Improving Throughput with Cascaded TCP Connections: the Logistical Session Layer
UCSB Technical Report 2002-24

Martin Swany and Rich Wolski

***Abstract*—**

This paper describes a system designed to improve network performance and functionality in a way that is applicable from high-performance distributed computing to wireless and mobile networking. Our architecture captures the benefits of application-level routing while being designed as a general-purpose network service. We call our abstraction the Logistical Session Layer as it allows decisions about a flow's path through the network to be made based on what can be thought of as network logistics. The "session" layer affords a conversation based on multiple transport-layer hops. We empirically examine the performance implications of the "cascaded" TCP connections that our approach facilitates.

I. Introduction

THE landscape of computing and networking is changing. On the one hand, Computational Grid environments [1] demand high performance networking as large, distributed systems use the network like never before. On the other hand, lightweight edge devices are proliferating and driving evolution of the network in a different manner.

Network capacities are growing at an amazing rate. However for the most part, that increased capacity is realized in link aggregation and not in individual flows. For the Transmission Control Protocol [2] (TCP), congestion control mechanisms provide vital network-wide stability and fairness properties at the potential expense of individual end-to-end throughput. Moreover, the proliferation of high-speed connectivity exacerbates this cost since the overhead associated with this congestion control becomes more appreciable – particularly when the network is overprovisioned. At the same time, more and more devices are connected to the Internet via wireless mechanisms, so the trend is toward a broadening range of network technologies (having differing performance characteristics) with which end-to-end protocols such as TCP must contend.

In this work, we present a novel approach to *optimizing end-to-end throughput* by mitigating the cost of end to end congestion control while being, by definition, "TCP friendly." We detail our approach in terms of the implementation and performance of a new "session" layer [3] — the **Logistical Session Layer** (**LSL**) — that can use TCP for transport. Logistics refers to the practice of moving things from place to place while making best use of available resources according to some objective function (e.g. lowest cost or shortest time). We define the term "Logistical Networking" to describe the application of these principles to improving performance and functionality in network settings [4]. A Logistical Network is one in which "depots" located throughout the network can provide application-controlled buffering (via a complete storage hierarchy) to potentially anonymous clients. Whereas other work in this area focuses on larger, longer-lived storage allocations [5], LSL uses small, short-lived intermediate buffers to improve end-to-end performance between communicating hosts. The protocol constructs a *session* consisting of multiple sequential TCP/IP streams "cascaded" from source to sink through intermediate LSL storage depots. Because TCP's control mechanisms continue to govern packet traffic, the underlying network performance remains stable. At the same time, *despite the additional transport level processing* and buffer-copying overhead at each depot, our results show that *LSL can increase end-to-end throughput by an average of 40% and as much as 75%* in a variety of network settings.

Perhaps it is initially counter-intuitive that intermediate buffering would provide performance gains in this range, and that it would do so for high-performance networks as well as those with wireless components. Further, this prototype is implemented as an unprivileged process on a

Department of Computer Science, University of California at Santa Barbara.

Authors' Email: {swany,rich}@cs.ucsb.edu

general-purpose system, so it represents a worst-case scenario in some sense. This paper investigates this potential performance improvement, and we will

- describe a prototype LSL implementation we have developed to investigate the efficacy of end-to-end throughput optimization,
- quantify the throughput effects empirically, including all concomitant processing overheads, and
- explain, through packet trace analysis, the observed LSL effect for different network configurations.

Taken together, our results show that judicious intermediate buffering can dramatically improve throughput in a broad spectrum of network settings. At the same time, our approach is intended to remain consistent with current and future Internet protocol technologies in general and TCP congestion control in particular.

II. Related Work

While our design for LSL [6] is novel, it is related to many efforts in the network community. Classically, many Internet protocols and systems have relied on hop-by-hop (rather than end-to-end) forwarding. SMTP [7] and NNTP [8] are two well-known examples. The notion that end-nodes might want to direct the path taken by a flow of packets was provided for with the loose and strict source route options to IP [9], although this has fallen into disfavor because of its (often realized) potential for abuse and the per-packet overhead associated with forwarding such traffic.

LSL is similar in spirit to recent work in application level routing (or overlay networks) and non-default route selection [10–13]. This work has addressed a number of issues including route asymmetry and optimal, or parallel, route selection. There is also a growing industry surrounding traffic tuning based on these principles. The benefits of retransmission from strategic locations for reliable multicast has been observed as well [14]. LSL differs in that it is presented as an evolution of the Internet architecture, 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 extreme 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 LSL effect that we observe.

MSOCKS [15] is conceptually similar in using segmented TCP to facilitate mobility. Link-layer retransmit and other techniques to mitigate high RTT and loss rates are discussed in RFC 3135 [16]. Techniques developed for wireless networks [17,18] seek to reduce the cost of retransmission in lossy environments. Systems to proxy TCP have been developed with the same goals in mind. One example is TPOT [19], which alters TCP to allow this mode of operation. A similar approach that targets the caching of web objects also proposes modifications to TCP [20]. Solutions that require modifications to the TCP protocol limit the potential for incremental deployment in a global Internet sense as it is often difficult to change extant protocol infrastructure. Our approach takes the form of a service layer atop standard TCP that interested clients can choose to use without modification to the underlying networking protocols.

The reduction in CPU utilization stemming from segmented or cascaded TCP have been explored as well [21]. 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.

Also related are projects designed to increase bandwidth available to distributed applications. The PSockets [22] 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.

To a certain extent, our results are based on the nature of TCP's flow control. The research community has recognizes the issues inherent in using TCP over networks with high bandwidth/delay products. There is a tremendous body of research [23–29], too vast to properly cite here, devoted to understanding and improving TCP's performance. Appealing to intuition, however, we note that there is a fundamental cost associated with buffering data for potential retransmission. Allowing data to leave a host without being acknowledged by the receiving host effectively allows an increase in the amount of *pipelining* that a network link supports. Also, TCP's performance is widely understood to depend greatly on the end to end Round Trip Time (RTT). We leverage the well understood stability and fairness qualities of TCP by utilizing it between depots. Whereas more drastic changes to TCP (such as Explicit Congestion Notification) may take time for their ramifications to be fully understood (and to be deployed ubiquitously), LSL is treading in more familiar territory.

Also note that this can be thought of as an extension of the increasing requirements for network state [30,31] with the benefit of being deployable without modification of extant software in the network. End stations are, in many cases, becoming more simple and routers in the core of the network are more and more sophisticated. It is is no longer necessary to assume that the network is inherently unreliable and should remain stateless. LSL is another example of how careful state management within the network fabric itself can improve delivered network performance while, at the same time, preserving the stability and reliability characteristics that the Internet Protocol suite provides. In addition, the architecture we have defined is compatible the current implementations of TCP/IP while offering a similar programming interface to that provided by the Unix socket abstraction.

III. Architecture

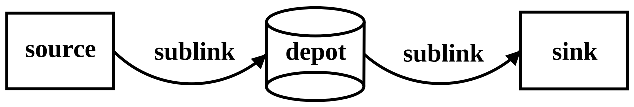

Fig. 1. LSL connection illustration

The architecture of the Logistical Session Layer is very simple. A "session" layer is logically layered atop the Transport layer. In the same way that a transport layer connection may traverse multiple network layer hops, a session layer may make use of multiple transport-layer connections. The session is described by a 128-bit session identifier. Conceptually, the ultimate sending and receiving ports need not exist at the same time, enabling a wide range of functionality. The connection from source to sink can use multiple TCP "sublinks" as depicted in Figure 1.

Currently, the path through the network is specified with a "loose source route" – an initiator-specified path through some number of session-layer routers (which we refer to as *depots*.) LSL clients and depots are assumed to have network performance information available from a system such as the Network Weather Service [32], in order to make decisions about paths. We consider the depot nodes themselves to be in a system orthogonal to a larger measurement system except to make passive performance information available via the TCP extended statistics MIB [33] or the like.

An MD5 [34] message digest over the complete stream should be sent between end-systems. We note that this will counter the potential for data corruption that is already present with TCP error detection [35]. Note that in terms of the end-to-end argument [36], we do consider the ultimate verification of data integrity to be the responsibility of the end nodes. We have simply moved away from the requirement that flow control and buffering also be considered exclusively in an end-to-end fashion.

The interface to LSL is simply BSD sockets using the {**P/A**}**F_LSL** protocol/address family (as described in [6]). However, we note that to support session-layer framing, we would need a host interception mechanism (e.g. SOCKS [37]) or a kernel-level implementation of the protocol.

While the results in this paper deal with increased throughput across a path, we envision many cases in which a session layer such as this is an appropriate abstraction. Intermittently connected devices could use the session layer to mitigate connection creation overhead and the effects of roaming (in that the ultimate server need not know of an address change). Also, when considering TCP running over IPv6, the community has yet to come to consensus about the implications of site multihoming in that a different route to the same host might entail using a different IP address. A session layer identifier would allow us to leave the semantics of TCP unchanged and yet accommodate the notion that transport connections may come and go without disrupting the integrity of the session-layer handle.

In this work we focus on the performance improvements offered as the network becomes *articulated* so that TCP can adapt to local conditions more rapidly and with more specificity.

Thus LSL is

- Incrementally deployable – no changes are required to extant hardware and protocols
- Voluntarily utilized – no need for automatic "interception", easy to implement in existing software, and can be employed selectively
- TCP compatible – by definition cooperates with TCP and can track and benefit from its evolution
- Throughput optimizing – improves performance in certain cases.

IV. Results

Next we deal with the performance implications of a system such as LSL, focusing on the synchronous connection case. This section will present measurements from our prototype of the system. Section V will demonstrate some of the underlying reasons for the improvement in performance.

A. *Experimental Method*

As described in previous work [6], we have implemented a simple session layer forwarding processes (called *lsd*) and a library that is used by a client and a server. The daemon runs without privileges – it is a user-level process. For this set of experiments, we focused on synchronous, connection-oriented streams from end-system to end-system. So, the *lsd* process very simply establishes a transport to transport binding based on the LSL header information.

All machines in this study were running the Linux operating system, with kernel version 2.4.x, although previous work has validated the effects across a range of systems. The machines at both ends supported large windows and were configured with 8 MByte TCP buffers for the exercised direction. Earlier work [6] noted that the performance improvements are more profound in the case of limited buffers at the end nodes as might be the case with lightweight mobile devices. The tests took place at sites connected to the Abilene [38] network and all wide area transfers traversed it.

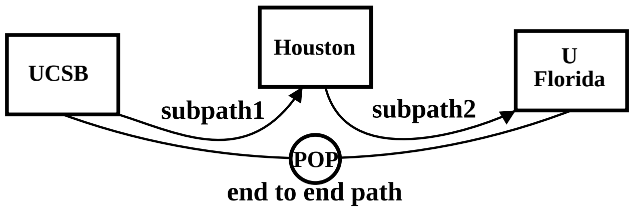

Fig. 2. Diagram of Experiment Configuration. The depot was chosen for its proximity to a POP on the default path.

It is important to note that in no case did we fail to obey the Acceptable Use Policy (AUP) of any network traversed. Previously [6], we described a case in which the routing configuration at two autonomous systems did not provide a symmetric route and we note that even in this case, our path did not make unacceptable use of any network. However, for this set of experiments, we do not even alter the default path through the network, except insofar as we insert a depot in the connection. These depots were again chosen to minimize the divergence of the LSL path from the default TCP path to model, as closely as possible, integral depots.

Figure 2 depicts the conceptual layout of some of the hosts in question. We refer to LSL "subpaths" as the (TCP) connections between depots, clients and servers. We compare this with the default end-to-end TCP connection. In the first two datasets, the depot machines are located close to Abilene POPs in Houston and Denver. In each dataset, the route between client and server traverses the intermediate gigapop (either Houston or Denver, as the case may be) so the latency being added should be minimal. Using a depot in Denver, we measured transfers from UC Santa Barbara (UCSB) to the University of Illinois (UIUC) and separately to Ohio State University (OSU). We also measured transfer performance between UCSB and the University of Florida (UF) using a depot located in Houston for the LSL communications.

We used the tcpdump program to capture packet traces from each TCP connection (whether "sublink" or end-to-end). The captures took place at the sending host in each case.

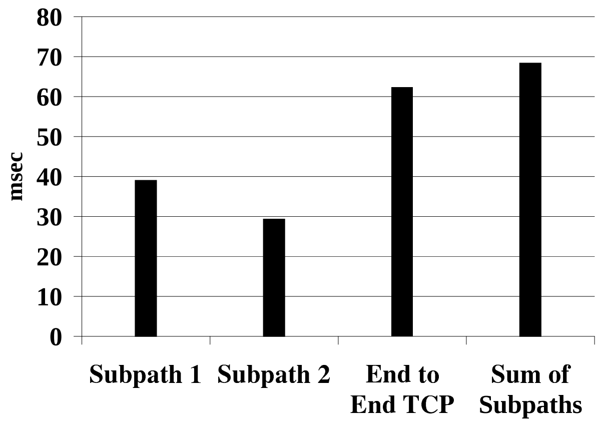

Fig. 3. Average Observed TCP Round Trip Time for Case 1.

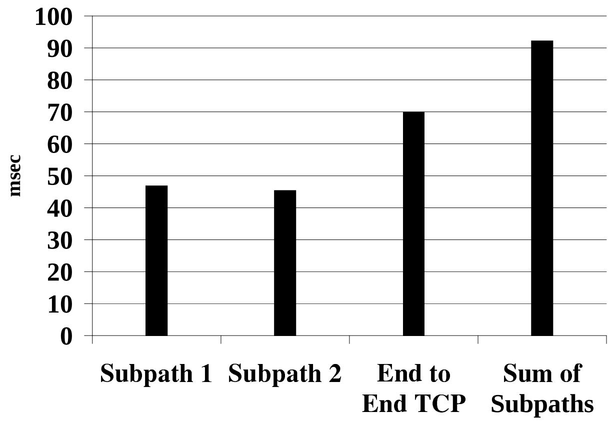

Fig. 4. Average Observed TCP Round Trip Time for Case 2.

To determine the effect of "detouring" from the direct route to the LSL depot (see Figure 2), we observed RTTs for the end-to-end link, and both sublinks. Figure 3 compares the average RTT for each subpath and the end-to-end TCP connection. The final bar in the figure shows the sum of the average RTTs for both sublinks as an indication of the overall RTT for LSL. These RTT measurements are based on TCP acknowledgments from the traces, hence they do not include the latency associated with traversing the depot itself. A more accurate measure of end-to-end RTT would take into account the intra-host latency, but the values we present do represent a lower bound. We can see in Figure 3 (via Denver), that the additional RTT for LSL is minimal at approximately 6ms. However, Figure 4 (via Houston) shows an average increase over the direct end-to-end connection of about 20ms. [1] Despite these increases in overall RTT, however, LSL improves throughput through both intermediate depot points.

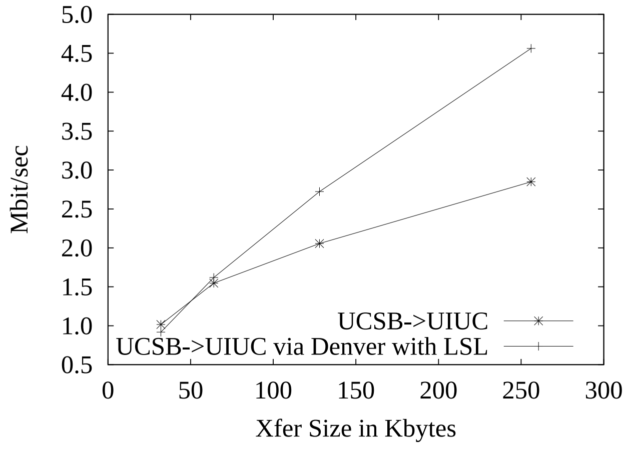

Fig. 5. Average Observed Bandwidth of data transfers from UCSB to UIUC (32K - 256K)

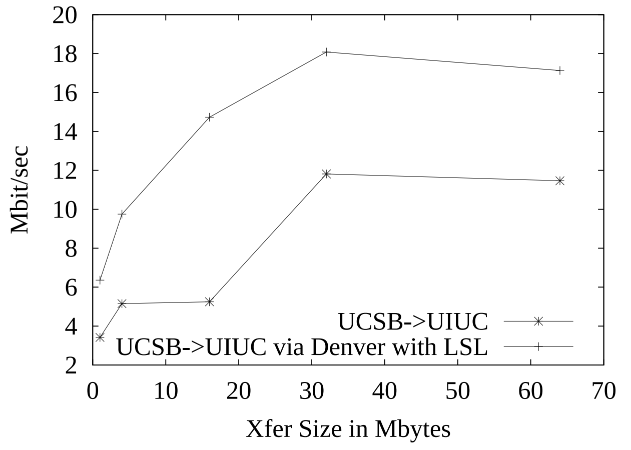

Fig. 6. Average Observed Bandwidth of data transfers from UCSB to UIUC (1M - 64M)

Tests of various-sized transfers were run over these two configurations. In these cases, 10 iterations were run and the wall clock times were recorded. That is, when measuring throughput, we did not rely on TCP packet trace timings, but rather we observed the host to host throughput empirically so as to include *all* additional overheads associated with traversing the relevant intermediate depot. Figure 5 shows the average throughput for small transfers from UCSB to UIUC (via Denver). For the smallest

[1]For the Houston depot the latency difference measured by *ping* information shows an approximate additional latency under 2ms, so much of the 20ms average is load induced and likely related to the transfer time through the host. At the time of this writing, we did not have the data available to deconvolve the queue delay effect from the effect of detouring to the depot.

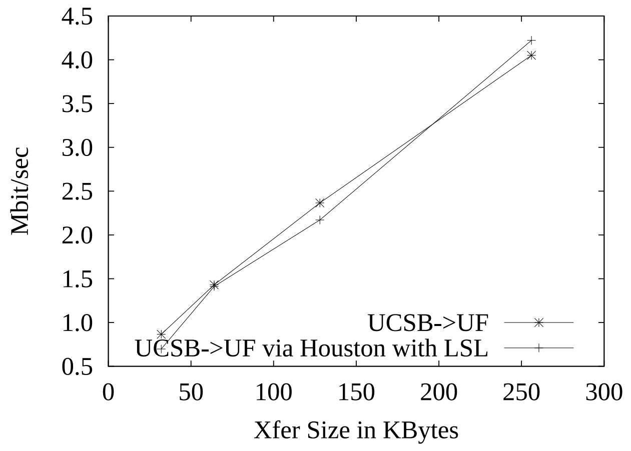

Fig. 7. Average Observed Bandwidth of data transfers from UCSB to UF (32K - 256K)

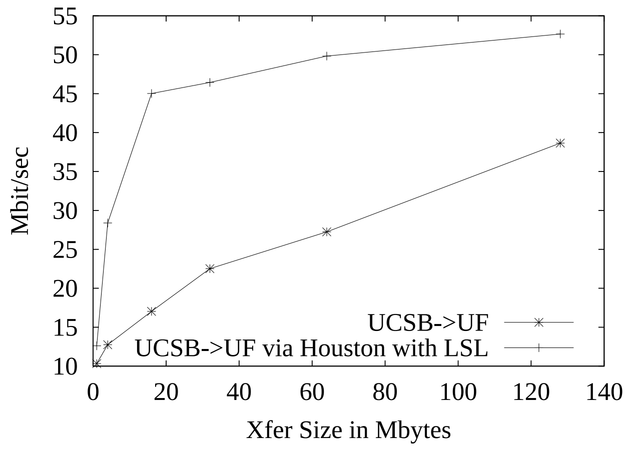

Fig. 8. Average Observed Bandwidth of data transfers from UCSB to UF (1M - 128M)

transfer sizes, the overhead of setting up two connections is too great and LSL achieves a lower throughput than the standard end-to-end transfer. However, LSL transfers begin to enjoy better performance quickly as the amount of data grows, increasing the bandwidth by about 60 percent for 256KByte transfers. Figure 6 shows larger transfers and the differences remain significant, still with an improvement of approximately 60 percent. Figure 7 shows small transfers going from UCSB to UF (via Houston). We note that for small transfers along this path the performance is roughly equivalent. Figure 8 shows larger transfers from UCSB to UF and the bandwidth using LSL again becomes significantly higher as the overhead is amortized away. Even though with LSL the one way transmission latency is longer from end system to end system, and there is additional processing overhead at the depot, the average throughput is significantly higher for large transfers for both the Denver and Houston depot locations.

We also wanted to examine the effectiveness of LSL for a lightweight, edge node connected via 802.11b wire-

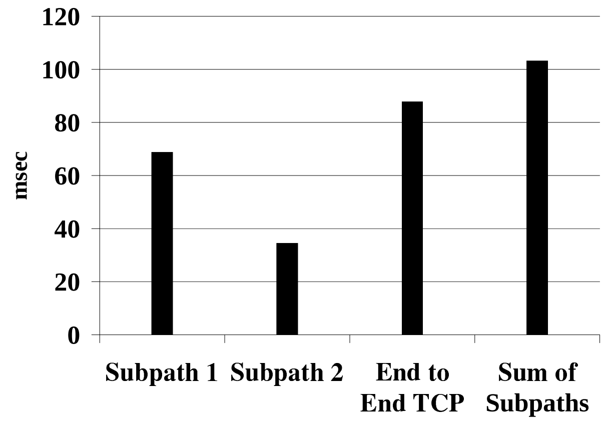

Fig. 9. Average Observed TCP Round Trip Time for Case 3.

less. In this case, the experiment was designed to model a mobile node on the edge of the network and the LSL depot was placed much closer to the client. Our intention is to use this experimental configuration to model a wireless provider with infrastructure willing to gateway LSL into TCP for users. We used end-points at the University of Tennessee, Knoxville (UTK) and UCSB, but the UCSB end-point was connected to the campus network infrastructure by an 802.11b wireless network. We located the LSL depot on a machine at UCSB, near the "edge" of the wired network. Figure 9 again shows the average observed RTTs. Note that the average RTT value is quite high on on sublink 1 – the UTK to UCSB wired sublink.

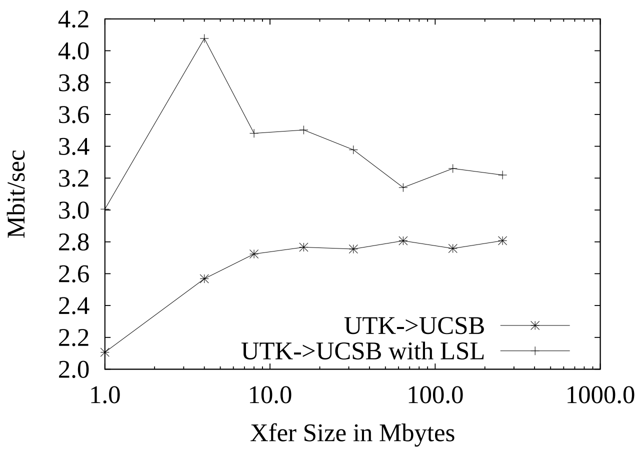

Fig. 10. Average Observed Bandwidth of data transfers from UTK to UCSB (1M - 128M)

Figure 10 shows the observed bandwidth of the wireless case. Note that the X axis is graphed using a log scale as the transfers in this test ranged from 1MByte to 256MByte. Again, we see the same pattern. In this case, sublink 1 (ironically, the non-wireless link) appears to be

percent average increase in bandwidth.

V. Analysis

Despite the additional protocol processing and additional latency introduced in the path, LSL's cascaded TCP connections frequently outperform direct TCP. This benefit results from a shorter RTT along each sublink (even though the combined "RTT" is longer) for LSL allowing TCP congestion control and avoidance mechanisms to function more rapidly. It has been long understood and repeatedly observed that the performance of TCP's congestion control state machine is governed by end-to-end RTT [24, 25] — a longer RTT implies a slower response and potentially less throughput at the beginning of a transfer, and immediately after the loss of a packet. Simply put, TCP can only resize its congestion window for a given stream (up or down) according to the speed with which it received acknowledgement packets. Our experiments with LSL demonstrates that by cascading TCP streams, the overall throughput is increased.

To investigate why this phenomenon might be occurring we use the commonly-accepted method for understanding the life of a TCP connection — the growth of the sequence number over time. From graphs such as these, we can see the duration of the transfer and understand some of its dynamics.

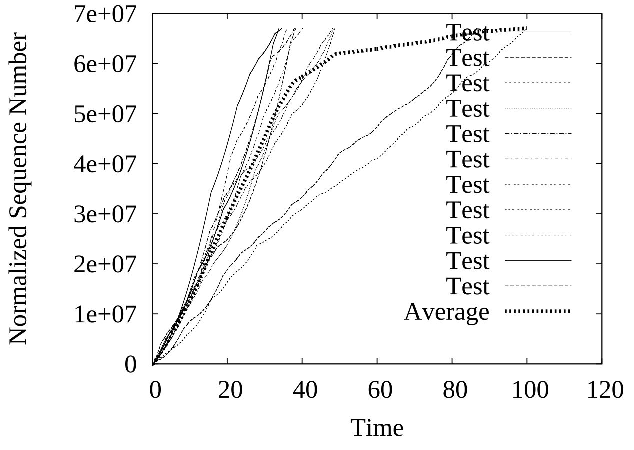

Fig. 11. Direct TCP connection sequence number growth through 64MB transfers from UCSB to UIUC – individual connections and their average.

For each of the cases presented above, packet traces were gathered at the sender. In addition to using these traces to compute the RTT figures above, we also normalized the sequence number so that the relative growth of the various iterations could be averaged. In Figure 11 we show the individual tests on a single graph (note the variation) and the average of the sequence number growth for

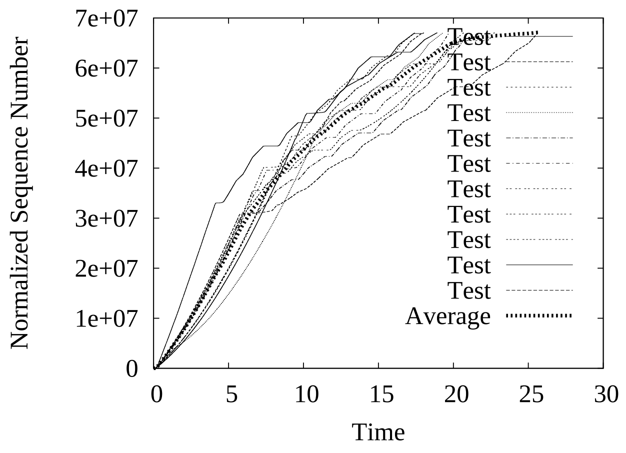

Fig. 12. Sublink 1 connection sequence number growth through 64MB transfers from UCSB to UIUC – individual connections and their average.

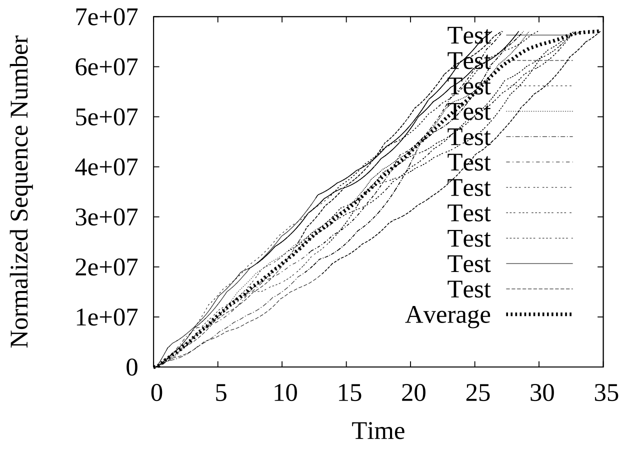

Fig. 13. Sublink 2 connection sequence number growth through 64MB transfers from UCSB to UIUC – individual connections and their average.

connections (no LSL). Figure 12 depicts subpath 1 of the LSL transfer between the same hosts and Figure 13 shows subpath 2. Note that subpath 2 has been normalized with respect to subpath 1 so that we can observe the relative growth of these cascaded connections.

Figure 14 compares the average sequence number growth over time for subpath 1, subpath 2 and the direct connection (taken from Figures 12, 13, and 11 respectively). It is easy to see the effects of RTT clocking on the window growth. Note that the seemingly flattened area toward the end of Figure 14 for direct TCP does not represent a single connection slowing toward the end of the transfer but is rather is an effect of averaging of all the connections in Figure 11.

To further isolate the effects of LSL on throughput we compare transfers of similar sizes having similar loss characteristics. That is, we wish to compare sequence number growth rate between LSL and direct TCP streams having similar loss rates.

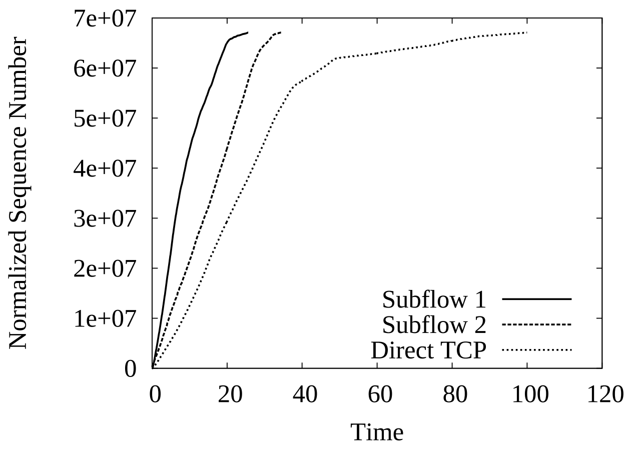

Fig. 14. Average sequence number growth through 64MB transfers from UCSB to UIUC – Sublinks and direct TCP.

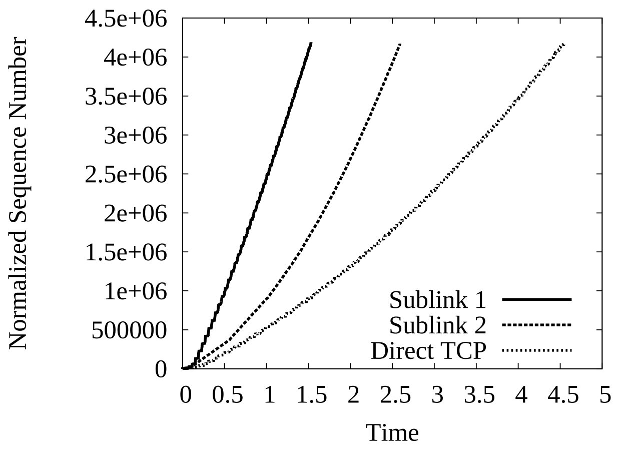

Fig. 15. Comparison of 4M transfers from UCSB to UIUC with no packet loss.

Figure 15 shows the sequence number growth for a 4MB transfer between UCSB and UIUC in which no packet loss occurred. Figure 16 shows cases in which the median observed packet loss (amongst all of the 4MB experiments) happened. Figure 17 shows a connection with the maximum loss rate and finally Figure 18 depicts the average of the sequence number growth over all tests. Our intention with this additional detail is to show the performance response (in terms of sequence number growth) of LSL and direct TCP under three conditions: minimal loss, median loss, and maximum loss. As these graphs clearly illustrate, for 4MB transfers even in the case when no packets are lost, the congestion control mechanisms can take significantly longer to open the window fully than in the LSL case. Indeed, even after 4MB, the slopes do not appear to have converged. The effect becomes more pronounced with increased loss rate as each LSL sublink can respond more quickly to the loss of a packet.

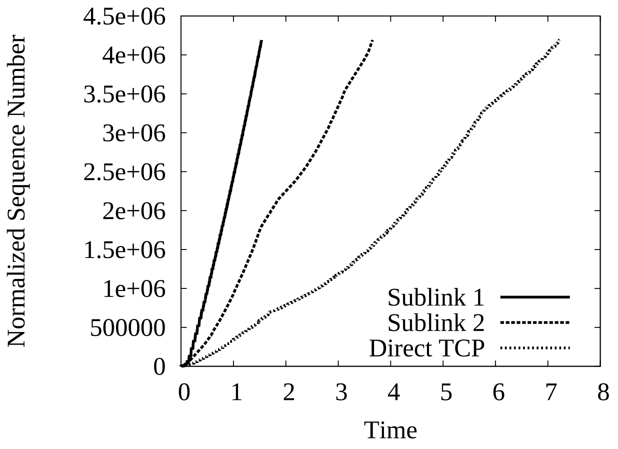

Fig. 16. Comparison of 4M transfer cases from UCSB to UIUC in which the median observed number of retransmissions occurred.

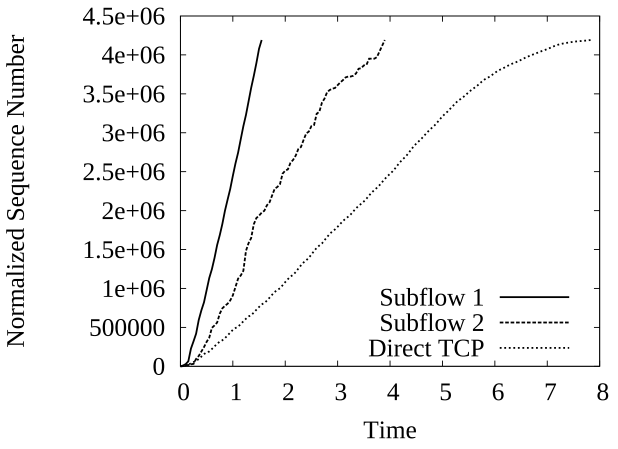

Fig. 18. Average sequence number growth for 4MB transfer cases from UCSB to UIUC.

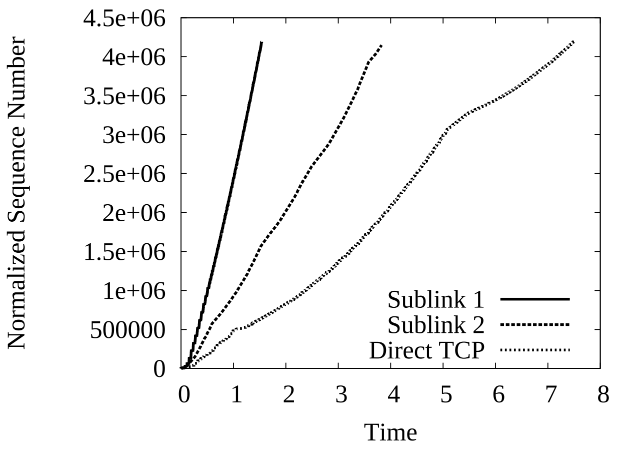

Fig. 17. Comparison of 4M transfer cases from UCSB to UIUC in which the maximum observed number of retransmissions occurred.

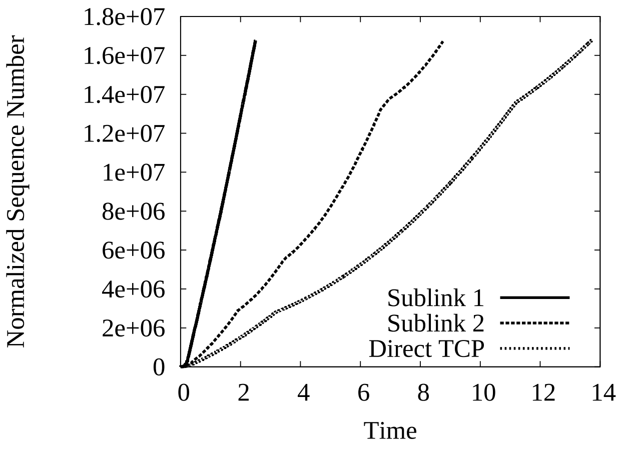

Fig. 19. Comparison of 16M transfer cases from UCSB to UIUC in which the minimum observed number of retransmissions occurred.

able before TCP has reached its "steady state" we conduct the same analysis as described above for successively larger transfer sizes. Figure 19 shows sequence number growth for a 16MB transfer from UCSB to UIUC in which the minimum observed packet loss occurred. No cases were observed with zero packet loss for transfers of this size. Again, Figure 20 and Figure 21 depict cases in which the median packet and maximum packet loss occurred respectively. Finally, Figure 22 shows the average sequence number growth over all cases as a measure of expected performance.

Similarly for 64MB transfers, Figure 23 shows the minimum loss case, Figure 24 show the median loss case and Figure 25 show the maximum loss case. The average sequence number growth is shown in Figure 14.

Figure 26 shows the average sequence number growth for 32MByte transfers from UCSB to UF (which uses the Houston LSL depot). Here, we note that the slopes are very close together, suggesting that subpath 1 (the link nearer the sender) was the bottleneck rather than subpath 2.

Finally, Figure 27 shows the sequence number for a 256MB transfer over the wireless link. This data is again consistent with the figures above and does indicate that sublink 1 was the bottleneck in this case.

VI. Discussion: TCP Performance and RTT

Some might argue that these transfers are all of moderate size and that the steady-state performance of TCP would ultimately be better than that using LSL. Certainly there are many configurations in which direct TCP would outperform LSL (e.g. if the LSL detour were a significant bottleneck or in cases where the end hosts are "close").

However in general, the control mechanisms used by TCP require that acknowledgments (ACKs) be sent from the receiver. This stream of ACKs acts as a clock for strobing packets into the network [39]. The speed with which slow-start allows the TCP connection to approach the advertised flow-control window is determined with the RTT (measured as the sum of the transit time of a

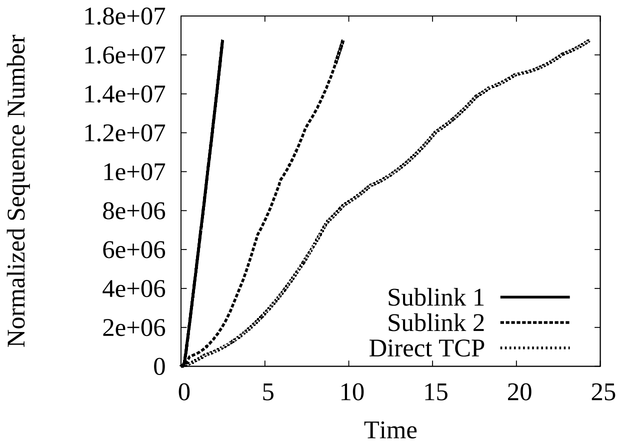

Fig. 20. Comparison of 16M transfer cases from UCSB to UIUC in which the median observed number of retransmissions occurred.

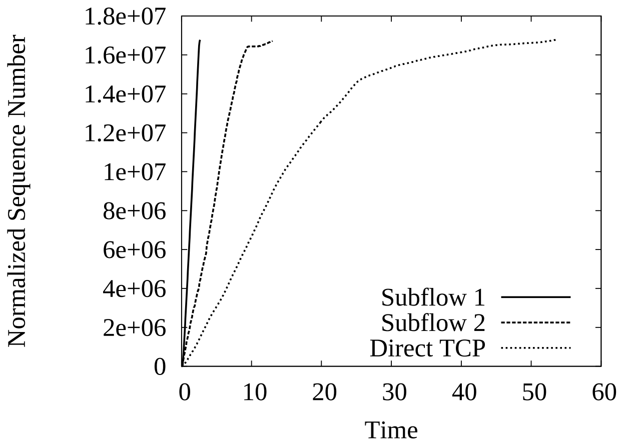

Fig. 22. Average sequence number growth for 16M transfer cases from UCSB to UIUC.

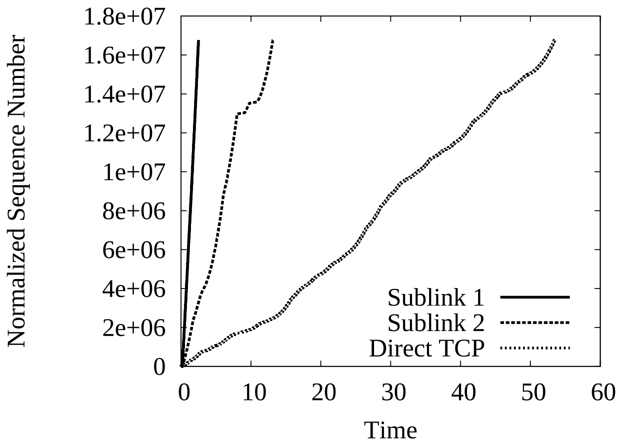

Fig. 21. Comparison of 16M transfer cases from UCSB to UIUC in which the maximum observed number of retransmissions occurred.

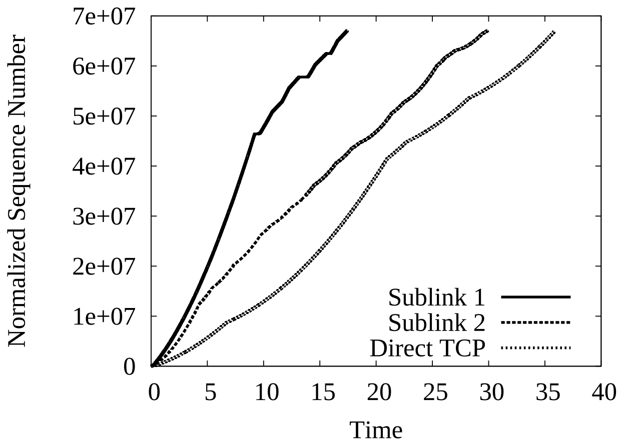

Fig. 23. Comparison of 64M transfer cases from UCSB to UIUC in which the minimum observed number of retransmissions occurred.

packet and its ACK.) The effects of RTT have been observed [24–26] but intuitively, since increase in congestion window requires a full RTT, the longer the RTT, the longer it takes TCP to reach full link capacity. Further, since loss events cause the congestion window to be reduced, the rate at which TCP increases the window is again governed by the RTT. The performance benefits of our system over direct TCP are simply due to effects that are endemic to TCP's congestion control mechanism [40] and we note that RTT of a path is important for the life of a connection.

To empirically examine the "steady-state" TCP performance, we performed experiments from UCSB to OSU with larger transfers sizes and more measurements of each case. Transfers from 32K to 512MB were made, with 120 tests of each transfer size. Figure 28 shows the average observed bandwidth (not sequence number growth) on transfers from 1MB to 512MB (and Figure 29 shows smaller transfers). Larger transfers very much seem to have captured the maximum available bandwidth and the trend shows no signs of convergence. This data seems to validate the dependence on RTT for the life of the connection even when steady-state is obviously reached.

VII. Future Work

To carry this work forward, we hope to continue to refine the design of our session layer. We believe that this abstraction is also useful for other approaches such as multi-path performance optimizations and parallel TCP streams. To facilitate this generalization and to allow for a more general-purpose implementation, we will investigate session-layer framing.

A. Scalability

This paper has not addressed the scalability of this system in any way, although we hope to do so in future work. We did not measure the effects of multiple-connection contention or examine carrying capacity of the experimental configuration. However, the hosts used in these

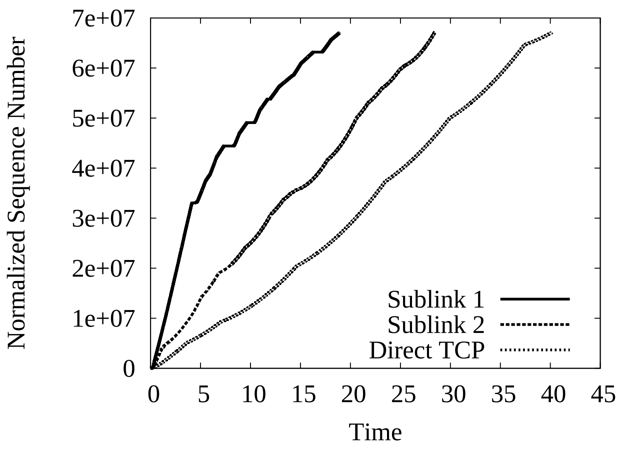

Fig. 24. Comparison of 64M transfer cases from UCSB to UIUC in which the median observed number of retransmissions occurred.

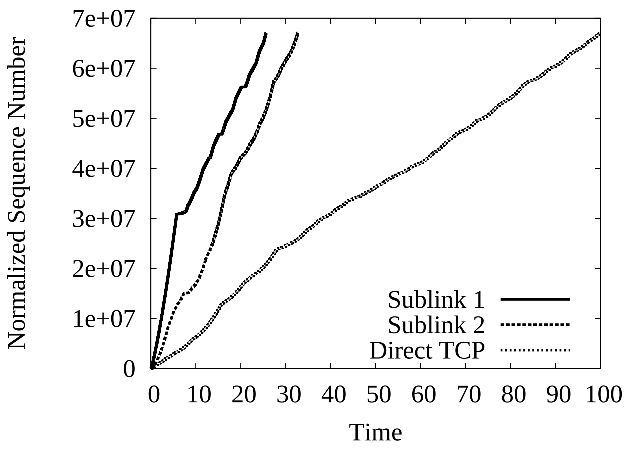

Fig. 25. Comparison of 64M transfer cases from UCSB to UIUC in which the maximum observed number of retransmissions occurred.

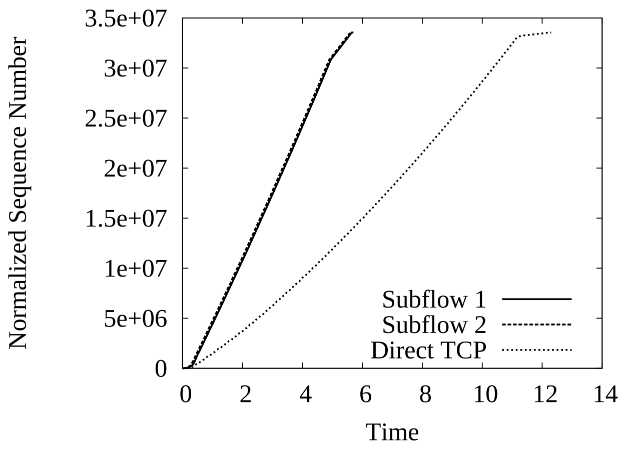

Fig. 26. Comparison of 32MB transfers from UCSB to UF.

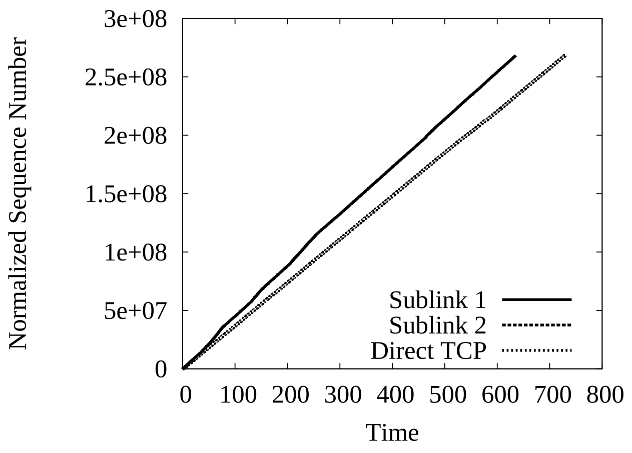

Fig. 27. Sequence number growth for 256MB wireless case.

experiments were general purpose, single-homed computers and were not designed to forward traffic efficiently like dedicated network hardware. Also, due to the very nature of the protocol, admission control and load balancing over a pool of available depots could easily be used to provide scalability.

VIII. Conclusion

This paper discusses an evolution to the Internet model that removes the requirement for end-to-end congestion control and instead allows for hop-by-hop congestion control in a novel manner. We present an architecture that allows us to speak about such systems as an integral part of the Internet rather than application-level workarounds. We show a performance advantage to segmented techniques such as this over TCP for reasons that are simply due to TCP's control mechanism. Finally, we empirically demonstrate this architecture in two very different contexts – Grid computing and wireless networks – both of which will see more and more use in the coming years.

References

[1] Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc., 1998.

[2] J. Postel, "Transmission control protocol," RFC 793, USC/Information Sciences Institute, September 1981.

[3] Information Technology: Open Systems Interconnection International Organization for Standardization, ," ISO/IEC 8827, 1990.

[4] 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.

[5] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski, "Managing data storage in the network," *IEEE Internet Computing*, vol. 5, no. 5, pp. 50–58, September/October 2001.

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

[7] D. Crocker, "Standard for the format of ARPA internet text messages," RFC 822, August 1992.

[8] P. Lapsley B. Kantor, "A proposed standard for the stream-based transmission of news," RFC 977, February 1986.

[9] J. Postel, "Internet protocol," RFC 791, USC/Information Sciences Institute, September 1981.

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

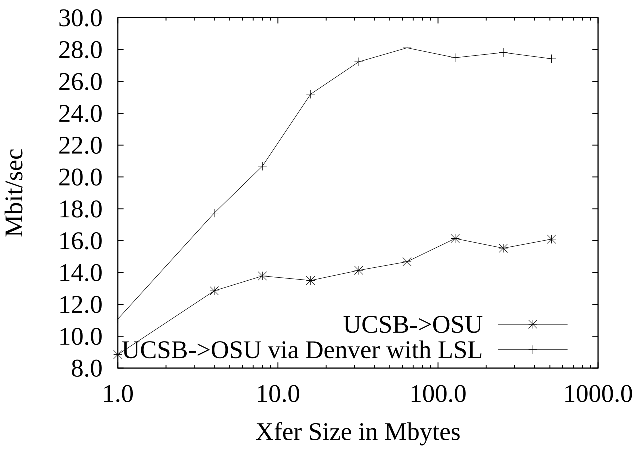

Fig. 28. Comparison of transfers from UCSB to OSU (1MB - 512MB in size – log scale).

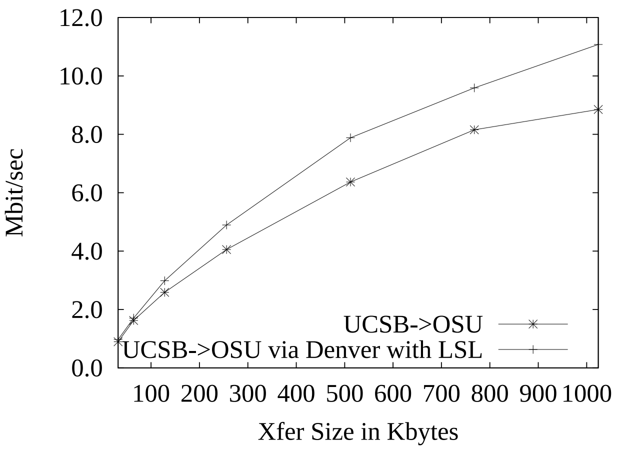

Fig. 29. Comparison of transfers from UCSB to OSU (32KB - 1024KB in size).

[11] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson, "The end-to-end effects of internet path selection," in *SIGCOMM*, 1999, pp. 289–299.

[12] 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.

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

[14] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr., "Overcast: Reliable multicasting with an overlay network," pp. 197–212.

[15] David A. Maltz and Pravin Bhagwat, "MSOCKS: An architecture for transport layer mobility," in *INFOCOM (3)*, 1998, pp. 1037–1045.

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

[17] Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, 1995.

[18] Ajay Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," 1995.

[19] P. Rodriguez, S. Sibal, and O. Spatscheck, "TPOT: Translucent proxying of TCP," Technical report TR 00.4.1, ATT Research Labs, 2000., 2000.

[20] Ulana Legedza and John Guttag, "Using network-level support to improve cache routing," *Computer Networks and ISDN Systems*, vol. 30, no. 22–23, pp. 2193–2201, 1998.

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

[22] 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.

[23] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," Network Working Group, Internet Engineering Task Force. Request For Comments: 1323, May 1992.

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

[25] 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.

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

[27] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe, "Modeling TCP throughput: A simple model and its empirical validation," *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 303–314, 1998.

[28] T. Ott, J. Kemperman, and M. Mathis, "The stationary behavior of ideal tcp congestion avoidance," `ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps`.

[29] Mark Allman and Vern Paxson, "On estimating end-to-end network path properties," in *SIGCOMM*, 1999, pp. 263–274.

[30] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field," 1998.

[31] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," RFC 1633, June 1994.

[32] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, pp. 119–132, January 1998.

[33] M. Mathis, R. Reddy, J. Heffner, and J. Saperia, "TCP extended statistics MIB," Internet Draft, draft-mathis-rfc2012-extension-00.txt, November 2001.

[34] R. Rivest, "The md5 message-digest algorithm," Network Working Group, Internet Engineering Task Force. Request for Comments: 1321, April 1992.

[35] Vern Paxson, "End-to-end internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.

[36] Jerome H. Saltzer, David P. Reed, and David D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, 1984.

[37] "SOCKS," http://www.socks.nec.com/.

[38] "Abilene," `http://www.ucaid.edu/abilene/`.

[39] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, vol. 18, 4, pp. 314–329, 1988.

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