics of the session channel. In other words, the end systems should be able to ask “what if” questions and get

back estimates of what would happen if data were sent according to the proposed priority scheme.

In summary, the application needs to be able to (1) obtain information about how well the network will perform in the future (based on past performance), (2) specify a prioritized delivery plan, and (3) then let the underlying service map the delivery plan onto actual network paths. It should be noted that we do **not** require that the service offer any Quality of Service (QoS) guarantees. This allows the service to operate across any best-effort network (i.e., does not require diffserv support), but implies the application is responsible for (continuously) adjusting to changing network conditions.

To meet the foregoing requirements, we designed a new service model and API for transporting Metaverse data. The service abstraction is receiver-driven, in that the destination display decides what parts of the data are most important.

To achieve prioritized data transmission, the receiving application specifies the data to be transmitted as a hierarchy of nodes, where each leaf node represents a (semantically meaningful) data unit (“object”) to be transmitted. Objects are the smallest logical unit of transmission and must be delivered to the destination application as a whole¹. Internal nodes represent groups of objects and define the percentage of the bandwidth to be given to each object in the subtree. Figure 3 shows an example hierarchy. Each tree denotes a priority level. Within a priority tree, node weights denote the fraction of bandwidth used to transmit that node. Weights on the children are w.r.t. the weight of their parent. In some cases, a node’s children objects need to be delivered in a particular order to the application—e.g., a series of texture maps encoded as a delta off the previous map. To support ordered delivery, each node contains a flag indicating whether the children of a node are ordered or not.

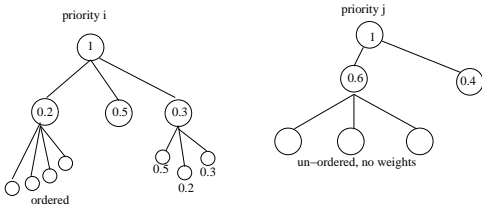


Figure 3. Example data delivery specification. Each tree denotes a priority level; weights denote fractional bandwidth.

The API consist of four basic calls: *open_session()*, *close_session()*, *set_recv_handle()*, and *schedule_objects()*. The first two establish and tear down a session. The

¹Note that an object may be further decomposed by the underlying service into fragments.

set_recv_handle() call is used by the receiver to process incoming data, while the *schedule_objects()* call is used by the receiver to set/change the priority/weights used to transmit the data. Moreover, the *schedule_objects()* call returns a “estimated arrival time” for each of the objects specified in the hierarchy. The specific parameters for each call are shown in Table 1. The structure *MetaInfo* refers to the priority trees, including a field in each node of the tree with its estimated finish time.

3 A Multi-path Transport Service

A key goal of the service is to maximize session throughput. A well-known technique to boost application throughput is to open multiple TCP connections (e.g., used in web browsers). Unfortunately, this approach allows a flow to grab a larger share of the bottleneck bandwidth than a single TCP flow, and thus, some would argue, exceeds the limits of good “net citizenship”. Worse yet, an application could use non-TCP friendly protocols (say UDP) to grab an unfair share of the bottleneck bandwidth. However, even if one took these approaches, the achievable bandwidth would be limited by the bottleneck link on the (single) IP path from the source to the destination.

Overlay networks avoid this problem by supporting multiple concurrent paths through the Internet while maintaining TCP-friendliness on each path. Where standard Internet routing protocols try to find the *single* shortest path route from source to destination, the set of overlay nodes can cooperate to discover multiple routes across the Internet. Although the discovered routes may have higher latency than the shortest path, the ensemble of path(s) can offer higher throughput. To verify this, we compared throughput of the shortest path route to the throughput of an alternate (overlay) path through the Internet using the Planetlab overlay network [9]. Our results showed that throughput improvements as large as 30 times were possible. For example between *planetlab1.postel.org* and *planetlab1.lcs.mit.edu*, the default route offered a throughput of 714 Kb/s, while using an overlay path through UCLA, Utah, WUSTL, Georgia Tech, Duke, NYU resulted in an average throughput of 20.84 Mb/s.

In the following, we present a new multi-path overlay routing network that (1) passively monitors available path bandwidth, (2) dynamically discovers under-utilized paths, (3) selects appropriate routes subject to constraints limiting the load imposed on the network, and (4) dynamically distributes a session’s load across the selected paths.

3.1 Issues in Multi-path Overlay Transport

The first issue that must be addressed is *available bandwidth discovery*. In order to find alternate routes through

Function Name	Description
<code>int open_session(struct sockaddr* svrAddr)</code>	Open a session to the server and return the session ID
<code>int close_session(int sessId)</code>	Terminate the specified session
<code>int set_rcv_handle(int sessId, int objId, void *buf, int len, CallBackFunction* call_back)</code>	Register a callback function to handle incoming objects as they arrive. Sufficient buffer space must be provided or, incoming objects will be discarded.
<code>MetaInfo* schedule_objects(int sessId, MetaInfo* objects, int modified)</code>	Specify the transmission schedule (as a list of MetaInfo priority trees). Side-effect: initiates (or restarts) transmission using the specified schedule. Returns the given MetaInfo tree with the estimated object arrival times filled in.

Table 1. Metaverse Transport Service API.

the overlay, an overlay node must be able to estimate the available bandwidth from itself to other overlay nodes. This information can then be shared among nodes (e.g., much like link state routing algorithms), allowing a source to select the best set of paths for its session. A variety of bandwidth estimation approaches have been proposed in the past [10, 11, 12] which *actively* probe by injecting traffic into the network to determine the available bandwidth. Unlike these approaches, we employ a *passive* approach that continuously monitors and records the bandwidth usage. In our design, “virtual links” are established hop-by-hop between overlay nodes using TCP connections. Each node keeps track of two variables on each of its adjacent links: one for the maximum achievable bandwidth, measured as the peak bandwidth corresponding to when the TCP window is the largest; and the other for the average measured bandwidth. The difference between these two values can be used to estimate available bandwidth. Because the overlay’s “virtual links” are shared by all flows, the estimates benefit all flows, and can be built up over time; whereas two random endpoints that suddenly decide to communicate using standard routing protocols must use active/intrusive methods to estimate available bandwidth.

The second issue to be addressed is *path selection*. Path selection involves several sub-issues including how many paths should be selected, what is the maximum length a path can be, what is the set of disjoint paths, and ultimately, which paths optimize the objective function. Note that a wide range of objective functions are possible, where an objective function defines the tradeoffs between the above factors. In the next section, we describe three different algorithms for path selection, and present simulation results showing the expected throughput for each approach.

The third issue involves *load balancing* among the selected paths. Given a set of paths, the transport service must map the user’s MetaInfo specification onto the available paths. Although a variety of mappings are possible, our service attempts to minimize buffering and reassembly effects by mapping all the packets from a single object unit (a leaf node in Figure 3) to the same network path, while

distributing the objects across network paths to match the session’s share of bandwidth on that path. In general, the goal is to minimize the “finish time,” while satisfying the buffering constraints at the receiver. The specific “finish time” computation is described in the following section.

3.2 Computing Object Finish Time

The finish time of an object is a receiver-side estimation of the object’s transmission completion time. It is a function of the available bandwidth, delay, and size and priority of the objects waiting to be transmitted.

The finish time is computed from high to low priority and from the top of the object hierarchy to the bottom follows: Let B be the total network bandwidth and $\Omega_k = \text{MetaInfo}[k]$, the set of top-level nodes of priority k . We compute the finish time τ_j of $n_j \in \Omega_k$ as follows: (assume $s_j/B \gg d_j$, where s_j is the size of object n_j and d_j the path delay)

while $\Omega_k \neq \emptyset$ **do**

let n_x be the node with $\tau_x = \min_{n_j \in \Omega_k} \frac{s_j}{w_j \times B}$

for all $j \neq x$ **do**

$$s_j = s_j - \tau_x \times w_j \times B$$

$$w_j = \frac{w_j}{(\sum_j w_j) - w_x}$$

$$\Omega_k = \Omega_k - \{n_x\}$$

The finish time for each object in priority k is adjusted to include the longest finish time in priority $k - 1$. For each internal node, if its finish time is not directly obtained, it is set to the maximum of any node located in its subtree. If nodes are ordered and the parent node of a node at level i is n_k^{i-1} , then the estimated finish time is simply: $\tau_j^i =$

$$\tau_j^{i-1} * \frac{s_k^i}{s_k^{i-1}}.$$

4 Overlay Multipath Routing Algorithms

Our goal of using multiple paths within the overlay is two-fold. First, we want to improve the overall (overlay) network utilization. Second, we want to share the bandwidth fairly while maximizing the average throughput of

connections. The implication of these goals is that sessions should collectively share network capacity to maximize total bytes transferred while preventing any session from gaining an unfair share of the bandwidth.

The objective functions for the routing algorithm are shown below. We model an overlay network as a directed graph $G = (V, E)$, and let $C_{i,j}$ denote the capacity of link (i, j) , $F_{i,j}$ the flow on link (i, j) , x_p the flow on path $p \in P_i$, where P_i is the collection of all available paths between a source destination pair. Finally, \mathcal{W} denotes the set of all source-destination pairs.

$$\begin{aligned} \text{Objectives:} \quad & \text{maximize } \sum_{(i,j) \in E} \frac{F_{i,j}}{C_{i,j}} \quad \text{for all } (i, j) \in E \\ & \text{minimize } \sum_{\mathcal{W}} \frac{1}{\sum_{p \in P_i} x_p} \\ \text{Subject to:} \quad & F_{i,j} = \sum_{(i,j) \in p} x_p \leq C_{i,j} \end{aligned}$$

The second goal is achieved when $\sum_{\mathcal{W}} \left| \sum_{p \in P_i} x_p - \sum_{p \in P_j} x_p \right|$ is minimized. Therefore, with uniform connection size, it can be captured as minimizing the throughput variance among all connections.

The optimization of the first objective is quite straightforward: we want each connection to use as much of the residual network capacity as possible. The optimization of the second objective, on the other hand, is less clear. It requires an even allocation of flow rate for each connection, which would be particular hard to achieve with dynamic traffic. There are two choices made by a routing algorithm that impacts this objective: one is the number of paths chosen and the other is which set of paths to choose. For the first objective, we would like to choose as many paths as possible (subject to some number manageable by the application) and as wide paths as possible (where wide means higher available bandwidth). However, such a greedy approach can easily cause the overload of the network and bandwidth starvation of some sessions. Therefore, it is necessary to have the second goal to balance the usage of bandwidth among sessions and to optimize the overall connection time of all sessions.

Unlike a conventional router, the traffic forwarded by an overlay node may originate on the overlay node itself, as well as elsewhere in the network. To allow for different service levels for local vs. non-local traffic, we divide flows at each node into two categories: flows generated locally at that node, and flows in transit through the node. We use a parameter α to characterize the fraction of bandwidth used by the class of local flows and assign higher bandwidth to local flows. Furthermore, we assume flows belonging to the same class receive an equal share of bandwidth at the bottleneck link.

4.1 Path-Selection Approaches

We discuss three heuristic algorithms for selecting multiple paths in the overlay network and present preliminary simulation results. We first define a *feasible path* as a path between the source and destination pair that satisfies a set of constraints on the residual bandwidth, hop counts and path delay. Although the objective functions do not include any of these constraints, in practice, they are important properties that directly affect application performance. We use a fixed hop limit (*Max Hops*), and a relative delay bound (*Max Delay*) – the ratio of the shortest path delay to α times the shortest path residual (which is the fraction of bandwidth relative to the shortest path flow) – as the minimum path residual value. The link residual is computed as an exponentially-weighted moving average that is updated every unit time. Every update interval, a node broadcasts the residual values of all its adjacent edges to all other nodes.

Another important parameter is the maximum number of paths selected by the routing algorithm. Aside from being manageable by the application, this number represents a tradeoff between how much bandwidth an individual connection can get and how much load it contributes to the network. The optimal routing, as defined in Bertsekas and Gallager[13], injects flows only to the shortest (equal-length) paths. When using paths with length (hop counts) greater than the shortest path, no matter how the flow rates are distributed across the paths, the total utilization is always greater than that used by shortest path routing. This effect is exacerbated when we model the network as a complete graph, since an alternative path is at least twice the length of the shortest path. If the network offered load is low, the actual load generated by multipath routing is sustainable; however, as the network load increases, simply choosing multiple paths ultimately leads to overloading of the network. Therefore, an intelligent routing algorithm must dynamically adapt the paths it choose based on the current network load.

Random: As the name suggests, this algorithm randomly selects a few paths from the source to the destination. It starts from the source node and randomly selects the next hop. If the newly selected edge in addition to the current partial path satisfies the delay and hop limit constraints, it continues; otherwise it looks up another node as the next hop. When the destination is reached, it selects the new path and restarts from the source to select another path until it reaches the specified maximum paths, or no more path can be found. The selected paths are not necessarily disjoint, and the selection does not base on the residual bandwidth of the links.

Closest First: The closest-first multipath routing (CFM) proceeds in the breadth-first fashion in growing the paths.

Starting from the source, all possible edges that satisfy the constraints are remembered. When a path reaches the destination, all edges belong to the path are removed from the current path list. Therefore, paths are disjoint. Since it will always find a feasible path with shortest hop count first, we call it “closest-first.” Additionally, the growth of the paths at a certain depth uses a random order of nodes as the next hop; this is to avoid always first selecting nodes with lower identifiers. When all feasible paths are found, and there are more feasible paths than the maximum paths, we randomly select a few paths among the feasible set.

Widest First: The widest-first multipath routing (WFM) is a modified Bellman-ford algorithm to compute the widest path tree subject to the constraints. In each iteration, a widest path from the source to the destination is computed. Edges belong to the path are then removed from the graph. The algorithm then computes another widest path on the remaining edges and continues until no more path can be found. If there are more feasible paths than the maximum paths, we select randomly.

4.2 Simulation Results

Figure 4 and 5 shows the comparison of the algorithms on a complete graph of 5 nodes. We chose uniform link bandwidth and uniform link delay in this simulation. There are no path constraints in this setting (the parameters are chosen such that all possible paths are feasible). We observe the following:

- When $\alpha = 1.0$, sessions should get at least the same throughput as the shortest path routing because of the scheduling mechanism at intermediate nodes that protects the shortest path flows.
- The random algorithm can easily overload the network since it is more likely to select paths of longer hop counts and paths of small residual capacity, thus increasing traffic load and contention, respectively.
- The CFM and WFM algorithms performs similarly in this setting. We expect CFM to perform better for non-uniform settings with low network load (since it distributes traffic more evenly over the network), while WFM may suffer from always trying to select the widest paths (although there is a randomization of final selections among all “wider” paths).
- Under lower load, the larger the value of α , the fewer choices available for selecting alternative paths since there are fewer paths with capacity greater than the shortest path. It is interesting that with moderate α , the algorithms seem to be quite adaptive to the offered load, achieving high throughput with low load and converging to shortest paths at higher offered load.

- WFM executes much faster than CFM particularly with higher load, because when fewer paths are available, WFM can find them quickly while CFM will have to search for them.

5 Issues with Correlated Paths

The main difficulty of implementing the multipath routing as an overlay service is the lack of accuracy of path information obtainable at the overlay nodes. We have so far modeled the network assuming independent links, while the actual overlay paths are almost inevitably correlated. If two paths share a bottleneck link, selecting both paths will not necessarily increase the actual throughput. How to infer a shared bottleneck is a difficult problem in the current design. Furthermore, we not only want to know paths that are link-disjoint but also whether the shared link is the bottleneck link or not. One example is that links close to the source are always shared by all sessions, but they are unlikely the bottlenecks for session flows.

The correlation of overlay paths affects not only the selection of paths but also the scheduling mechanisms as well. At the least, the scheduling mechanism needs to be per node rather than per destination. Since some paths overlap, the sum of actual achievable throughput over all destinations must be less than the sum of the measured throughput on individual paths. One question is what rate should the shortest path flows receive? Is it α times the actual overall throughput, or α times the throughput of the overlay path?

6 Related Work

A variety of approaches have been developed to leverage the existence of multiple, redundant routing paths to improve efficiency and to provide more reliable and fault-tolerant network routing services. Gallager [13] first showed that splitting data transfer across multiple equal weight paths resulting in improved network utilization. This approach is often adopted in traffic engineering schemes such as in OSPF, but typically only exists at the network layer as a load balancing mechanism. Byers et al. [14] suggested using erasure codes and parallel downloading from multiple mirror sites to speed up data transfer. This approach requires data to be stored redundantly at multiple sites and therefore, is not directly applicable in our immersive display environment.

More recently, RON [15] has proposed a fault-tolerant routing services in the context of overlay networks. The key characteristic of their work is to find *alternate routing paths* in the time of routing failure on the primary path. In their more recent paper [16], they have suggested using multiple paths to provide reliable routing services by transferring redundant data across these paths. A commonality between

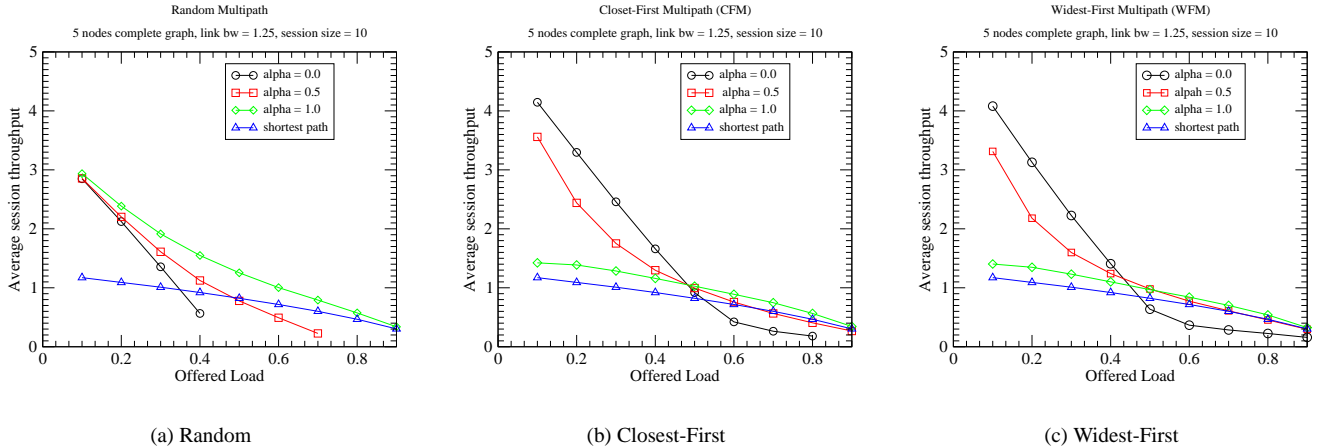


Figure 4. Average Session Throughput: Max Hops = 5, Max Delay = 5, Max Paths = 4

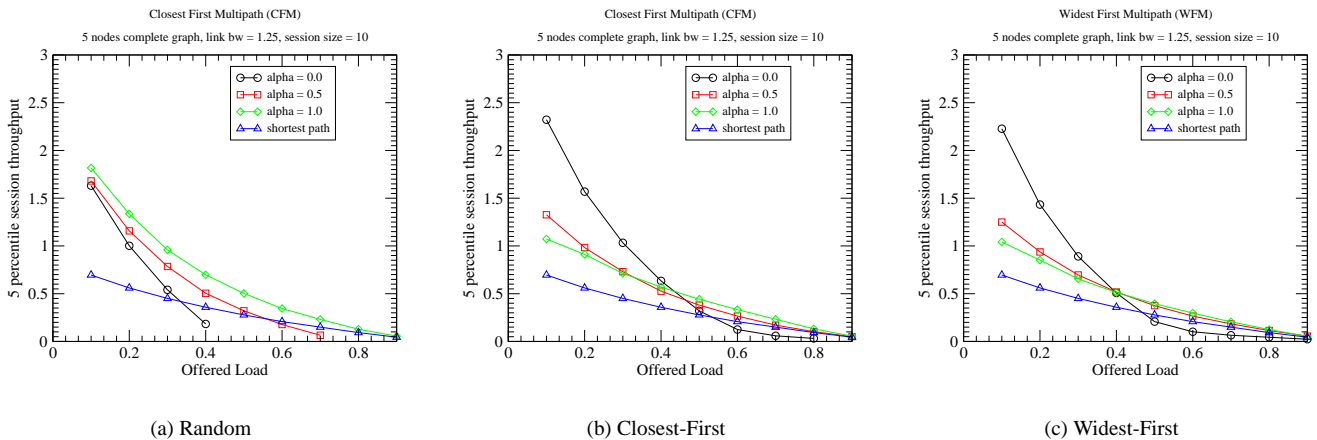


Figure 5. Lower 5th percentile Session Throughput: Max Hops = 5, Max Delay = 5, Max Paths = 4

our work and RON is that both leverage overlay networks to provide better routing services that are reactive to network topology and traffic conditions. However, in RON, the objective of the routing service is for fault-tolerance, and the paths discovery mechanism requires continuous active probing of a mesh network; while in our work, the objective is to improve application data transfer performance with a passive monitoring component. Additionally, our framework also extends to transport services such as data fragmentation.

7 Conclusions

Multipath routing offers the potential to support emerging multimedia applications such as collaborative immersive environments. In this paper we have presented a multipath routing service based on an overlay network, and

showed that with intelligent route selection mechanisms, multiple path routing can achieve the higher bandwidth desired by these demanding applications. However, raw bandwidth increase does not translate directly to application perceived performance increase. A flexible API is necessary to integrate the overlay service with the application demand. We have developed a simple API that gives applications limited control over the way their data is treated. In particular, we proposed a receiver-driven API that allows viewers to alter the way the data transmitted to meet their viewing needs.

Clearly, providing an overlay transport system with an alternative routing service involves many components that cannot be addressed in a single paper. To this end, we have described a multipath overlay network that: a) uses passive measurements to identify available bandwidth; b) dynamically selects paths subject to a set of constraints that limit

the load imposed on the network; and c) offers load balancing across selected paths based on user-specified priorities. We have prototyped the system and are currently investigating its performance on the Planetlab network.

References

- [1] “The Mbone Utilities,” <http://www.mbone.com>.
- [2] Microsoft, “The Microsoft Netmeeting Program,” <http://www.microsoft.com/netmeeting/>.
- [3] E. Chen, “Quicktime VR - An Image-Based Approach to Virtual Environment Navigation,” in *SIGGRAPH 95 Conference Proceedings*, Aug. 1995, pp. 29–38.
- [4] Volker Coors and Volker Jung, “Using VRML as an interface to the 3D data warehouse,” in *VRML 98: Third Symposium on the Virtual Reality Modeling Language*, Don Brutzman, Maureen Stone, and Mike Macedonia, Eds., New York City, NY, Feb. 1998, ACM SIGGRAPH / ACM SIGCOMM, ACM Press, ISBN 0-58113-022-8.
- [5] “The metaverse project,” 2003, <http://www.netlab.uky.edu/theme.html>.
- [6] D. Bennett, “,” Alternate Realities Corporation, Durham, NC 27703. Cited July 1999, <http://www.virtual-reality.com/>.
- [7] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti, “Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE,” in *SIGGRAPH 93 Conference Proceedings*, Aug. 1993, vol. 27, pp. 135–142.
- [8] M Czernuszenko, D Pape, D Sandin, T DeFanti, L Dawe, and M Brown, “The ImmersaDesk and InfinityWall Projection-Based Virtual Reality Displays,” in *Computer Graphics*, May 1997.
- [9] “Planetlab, <http://www.planet-lab.org>,” .
- [10] S. Keshav, “Packet-Pair Flow Control,” *IEEE/ACM Transactions on Networking*, February 1995.
- [11] Constantinos Dovrolis, Parmesh Ramanathan, and David Moore, “What do Packet Dispersion Techniques Measure?,” in *Proc. of IEEE Infocom*, Anchorage, Alaska, April 2001.
- [12] Robert Carter and Mark Crovella, “Measuring bottleneck link speed in packet-switched networks,” Tech. Rep. 1996-006, Boston University, 15, 1996.
- [13] Dimitri Bertsekas and Robert Gallager, *Data Networks, Second Edition*, Prentice Hall, 1987.
- [14] John Byers, Michael Luby, and Michael Mitzenmacher, “Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads,” in *Proc. of IEEE INFOCOM*, March 1999, pp. 275–283.
- [15] David G. Andersen and Hari Balakrishnan, “Resilient Overlay Networks,” in *Proc. of the 18th ACM Symposium on Operating Principles (SOSP)*, Banff, Canada, October 2001.
- [16] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan, “Best-Path vs. Multi-Path Overlay Routing,” in *Internet Measurement Conference*, Miami, Florida, October 2003.