Improving Throughput for Grid Applications with Network Logistics

Martin Swany
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716
swany@cis.udel.edu

Abstract

This work describes a technique for improving network performance in Grid environments that we refer to as "logistics." We demonstrate that by using storage and cooperative forwarding "in" the network, we can improve end to end throughput in many cases. Our approach uses TCP connections in series and offers performance benefits for high-bandwidth, high-latency networks. First, we examine the underlying causes of the logistical effect. Next, we present a graph based scheduling approach that can be solved quickly and, within our assumptions, optimally. Finally, we present a large-scale empirical evaluation of the system in order to validate our scheduling approach for taking advantage of network logistics. This study demonstrates performance improvement in many situations and aggregate speedup results are presented.

1 Introduction

The rapid evolution of the Internet has brought with it the capability of harnessing huge quantities of computational resources over the network. The "Computational Grid" [14], as this burgeoning paradigm is called, refers to the vision that computational "power" should be available from anywhere in the network – much as the electrical power grid allows us to plug in and draw power without regard for its source. The metaphor of a "Grid" is broad, but the core idea – that of utility or service-oriented computing – is compelling and promising.

0-7695-2153-3/04 $20.00 ©2004 IEEE

Unlike an electrical power Grid, resources on the computational Grid are not necessarily fungible, as the usefulness of a resource can depend on that resource's location. Local voltage standards aside, one can use electrical power regardless of how or when it was generated. For computational "power," this is not always true. Underlying the service abstraction are physical systems and they must generally load or store some amount of data. Thus, our goal is to increase observed bandwidth of data transfers – that is, to reduce the time it takes to transfer data. We couch these optimizations in terms of logistics and augment the general Grid model by including short-term, cooperative storage of user data.

1.1 Data Logistics for the Grid

We can improve end to end throughput in Grid environments by introducing "logistical" storage in the network. We approach the issue of bandwidth-limited Grid applications by asking how we can improve network performance for Grid systems using unmodified operating systems and networks. Our hypothesis is that we can improve observed network performance in Grid environments by scheduling the flow of data through logistical storage "depots" and further, that we can automatically produce these schedules based on information about the current network performance or the network's "performance topology." Finally, in order to make use of such a system in Grid environments, we must bundle the functionality in "middleware."

We refer to data *logistics* as the use of storage in the network to improve performance or functionality. This work is closely related to the logistical networking [6] research that produced the Internet Backplane Protocol [26] (IBP) in that we consider a storage-enabled network environment. Our current approach and IBP share a common notion of a logistical network depot for distributed computing environments. The IBP approach focuses on explicit control of buffers in the network whereas our work investigates the end to end performance effects of network logistics on synchronous *streams* of data. Thus, while the depots are conceptually the same and could potentially even share an implementation, our current approach never directly addresses the buffers in the network. Rather, buffer allocation and utilization are implicit. Therefore, this work is complementary to the IBP work, but considering a different level of abstraction and targeting a very focused problem – improvement of end to end throughput in Grid environments. We make no claims that this technique is appropriate for the Internet at large, only that it addresses problems being faced today in distributed, high-performance computing.

The contribution of this work is in an empirical evaluation of TCP connection dynamics and the proposal of a scheduling algorithm for forwarding through depots that takes these dynamics into account. We present an argument for logistical forwarding and demonstrate how such a technique might be used in computational Grid environments. First, we will provide a brief overview of our logistical approach. Then, we develop an understanding of the behavior that gives rise to the logistical effect. Next, we present a scheduling algorithm based on this behavior. Finally, we present empirical results from a large-scale evaluation of our scheduling approach.

2 Overview

The Logistical Session Layer (LSL) is a layer of middleware that enables improved network throughput via buffering in the network [32]. The basic model is that end to end communication between hosts is no longer bound directly to the Transport layer, but rather to a Session layer which is semantically similar to that defined by in the OSI model [13]. Recall that a transport-layer connection may consist of multiple network-layer hops. Analogously, a session-layer connection may consist of multiple transport-layer connections. In fact, the ISO standard specifically allows the binding of a single session “connection” to multiple transport connections. The session layer as designed by the ISO was never widely deployed. There are are other “session” protocols in the Internet protocol suite such as the Session Initiation Protocol (SIP.) However, none realize this sense of the term because none exist at layer 5 of the OSI protocol model, logically atop the transport layer.

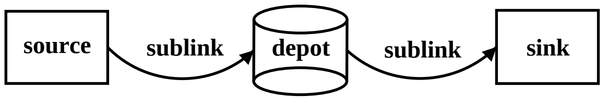

Figure 1. LSL connection illustration

The architecture of the Logistical Session Layer is relatively straightforward. Each session begins with a header containing a 128-bit session identifier. The header also includes a source and destination IP address (version 4 currently) and 16-bit port number. Additionally, the header contains 16-bit Version and Type fields to allow for future modification of the header format. Finally, there is a header length field, as the size of the header will vary when it contains options.

A few header options are currently defined. One is a header option to form a synchronous application-layer multicast tree for data staging (described in [33].) This paper, however, focuses on the case in which the *point to point* path through the network uses some number of depots. This path could be specified with a “loose source route” – an initiator-specified path through some number of session layer routers or via forwarding tables in the depots themselves. These header options are similar to the “loose source route” (LSRR) option in IP.

The connection from source to sink can transit one or more session routing processes (depots) and make use of multiple TCP “sublinks” as depicted in Figure 1. This manner of using multiple TCP connections can be thought of as “serial” rather than “parallel” sockets. For purposes of this discussion, we assume that all connections occur synchronously, i.e. the sender and receiver exist at the same time. We note that an asynchronous session

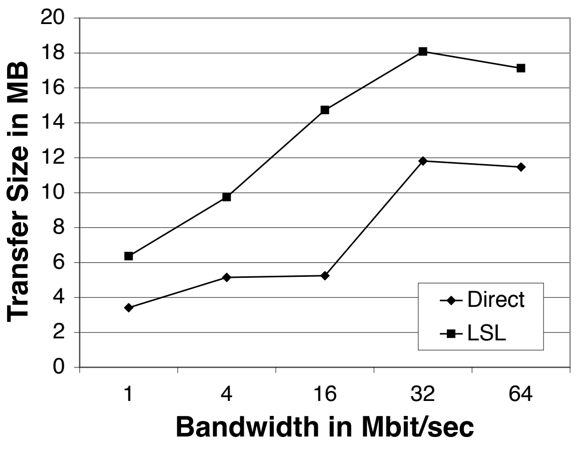

Figure 2. Data transfers from UCSB to UIUC (1MB - 64MB)

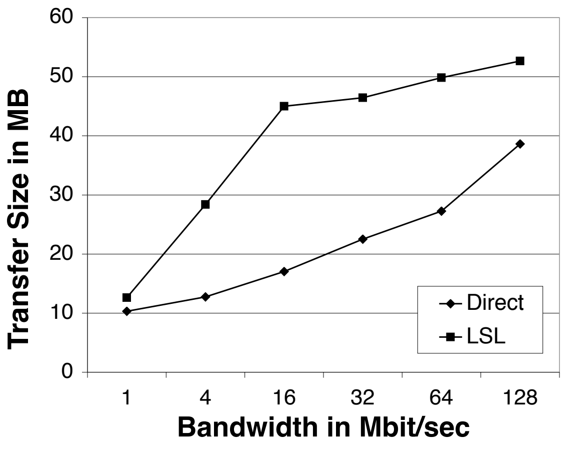

Figure 3. Data transfers from UCSB to UF (1MB - 128MB)

is possible with the receiver discovering the session identifier and reading the data from the last depot.

One significant benefit of this approach is that it has predictable impact on the network. The stability and fairness are known as the system relies on TCP connections between depots. The impact on the network is not in question and the system is safe for incremental deployment.

3 The Logistical Effect

Our previous work [32] has demonstrated the "logistical effect", which is essentially improvement of end to end throughput by dividing a connection into a series of shorter, better performing connections. In order to generalize our approach, we must evaluate the causes underlying the logistical effect. The goal is to develop an automatic mechanism for scheduling flows of data through cooperating depots. This section provides a conceptual overview of the features of network protocols that give rise to our observations and an empirical evaluation of the logistical effect.

The performance improvement offered by segmenting the end to end connection is due to the behavior of the dominant transport protocol in the Internet – TCP. Aside from implementing reliability via data retransmission, TCP controls the speed at which data is sent into the network in order to avoid causing, or to respond to, congestion. Despite the tremendous success of TCP, there are performance problems that are simply endemic to its congestion control mechanism [2]. This is particularly the case over networks with high bandwidth and long latency (commonly described as having a large "bandwidth/delay product" or BDP.) As the BDP grows, so does the amount of data that TCP must keep in transmission at any time – effectively distending the control loop. It has been observed repeatedly [11,18,22] that TCP's performance varies inversely with the end to end delay.

To perform our tests, we deployed user-level depot processes that implement the LSL protocol. These depots were located on general-purpose systems with single network interfaces. The depots were located in Denver, CO and Houston, TX, in order to be near the Internet2's Abilene [1] network "point of presence" (POP) in those cities. This work presents data from two paths. The first is from the University of California, Santa Barbara (UCSB) to the University of Illinois, Champaign-Urbana (UIUC) and the other is from UCSB to the University of Florida (UF). All machines in these tests were running Linux 2.4 kernels and configured with 8 Mbyte TCP buffers with the *setsockopt* system call. Tests of various-sized transfers were run over these two configurations. In each case, 10 iterations were run and the wall clock times were

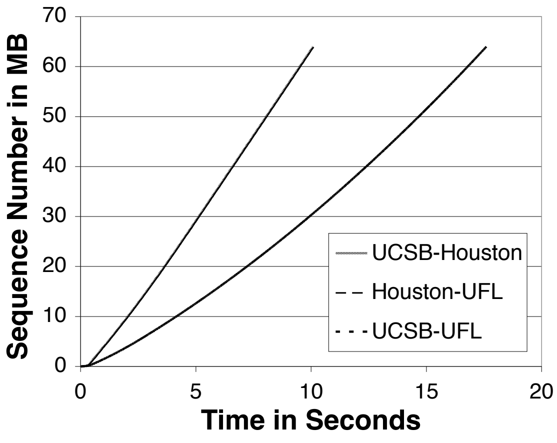

Figure 4. Average data transfered over time by acknowledged sequence number from UCSB to UF via Houston for 64Mbyte transfers.

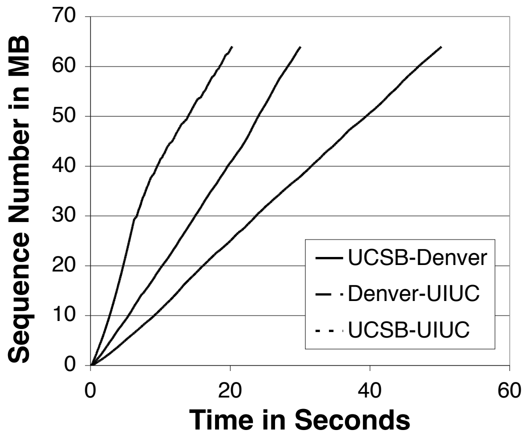

Figure 5. Average data transfered over time by acknowledged sequence number from UCSB to UIUC via Denver for 64Mbyte transfers.

recorded. In addition, we captured packet traces using the **tcpdump** utility.

The round trip times among the sites, as observed from the TCP acknowledgments in the following data, are as follows:

UCSB to UF	87ms
UCSB to Houston	68ms
Houston to UF	34ms
UCSB to UIUC	70ms
UCSB to Denver	46ms
Denver to UIUC	45ms

Figures 2 and 3 show the bandwidth observed for transfers from UCSB to UIUC and from UCSB to UF, respectively. The maximum test sizes were chosen to present the sizes after which no further speedup occurred. It is apparent from these tests that the connections segmented by the depot reach higher bandwidths with smaller transfer sizes. The largest transfers in each case are effectively the "steady state" of this configuration; the performance for tests with larger transfer sizes did not improve. Since each TCP connection was faced with a smaller RTT, the they were both able to perform better. This validates the assumption that TCP performance varies inversely with RTT.

Next, we examine the packet-level behavior of these transfers. To visualize the dynamics of each TCP connection, let us consider a technique that is commonly used for this purpose – the growth of the sequence number over time. The sequence number identifies the logical byte number in a stream of data. By plotting the highest acknowledged sequence number, we can display the actual progress of a connection – data that has been received and acknowledged. We can see from graphs such as these how quickly transfers take place and see the effects of RTT in the slope of the curve.

For each of the cases presented above, packet traces were gathered from the sender(s). We have normalized the sequence number by subtracting the random initial sequence number so that the relative growth of the TCP window over the various iterations could be averaged. Figures 4 and 5 show the growth of the sequence number over time averaged over 10 tests.

Figure 4 shows 64MByte transfers from UCSB to UF. We note that the slopes of subflow 1 and subflow 2 are very close together implying that subpath 1 (UCSB to Houston) was the bottleneck rather than subpath 2 (Houston to UF). In other words, subpath 2 was able to carry all the load that was presented to it. In contrast, observe that in Figure 5, sublink 2 (Denver to UIUC) is the limiting factor for the end to end communication. The growth of the sublink 1 curve up to 32MBytes is very fast. At the 32 MByte mark, however, the slope changes to roughly match that of the sublink 2 plot. This is due to the fact that the depot offers 32 Mbytes of total buffers. In the depot at Denver, there are 8MB kernel buffers for the sending and receiving connections and additional buffers in user space matching each of those buffers. Our depot internally allocates *send_buffer* + *receive_buffer* bytes of storage. That number is readily apparent in Figure 5. In other words, when the buffer pipeline was filled, the source had to wait until buffers were freed to send. Sublink 2 was clearly the bottleneck in this example.

4 Scheduling Approach

We hypothesize that we can automatically optimize data movement through the use of temporary storage depots in the network. Between each depot, source, and sink there is some capacity for data transfer. Our scheduling goal is to transfer data as quickly as possible from source to sink via some set of depots if necessary. The objective of this work is simply to minimize the time to move data (although more complex scheduling models could consider resource cost, utilization, etc.) This section discusses scheduling approaches for automatically reducing data transfer time using our layer of middleware, i.e. with logistical depots.

Our scheduling approach is to treat the resources of the Grid as a graph of cooperating elements. By representing the relationships between those elements in terms of a graph, we have an easy way to map our scheduling goals onto an optimization framework. We can address data movement as finding an optimal path through the vertices of the graph (i.e. one consisting of shorter, better-performing edges.) Our approach involves producing schedules based on recent network information. Thus, our algorithms must run quickly as they will be evaluated frequently.

Our notion of recent network information is a graph with node to node data transfer time as the "cost" of an edge. The graphs we consider are fully connected, as most Internet hosts can talk to most other Internet hosts with some achievable bandwidth. Thus, our approach is to begin with the performance matrix rather than the physical topology. This matrix is generated from Network Weather Service (NWS) [36] forecasts using aggregation techniques described in [34].

In order to evaluate the cost of a path, we need to know the time that it will take to transmit data between two nodes. Our approach is simply to convert measures of bandwidth between hosts to "transfer time" estimates by considering $1/bandwidth$ as the weight of an edge. We note that our input need not represent the bandwidth available to long-lived, aggressive flows, but need simply be an order preserving metric.

The time that it takes to transfer data down some path from source to sink is not the sum of the times of each edge of the link unless each node buffers the entire data stream. Flow of data along a path is *pipelined* as demonstrated by the packet traces presented in Section 3. Once the pipeline startup overhead is amortized, the end to end performance is dominated by the the performance of the slowest link.

Due to our pipelined use of depots, we have observed that the time to transfer data along a path is dominated by the link that takes the most time. Perhaps a more intuitive description is that the achievable bandwidth is limited by the link with the least bandwidth. Note that this includes the bandwidth *through* the host, although our current approach ignores this. Thus without loss of generality, the cost of a path is defined as $max(cost(i,j) \mid (i,j) \in P(source \rightsquigarrow sink))$ – that of the maximum-valued edge. The problem of optimizing for this notion of cost is called Minimax, as it deals with *mini*mization of the *max*imum-weight edge in a path.

4.1 Minimax Tree Building for point-to-point forwarding

In order to compute the best path through the network for a logistical flow of data, we need a path from source to destination. This problem is similar to what is known as the the Shortest Path (SP) problem in that the solution involves the minimum cost path. We refer to this case as the Minimax Path (MMP) and employ a greedy, tree-building algorithm that produces optimal results [10]. This al-

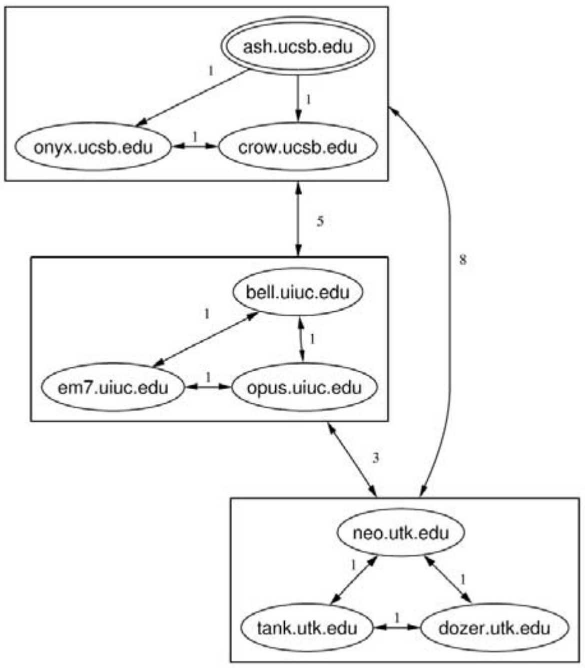

Figure 6. Hypothetical beginning graph for data movement originating at node $ash.ucsb.edu$ with edges coalesced for readability. The boxes simply indicate Internet sites. All edges are present in our model; we have merely simplified this graph for readability

gorithm is similar to Dijkstra's SP algorithm, only differing in the evaluation of path cost. It proceeds by building a tree of "best" paths through the graph and has running time of $O(NlogN)$ for implementations that keep the edges sorted. Once the tree of best paths is constructed, we can walk the tree to each destination to determine the route through the network that a session should utilize.

Figure 6 depicts an example graph to which we will refer during the discussion of our scheduling approach. The node groups (delimited with boxes) represent hosts at the same Internet site. The graphs that we consider are fully connected, but we have coalesced the edges between sites for readability. For the discussion that follows, we consider a directed graph G with vertices and edges: $G = (V, E)$. Each edge has a weight or cost c_{ij} for each $(i, j) \in E$. The MMP from s to t for $s, t \in V$ is the set of edges $S \subset E$ where $s \rightsquigarrow t$ with minimum cost (evaluated as $max(c(i, j))$.) Our algorithm builds a tree of paths from a specified start node (v_{start}) to all other nodes in V.

4.1.1 Tree Shaping with Edge Equivalence

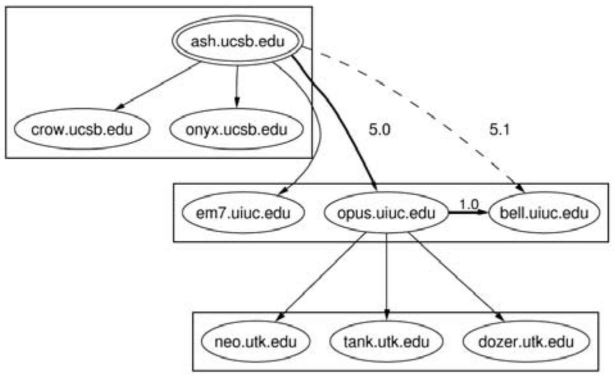

Figure 7. MMP Tree from $ash.ucsb.edu$ to all other hosts. The path to $bell.uiuc.edu$ is lengthened due to the marginal difference in edge costs.

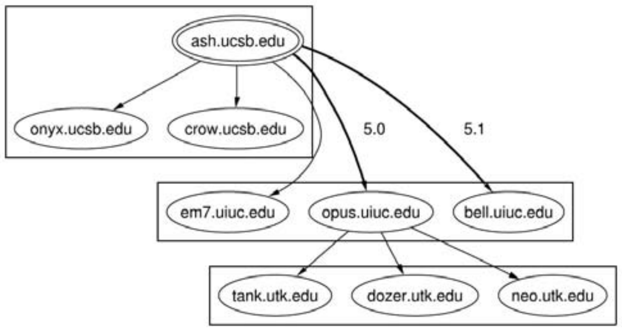

Figure 8. Revised MMP Tree from $ash.ucsb.edu$ to all other hosts. With $\epsilon = 0.1$ these edges are considered the same.

Due to slight variations in performance measurements taken over time, the paths produced by the above algorithm are more complex than they need to be. While our MMP algorithm provides a correct solution based on its input, we observe that this is not exactly what is required. Even in quiet environments, measurements of almost any aspect of computer performance can vary slightly from moment to moment. It is not desirable for hosts with functionally identical connectivity to be viewed as different. Hosts on the Grid are often located in tightly-coupled clusters with a shared connection to the world. In the case of the wide-area Internet, all hosts at a single “site” are connected similarly to all hosts at some other “site.” Thus, we turn to the modifications to our base MMP algorithm that are necessary to address these facts in the face of slightly different edge values.

To simplify our constructed trees, we would like to consider edges within some ϵ of one another as equivalent. Consider the graph in Figure 7, which depicts an MMP tree originating at the node labeled $ash.ucsb.edu$. These nodes represent Internet hosts and their “site” is the last two components of their name. Strictly speaking, the correct MMP from $ash.ucsb.edu \rightarrow bell.uiuc.edu$ is the dotted edge with cost 5.1. However, all machines at UCSB traverse the same network connections to UIUC and it is unlikely that $opus.uiuc.edu$ is significantly better connected than $bell.uiuc.edu$. With $\epsilon = 0.1$, these values are considered the same, producing the simplified MMP tree depicted in Figure 8.

Our edge equivalence approach consistently builds more appropriate trees. The edge equivalence ϵ makes tree building more conservative and serves to dampen adding unnecessary edges – that is, those with no significant improvement over the edges already selected. Appendix A provides pseudo-code for the algorithm we used. An automatic method of choosing ϵ would be very desirable. Prediction error from the NWS and variance of the measurement set are potentially good candidates for ϵ.

4.2 Scheduling Results

To test our point to point scheduling technique, we chose a testbed that offers complexity and network diversity in the hopes that it would provide opportunities for logistical scheduling. Planetlab [24, 25] is a multi-institution research project dedicated to providing a platform for the development of “planetary-scale services.” This environment is fairly representative of the distributed nature of Grids as there are a large number of well-connected sites, although each site has only one to three machines.

The scheduling system takes a fully-connected map of the network as its graph and produces a path tree from each node to all others. For hop by hop routing, the MMP tree is reduced to a list of destinations and the next hop along the chosen path. These destination/next hop tuples form a “route table” that is consumed by the logistical depot and used to control forwarding. The ϵ values were computed as 10% of the value of the edge; that is, if the evaluated edge was not 10% better than the previous edge, then it was not added to the path. We have not evaluated the choice of ϵ, noting that clusters coalesced around 10% and higher values did little to alter the generated schedules.

To test the efficacy of the routes chosen, we utilized a pseudo-random test generator. We implemented a mechanism that requests a depot to generate some amount of arbitrary data. Also, each depot was made to spawn a thread that initiated transfers to a random depot. Thus, in the experiments, each host could act as a source, sink or depot. To test a range of sizes (analogous to the tests presented in Section 3) we choose a random size as 2^n megabytes for $0 \leq n < 7$. The test logic chose direct routing or LSL scheduled forwarding randomly and initiated and measured the resulting data transfer.

This ultimately generated a set of routed and direct measurements for the cases in which the scheduler identified a better path. For each case in the test set, there are multiple measurements of each size, both direct and scheduled. In this way, we can evaluate the aggregate effects of scheduling rather than focusing on individual paths.

We had a pool of 142 machines in the set. The scheduler identified better routes via depots for 26% of the total number of paths in the system. Only routes where the scheduler chose to use depots were measured. We produced 362,895 total measurements for those paths in which alternatives could be compared. Figure 9 shows the speedup of scheduled transfers over direct transfers by comparing the average observed bandwidth per case (source,destination,size) and defining the speedup of that case as:

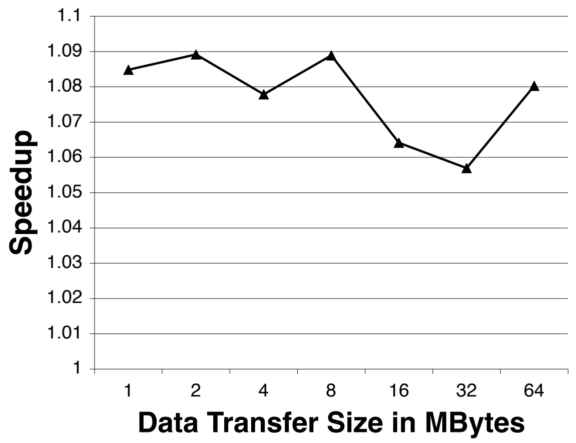

Figure 9. Average speedup per transfer size over all host pairs.

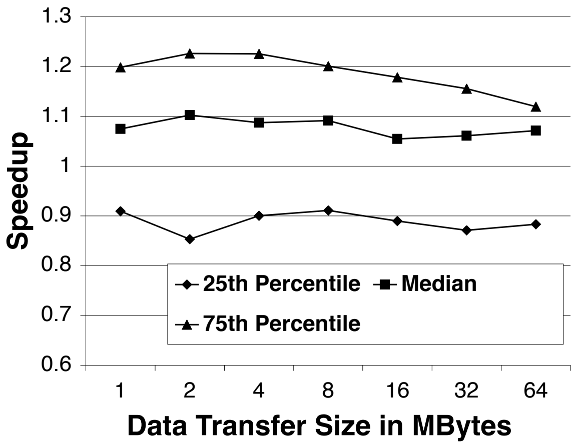

Figure 10. Median, 25th and 75th percentile of absolute speedup per transfer size over all host pairs.

$$speedup = \frac{average\_scheduled\_bandwidth}{average\_direct\_bandwidth} \quad (1)$$

These results show between 5.75% and 9% speedup for transfers of various sizes. Figure 10 shows the median, 25th and 75th percentile values in order to show the variance in the observed speedups. Taken together, these demonstrate that we were able to offer acceptable speedup in many cases, but there are quite a few cases in which we failed and actually caused worse performance. Looking at the individual measurements and the averages of direct and LSL transfers between host pairs, we see quite a lot of variance. There are cases where performance is improved by a factor of four and cases where using LSL causes performance to suffer. The following table shows the percentile where the speedup becomes greater than 1.

1M	2M	4M	8M	16M	32M	64M
39	43	48	43	48	46	49

There are a number of factors causing the performance to suffer in so many cases. First, it is important to note that Planetlab hosts are configured with *extremely* small TCP buffers for high-performance, wide-area use (64KBytes) and these could not be modified by user-level processes. Previous work has demonstrated that LSL can help in buffer-limited situations, but even then the performance improvement remains small [32].

While the Planetlab nodes are widely distributed they are, for the most part, located at university sites and not "in the network." LSL depots would serve best if located near the core of the network as opposed to at the leaves. Thus, any speedup in the first case is noteworthy since this represents a primarily peer-to-peer mode of operation [1].

Planetlab is a much used resource and each host is often running many other communicating processes. To share the resource fairly, each user is presented with a somewhat virtualized machine. This virtualization decreases the bandwidth *through* the nodes, particularly in the face of significant load.

To address the previous points, we conducted a smaller set of experiments with additional constraints. For these tests, we employed Planetlab hosts at 10 U.S universities that are connected to Abilene. Rather than use Planetlab nodes as depots, however, we used depots running on hosts in the Abilene POPs (as part of the Internet2 Observatory

[1]Planetlab now has nodes located at backbone sites.

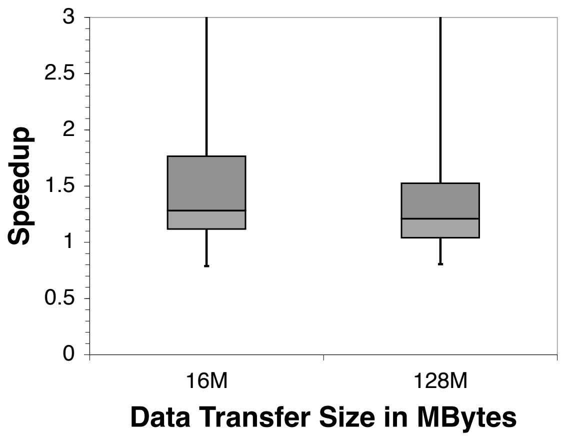

Figure 11. Min, Max, Median, 25th and 75th percentile of absolute speedup per transfer size over all host pairs. The line in the box is the median and its upper and lower edges are the quartiles. The lower "whisker" indicates the minimum value and the upper one (not shown) is the maximum value.

facility `http://abilene.internet2.edu/observatory/`). Note that we didn't need to explicitly specify that these depots be used. The output of the algorithm correctly identified paths using the "core" nodes as preferable.

We gathered 10 16MB measurements for each of the direct and LSL scheduled cases and 5 measurements for each case with transfers of 128MB. Figure 11 shows the results in terms of the speedup and depicts the minimum, 25th percentile, median, 75th percentile and maximum speedups observed for the two test transfer sizes. The maximum speedup was 10.15 for 16MB and 6.38 for 128MB, but the graph has been truncated to show more details of the distribution.

We observe that in the cases where the performance failed to improve we should have avoided using LSL at all. This points to modifications in our scheduling technique. The general scheduling mechanism is valid – the performance of a series of links *must be* dominated by the least capable link. What is at issue is the degree to which we take into account other factors that could affect the outcome. For instance, some nodes in the Planetlab testbed have been explicitly rate-limited with respect to their bandwidth utilization. That is, in some cases we were experiencing administrative, rather than technical, limits on achievable bandwidth.

Again, none of these suggest changing the algorithm itself, but rather suggest modifications to the algorithm's input. In our arrangement, the bandwidth *through* the host was not accounted for. Note that an administrative limitation that changes its behavior after a certain amount of traffic has been passed will be difficult to model, but it seems to be necessary.

Additionally, the frequency with which the algorithm can consider current network information, and its sensitivity to it, are key issues with broader use of this approach. In the first experiment, the scheduler was re-run at 5 minute intervals and was based on relatively current information. For the second experiment, it was run only initially and and was based on relatively static information.

5 Related Work

Our approach is similar to recent work in application level routing (or overlay networks) and non-default route selection [35, 29, 3, 27]. That work has addressed a number of issues including route asymmetry and optimal, or parallel, route selection. The benefits of retransmission from strategic locations for reliable multicast has been observed as well [15]. LSL differs in that it is presented as the addition of new higher-layer functionality, rather than a workaround for ineffective routing policy. Indeed in the examples that we present here, the "default" route has not been changed in any significant way (only insofar as necessary to model a general-purpose depot.) Specifically, we don't use this system to "route around congestion" but rather note that some of the effects observed in other work may complement aspects of the logistical effects that we observe. In the LSL approach, no persistent tunnels exist between depots; the session layer connection is created dynamically. This is unlike most overlay networks

Our approach is quite similar to recent work by

Malouch, et al. [19], which treats multicast proxies as nodes in a network optimization problem. We note that their arc incidence constraints are different than those that we propose. Further, their simulations were aimed at evaluating various heuristics, while our goal is to understand the performance improvements from simple scheduling in real networks.

There is an increasing interest in drastic modifications to end to end signaling in certain scenarios. Approaches such as Delay-Tolerant Networking (DTN) are being investigated for extra-planetary data transmission [9]. This work bundles data for asynchronous transmission in environments where end to end signaling is unreasonable. However, the same issues are beginning to be discussed for terrestrial networks. The Internet Indirection Infrastructure [31] also removes direct end to end communication with indirection points that are similar to our notion of depots. Our approach is also similar to the Performance Enhancing Proxy [7] (PEP) techniques described by the IETF's Performance Implications of Link Characteristics (PILC) working group.

Also related are projects designed to increase bandwidth available to distributed applications. The PSockets [30] work has spawned a great deal of interest in using multiple TCP sockets in parallel to increase throughput. The notion that the connection bundle is something more general than binding from file handle to transport layer is similar to the session layer abstraction that we describe. However, that work is focused on an application-level solution rather than "in the network" support for general mechanisms.

Evolution of TCP has been proposed with variants such as Vegas [8]. Net100 [23] investigates resource-allocation "rightsizing" to reduce network buffer over- or under-allocation and mechanisms to mitigate slow start by persistent connection statistics. XCP [17] proposes an alternative to TCP that addresses many of the problems we highlight. FAST TCP [16] and HighSpeed TCP [12] are recently-introduced variants that attempt to further evolve TCP and address high bandwidth/delay networks. Our approach can only benefit from this work and from improvement in TCP. The key to understanding this is that TCP depends on packet acknowledgment for control. When this control loop is larger, TCP simply must take longer to react. The difference in the nature of the end to end flow control will always be present regardless of the specifics of the TCP implementation.

Techniques developed for wireless networks [5, 4] seek to mitigate the cost of retransmission in lossy environments by performing retransmits from intermediate points in the network. However, they violate layering to do so. Systems to proxy TCP have been developed with the same goals in mind. One example is TPOT [28], which alters TCP (in an incompatible way) to allow this mode of operation. These approaches are targeted at cases where network subnet characteristics vary. Our approach is similar, but slightly different, in that it addresses cases where the network characteristics are not that different. Additionally, we provide a general framework in which these approaches can be used rather than applying them in an ad-hoc, per-application manner.

The reduction in CPU utilization stemming from segmented or cascaded TCP connections has been explored as well [20]. MSOCKS [21] is conceptually similar to LSL in that it uses segmented TCP to facilitate mobility. LSL differs in that some of the same benefits can be realized without violating the separation of functionality between protocol layers, and that the services must be explicitly requested by the program using them – there is no "transparency" expressed or implied. Reduction in CPU utilization on compute elements by offloading communication to depots could have great value in Grid environments. We intend to quantify this in future work.

6 Conclusion

This paper has demonstrated that it is possible to automatically schedule logistical forwarding paths that improve end to end throughput. In order to do so, we evaluated the underlying causes of the logistical effect, and formulated a scheduling approach based on our empirical observations. Next we proposed an optimal framework for determining paths based on observed network performance. Then, we discussed modifications to that algorithm that produce more appropriate solutions. Finally, we provided results from a large-scale empirical test of the scheduling and forwarding framework.

These results demonstrate the validity of our approach to scheduling data movement with LSL. There are still many open issues. The scheduling algorithms can be trivially extended to include the path *through* the host as another edge whose bandwidth must be taken into account. Yet, we must consider the scalability of host-based forwarding. This problem is surmountable with use of special-purpose hardware in the "depot" nodes and of session negotiation that allows a potential depot to refuse a new connection based on host load. This is an area for future research.

7 Acknowledgments

Much of this paper is based on thesis work performed under the supervision of Rich Wolksi. The author wishes to acknowledge his thoughtful input, adept guidance, and unfailing support. Additionally, the author thanks the members of the LoCI Laboratory at the University of Tennessee for their support and collaboration over the years. Finally, we thank the reviewers for their critical analysis and valuable feedback.

References

[1] Abilene. `http://www.ucaid.edu/abilene`.

[2] Allman, Paxson, and et al. TCP congestion conrol. RFC 2581, April 1999.

[3] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. The case for resilient overlay networks. In *8th Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[4] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts, 1995.

[5] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4), 1995.

[6] M. Beck, T. Moore, J. Plank, and M. Swany. Logistical networking: Sharing more than the wires. In *Proc. of 2nd Annual Workshop on Active Middleware Services*, August 2000.

[7] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. RFC 3135, June 2001.

[8] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[9] V. G. Cerf, S. C. Burleigh, A. J. Hooke, L. Torgerson, R. C. Durst, K. L. Scott, K. Fall, and H. S. Weiss. Delay-tolerant network architecture. Internet Draft draft-irtf-dtnrg-arch-02.txt, March 2003.

[10] T. H. Cormen, C. L. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[11] S. Floyd. Connections with multiple congested gateways in packet-switched networks part1: One-way traffic. Computer Communication Review, V.21 N.5, October 1991.

[12] S. Floyd. HighSpeed TCP for large congestion windows. Internet Engineering Task Force, INTERNET-DRAFT, draft-ietf-tsvwg-highspeed-01.txt, 2003.

[13] A. N. S. for Information Processing Systems. Open systems interconnection – basic connection oriented session protocol specification. ANSI/ISO 8327-1987, 1992.

[14] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, October 2000.

[16] C. Jin, D. Wei, and S. Low. Fast tcp for high-speed long-distance networks. Internet Engineering Task Force, INTERNET-DRAFT, draft-jin-wei-low-tcp-fast-01, 2003.

[17] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for future high bandwidth-delay product environments, 2002.

[18] T. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss, 1997.

[19] N. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A graph theoretic approach to bounding delay in proxy-assisted. In 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'02), May 2002. 143, 2002.

[20] D. Maltz and P. Bhagwat. Tcp splicing for application layer proxy performance, 1998.

[21] D. A. Maltz and P. Bhagwat. MSOCKS: An architecture for transport layer mobility. In *INFOCOM (3)*, pages 1037–1045, 1998.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion

avoidance algorithm. Computer Communications Review, 27(3), July 1997., 1997.

[23] Net100. `www.net100.org`.

[24] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet, 2002.

[25] Planetlab. `http://www.planet-lab.org`.

[26] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, September/October 2001.

[27] N. Rao. Netlets: End-to-end QoS mechanisms for distributed computing over internet using two-paths. Int. Conf. on Internet Computing, 2001.

[28] P. Rodriguez, S. Sibal, and O. Spatscheck. TPOT: Translucent proxying of TCP, 2000.

[29] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of internet path selection. In *SIGCOMM*, pages 289–299, 1999.

[30] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. SC2000, Nov 2000.

[31] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM 2002*, pages pages 73–88, August 2002.

[32] M. Swany and R. Wolski. Data logistics in network computing: The Logistical Session Layer. In *IEEE Network Computing and Applications*, October 2001.

[33] M. Swany and R. Wolski. Network scheduling for computational grid environments. In *Adaptive Grid Middleware '03 (in association with PACT)*, September 2003.

[34] M. Swany and R. Wolski. Building performance topologies for computational grids. *International Journal of High Performance Computing Applications*, 18(2):255–265, 2004.

[35] J. Touch. The XBone. Workshop on Research Directions for the Next Generation Internet, May 1997.

[36] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, January 1998.

A Minimax Tree Algorithm

```
Input: nodes in V, edges in E,
    Start_node, Epsilon

Create V_MMP, and two arrays with an element for
    each member of V:
    - parent[] init to -1
    - cost[] init to MAX

Designate Start_node as new_node, set
    cost[new_node]=0, parent[new_node] = new_node
    and move new_node to V_MMP

while(nodes left in V) {

  for each edge in E from new_node to other_end {

     relax_cost = max( E[new_node,other_end],
         cost[new_node])

     if( ( relax_cost * (1 + Epilson) ) <
         cost[other_end] )
     then {

         parent[other_end] = new_node
         cost[other_end]= max( E[new_node,other_end],
             cost[new_node])

     }

  }

  select node in V with lowest value in cost[]
  set new_node to that node and move it to V_MMP

}

Return V_MMP
```