Data Logistics in Network Computing: The Logistical Session Layer

D. Martin Swany and Rich Wolski
Computer Science Department
University of California, Santa Barbara
{*swany,rich*}*@cs.ucsb.edu*

Abstract

In this paper we present a strategy for optimizing end-to-end TCP/IP throughput over long-haul networks (i.e. those where the product of the bandwidth and the delay is high.) Our approach defines a Logistical Session Layer (LSL) that uses intermediate process-level "depots" along the network route from source to sink to implement an end-to-end communcation session. Despite the additional processing overhead resulting from TCP/IP protocol stack Unix kernel boundary traversals at each depot, our experiments show that dramatic end-to-end bandwidth improvements are possible. We also describe the prototype implementation of LSL that does not require Unix kernel modification or root access privilege that we used to generate the results, and discuss its utility in the context of extant TCP/IP tuning methodologies.

1 Introduction

The need for flexible and high-performance access to distributed resources has driven the development of networking since its inception. With the maturing of "The Internet" this community continues to increase its demands for network performance to support a raft of emerging applications including distributed collaboratoria, full-motion video, and Computational Grid programs.

Traditional models of high-performance computing are evolving hand-in-hand with advanced networking [13]. While distributed computation control and network resource control [14] techniques are currently being developed, we have been studying the use of time-limited, dynamically allocated network buffers [28] as a way of provisioning the communication medium. We term this form of networking *Logistical Networking* [7] to emphasize the higher-level control of buffer resources it entails.

In this paper, we present a novel approach to optimizing end-to-end TCP/IP performance using Logistical Networking. Our methodology inserts application-level TCP/IP "depots" along the route from source to destination and, despite having to doubly traverse a full TCP/IP protocol stack at each depot, improves bandwidth performance. In addition, we have implemented the the communication abstractions that are necessary to manage each communication without kernel modifications as a set of session-layer semantics over over the standard byte-stream semantics supported by TCP/IP sockets. As a result, we term the abstractions we have implemented the *Logistical Session Layer* (LSL).

LSL improves end-to-end network performance by breaking long-haul TCP/IP connections into shorter TCP segments between depots stationed along the route. Staging data at the session layer in a sequence of depots increases the overhead associated with end-to-end communication. In the LSL case, data emanating from the source must be processed twice (ingress and egress) at each depot thereby increasing the overall protocol processing overhead. In this paper, we show that this performance penalty is dramatically overshadowed by the performance improvement that comes from moving TCP end-points closer together. It is counter-intuitive that adding the processor overhead incurred by traversing the protocol stack on an additional machine could actually *improve* performance. Indeed, for some time the networking community has focused on TCP/IP overhead [9, 21] and examined ways to mitigate it [22, 32, 37]. To introduce additional protocol processing runs against the current optimization trends in high-performance wide-area networking and computing. However, *despite the additional processing overhead that comes from moving the data in and out of the kernel at each depot (including checksumming costs), moving TCP end-points closer together can improve end-to-end performance.*

We present this work in the context of recent networking trends that focus on state management in the network fabric itself. While the Internet Protocol suite (as typically implemented) mandates the communication state be managed at the end-points [34], new "stateful" facilities [8, 26]

which relax this restriction have been proposed. In this vein, we believe that there are several reasons that intermediate TCP processing helps, rather than hurts, end-to-end bandwidth performance. First, since the round-trip time (RTT) between any two depots is shorter than the end-to-end round-trip-time, LSL allows the inherent TCP congestion-control mechanism to sense the maximally available throughput more quickly. That is, even though the sum of the RTTs between depots may be longer than the end-to-end RTT, because the maximum RTT between any two depots is shorter, the congestion-control mechanisms adapt more rapidly. Secondly, a retransmission that results from a lost packet need not originate at the source, but rather, can be generated from the last depot to forward the data. Finally, recent advances in the processing speed, memory bandwidth, and I/O performance of commonly available processors has lowered protocol processing and data movement costs relative to available network performance. We describe, more completely, the confluence of these effects in Section 3.

In Section 2, we describe the architecture of a prototype application-layer LSL implementation that we have developed. The advantage of providing a session-layer interface is that applications do not need to employ their own customized buffer management strategies in order to use Logistical Networking to enhance end-to-end network performance. As such, our work not only provides a general methodology for improving deliverable network performance, but it also constitutes an important early example of a Grid-enabling network abstraction. At the same time, since our implementation does not require kernel modification, it is portable and easy to deploy.

Finally, in Section 4 we detail the effect of using intermediate TCP depots and LSL on end-to-end bandwidth, independent of end-point buffer settings, both with and without the RFC 1323 [20] window-scaling. Our results show that, using LSL, an application can gain a substantial end-to-end increase in bandwidth over standard TCP/IP sockets, even if the socket connections have been "tuned" for performance.

2 Architecture

The Logistical Session Layer (LSL) is a "session" layer (layer 5) in terms of the OSI protocol model. The session layer lies above the Transport layer (TCP, in the Internet Protocol suite). Recall that a transport layer conversation consists of multiple hops of network layer conversations. In an analogous fashion, a session layer conversation can consist of multiple hops of transport layer conversations. [18]. A connection that is initiated through the LSL will pass through a number of LSL-aware routers, or "depots." These devices can actually be thought of as "transport layer switches" in that they multiplex session-layer conversations onto sets of transport layer conversations. While we believe that a kernel-level implementation or dedicated system versions of these switches will, ultimately, improve performance over the results we report in the next section, we have chosen a non-privileged, application-level implementation initially. By doing so, we are able to gain two important benefits. First, because the LSL depots use standard, user-level sockets, our initial implementation of LSL does not (and, indeed, cannot) violate the current TCP congestion-control mechanisms. From the perspective of the network, an LSL session appears to be a series of user-level applications communicating in a chain. All resource control mechanisms governing "normal" user-applications (such as flow-control, congestion-control, memory-size, etc.) remain functional and need not be disabled.

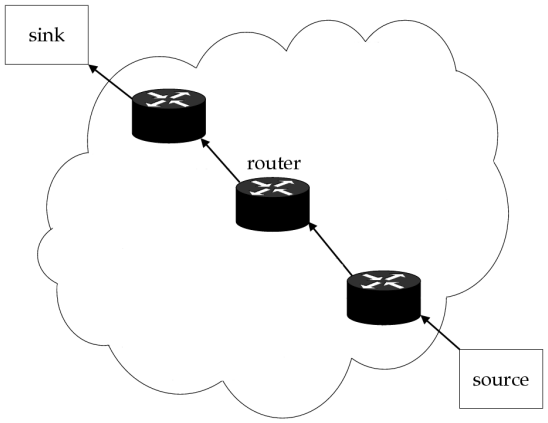

Figure 1. End-to-End TCP Communication

Secondly, because LSL depots can run under any user login id (i.e. do not require root access), security and stability concerns are greatly reduced. It is not possible for an LSL depot to be used as a vehicle for obtaining root access because it does not run as root and it does not execute any functions not compiled into its binary image.

Additionally, our first implementation of the LSL client API closely mimics the familiar Unix sockets interface. This design choice allows easy incorporation into legacy applications. Users of the socket interface are familiar with the "Internet" address family, denoted with **AF_INET**. We designate a new family, which we label **AF_LSL**. So, for a given program to use LSL, a simple text substitution in the source code would enable use of the system. The connection would "fall back" to using a direct TCP connection if necessary to make the change less intrusive.

Figure 1 illustrates a TCP stream traversing a series of routers. Figure 2 shows communication passing through an

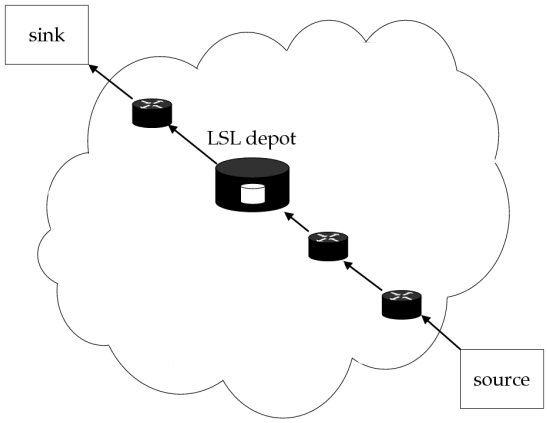

Figure 2. Network communication with LSL

LSL depot along the path from source to sink. Of course, a session may pass through zero or more LSL depots.

An application should be able to direct the LSL session to use a given depot if necessary to allow application-level tuning. In addition, we plan an end-to-end routing service based on the Network Weather Service [42, 43] that determines a "good" route for each client. In either case, utilizing the Sockets interface for this simply entails specifying a source routed path. When an LSL connection is initiated, a predicted path may be specified or local forwarding decisions may be relied upon. To specify a path explicitly, the sender will use the strict source route options with the LSL socket. In fact a combination of local and global forwarding strategies may be employed by specifying a loose source route in the same fashion.

To test out the effectiveness of LSL and begin to understand its potential performance benefits, we have implemented and deployed a rudimentary prototype having three components:

- a simple file server program called *lsrv*,
- a per-depot session-level daemon that establishes and releases TCP streams traversing each depot called *lsd*, and
- a client (responsible for choosing end-to-end routes) called *lget*.

Our intention is to use this framework to study both the performance characteristics of LSL and how LSL may be implemented for computational Grid settings. By thus modularizing the LSL system, we will be able to take advantage of the functionality provided by systems such a Globus [12], Legion [15], and the Network Weather Service [43].

2.1 The End-to-End Argument and the Session Layer

The architecture of the current Internet Protocol suite has been guided by what is known as the "end-to-end" argument [34]. This model (as commonly understood) dictates that state be kept at the end nodes and that the core of the network be stateless primarily for reasons of reliability. Recent trends in network service provision [8, 26], however, relax the requirement of statelessness in favor of better performance and service quality control. Indeed, the general question of end-to-end versus stateful networking is also being considered explicitly by many, including the original authors of the end-to-end argument [8, 31]. 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.

In short, the general application of the principle is somewhat different than the networking community at large has come to understand. However, since we use the semantics of a session layer to implement our system, even the most dogmatic network engineer will be left without argument.

3 Observations

The key idea behind LSL is that, by allowing an application to temporarily and anonymously utilize buffers "in" the network, end-to-end performance will be enhanced. It is intuitive that there is a fundamental cost associated with buffering unacknowledged segments for retransmission. Moreover, it is clear that the problem is only exacerbated as network speeds increase.

By its very definition LSL causes the end-to-end connection to have a larger *aggregate window*. We define this to be the sum of the minimum of the congestion window (*cwnd*) and the advertised window (*rwnd*) over each of the TCP connections. In exposing the pipeline we have, in fact, increased the "capacity" of the network.

In addition, LSL optimizes the end-to-end bandwidth performance in two ways: by improving the response of the congestion-control mechanisms that are currently in use, and by exploiting locality for packet retransmission.

3.1 Congestion Control and Cascaded TCP

By cascading TCP streams, LSL affects TCP congestion control in two ways. First, it shortens the RTT that any constituent TCP stream uses to "clock" the rate at which

congestion-window modifications are made. Secondly, it isolates RTT variance so that retransmission times across low variance links are based on more accurate estimates of variance.

By staging data above the transport layer at depots along the path from source to sink, LSL reduces the bandwidth-delay product associated with any single TCP connection. As such, it allows the adaptive congestion-control mechanisms [2]) to achieve maximal throughput and recover from loss more quickly. The flow-control and congestion-control mechanisms used by TCP require that an acknowledgement (ACK) be sent from the receiver. This stream of ACKs acts as a clock for strobing packets into the network [19]. 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 packet and its ACK.) The effects of RTT have been observed [23, 25] 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. By using multiple LSL depots with stream between each, TCP can discern the congestion-and flow-control-levels for each communication and achieve maximal link capacity more quickly.

The second effect on TCP comes from a reduced variance in RTT. By shortening the length of the links that TCP traverses, LSL potentially isolates variance in packet delay. The TCP protocol uses an estimate of the variance in RTT to determine when a retransmission is necessary in response to a lost packet. if the TCP stream traverses a large number of hops, high variance in queue delay at any point along the route affects the retransmission time out for the entire route. By breaking the end-to-end route up into segments between depots, LSL allows TCP to better estimate RTT variance on a link-by-link basis. The result is that retransmissions are more accurately triggered, and performance is improved.

3.2 Locality and Packet Retransmission

By buffering data at intermediate points along the route from source to sink, LSL reduces the overhead associated with retransmission. With the current TCP buffering model, a lost packet at any point between source and sink requires a retransmission from the communication source point. The retransmitted packet, then, must traverse the entire network using valuable bandwidth along the entire route. For example, consider an end-to-end communication between two hosts separated by 10 routers or gateways in which packets are being dropped at the last gateway before the sink. Every time the 10th router drops a packet, the resulting retransmission must traverse the other 9 routers, taking up scarce buffer and bandwidth resources. By buffering data at the session layer, LSL ensures that any retransmits traverse only the links between depots. The result is a savings in the bandwidth that would otherwise be wasted from end-to-end moving retransmitted data.

4 Results

The place that we expect the LSL optimization effects to be most apparent is in long-running transfers over networks where the bandwidth-delay product is high. In this section, we examine several example transfer paths that terminate at the University of Tennessee, Knoxville (UTK). Since we had complete control over the UTK machines, we were able to investigate the effects of different kernel-level TCP settings.

In the first test we study transfers from Argonne National Laboratory (ANL) to the University of Tennessee (UTK). To do so, we deploy an LSL daemon at Oak Ridge National Laboratory (ORNL) to serve as a depot between UTK and ANL. UTK is directly connected to ORNL via an OC-3 (155 Mb/sec) link, and ORNL and ANL are both connected to the Energy Sciences Network (ESnet) at OC-12 (622 Mb/sec) [11]. Appendix A.1 is the output of the *traceroute* command from UTK to both other sites, Appendix A.2 and A.3 are from ANL and ORNL, respectively.

The ANL, ORNL, and UTK machines were configured to use the RFC1323 [20] window-scaling optimizations and large kernel buffers. For this experiment, we set the kernel buffers (through the Unix *setsockopt()* command) to be eight megabytes at both ends, and verified that the correct window size was bring quoted using *getsockopt* and *tcpdump* at the UTK end.

The results in Figure 3 represent roughly 280 experiments in total. Along the x-axis we show a series of different transfer sizes. The y-axis of the figure indicates the observed, end-to-end throughout in megabits per second. Each data point represents the average throughput observed over 20 different transfers at transfer size corresponding to its x coordinate.[1] In addition, the x-axis is shown on a log scale.

Figure 3 shows that the LSL does indeed optimize end-to-end transfers of 256KBytes and larger. For transfers of 32Mbytes, use of the LSL depot at ORNL increases the average performance by well over a factor of 2.5.

While investigating these results, we observed that the route between UTK and ANL is asymmetric as the traceroute from ANL to UTK (in Appendix A.2 indicates. As this is the case, part of this improvement can be attributed to the fact that by explicitly routing through ORNL, we are enforcing symmetric paths. This "user-controlled" routing is somewhat analogous to the IP "source route" option and we discuss similarities to this and other approaches in Section 5.

[1] Appendix B shows some summary statistics and average transfer rates for each transfer size we consider in this paper. For visual clarity, the figures we present depict the averages only.

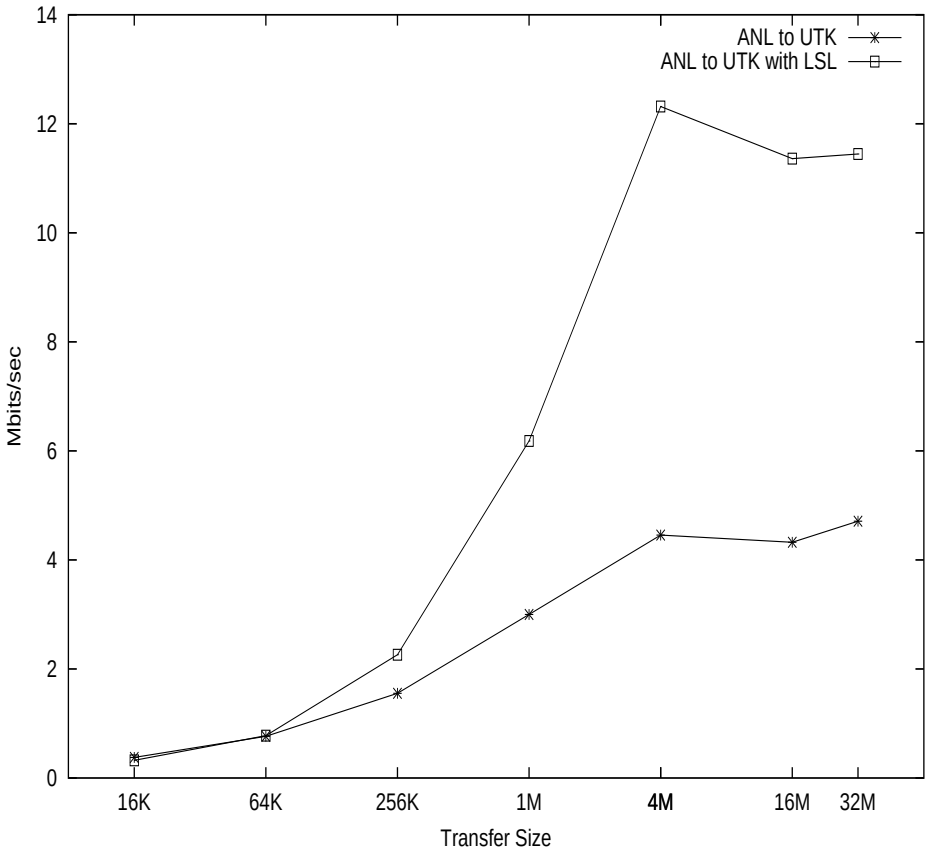

Figure 3. Data transfer from ANL to UTK

Since this might be viewed as a pathological case (although recent work [27] indicates otherwise) we sought a case in which the LSL route corresponded with the non-LSL route in both directions. We chose a path between the University of Tennessee (UTK) and the University of California at San Diego (UCSD) using a machine located at Texas A&M University, South Campus which is very near the Abilene [1] "Point of Presence" (POP) in Houston. Traffic from UTK to UCSD and vice versa traverses this POP. Appendix A.4 shows the traceroute from UTK to UCSD and TAMUS. Appendix A.5 and A.6 show traceroutes from UCSD and TAMUS, respectively.

Figure 4 illustrates the comparison of LSL-enabled and non-LSL-enabled end-to-end bandwidth performance from UCSD to UTK, again using 8 MB buffers and windows. Here, LSL offers as much as a %50 improvement over direct TCP *despite* adding to the gross latency and protocol processing overhead along the path from source to sink.

For high-capacity long-haul networks (like Abilene), large window sizes are necessary so that the sending side does not block due to flow-control before an acknowledgement has time to return from the receiver. That is, the buffering must allow the amount of data in flight to exceed the bandwidth-delay product. By choosing 8 MB windows and buffers, we ensure that the LSL effects we observe not not simply due "bad" buffering choices at the end points. We believe that an optimized TCP stream using large buffers at either end would see similar performance improvements.

Although we recognize that buffers of this size may not be optimally tuned, this does not effect our results. The danger in over-sizing buffers is in wasting resources on the host, not in causing poor TCP performance [37]. We judged this to be acceptable for this experiment, but we do await the products produced by groups like the Web100 [41] and the Internet2 End-to-End Performance initiative [17].

However, not all hosts support (through design or configuration) large window sizes. For this case, we wanted to investigate how using an LSL depot with large windows might enhance the performance of "untuned" TCP streams. Figure 5 shows the average transfer rates when the buffers at the sending and receiving ends are restricted to 64K bytes.

As expected, the absolute performance is lower. However, the LSL-enabled stream was still able to outperform the non-enabled stream by %43 for the largest transfer size.

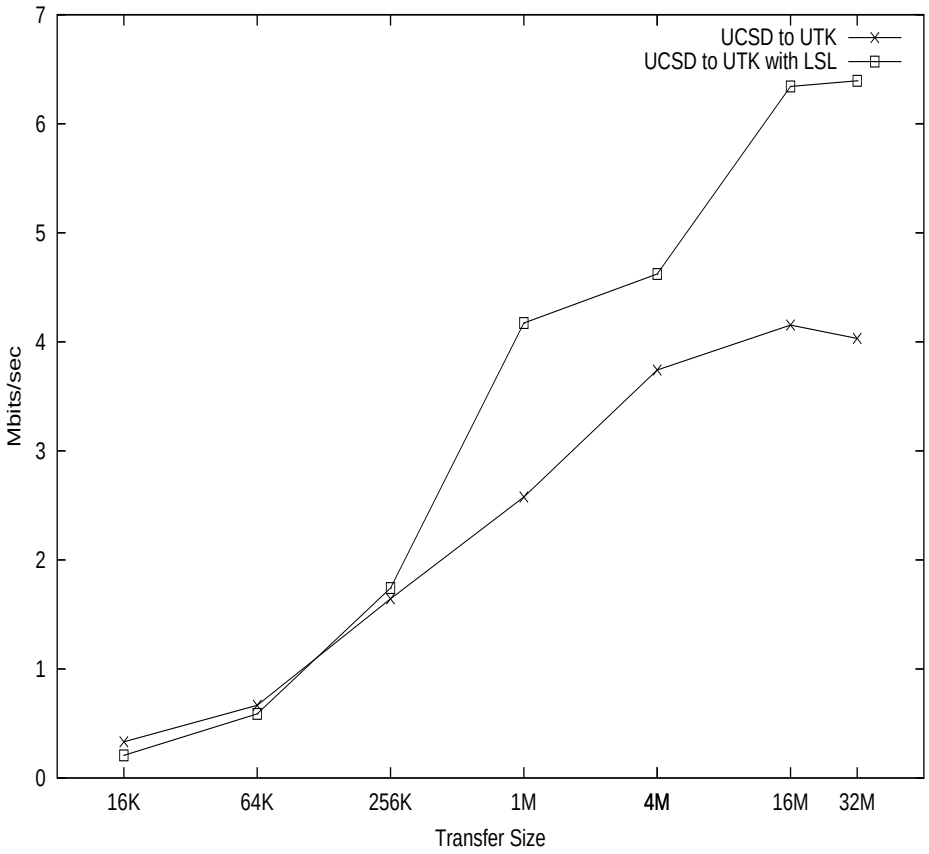

Figure 4. Data transfer from UCSD to UTK

As a non-intrusive optimization, however, we believe that such a performance improvement will be useful in many application settings.

Finally, Figure 6 shows the effect of using LSL on a TCP stream that does no *setsockopt()* buffer conditioning whatsoever.

Again, the performance is not as dramatic as in the tuned cases, but the LSL effect is still present.

5 Related Work

There are many areas in which facets of our results and similar mindsets can be seen in the community. Two broad categories are TCP effects (retransmission cost and locality) and control over the topology.

Techniques developed for wireless networks [5, 6] seek to mitigate the cost of retransmission in lossy environments. However, they violate layering to do so. Systems to proxy TCP have been developed with the same goals in mind. One example is TPOT [33], which alters TCP to allow this mode of operation (and therefore has dubious possibility for actual deployment in the global Internet.) A similar approach targeting caching of web objects also proposes modifications to TCP [24].

There are many approaches to reducing the cost of retransmission with the network's assistance. One of the areas that has pushed this notion forward is the wireless community. Since wireless links at the edges of the network tend to be much less reliable than other parts of the network, the almost inevitable packet loss hampers the performance of TCP. The "snoop" technique watches TCP traffic and can perform retransmits from and intermediate point in the network. [6]. This protocol comes from a very similar mindset, but could be considered inappropriate in that it violates layering. Further, flow-level snooping is expensive, so the scalability of this approach for high-bandwidth networks is questionable. Also targeting wireless is Indirect TCP [5], which is similar to our approach. Another similar approach is that proposed for use in multicast video transmission [38] in that it allows a data stream to take advantage of retransmit locality and finer-grained adaptability.

The PSockets [39] work addresses the difficulty of getting high-bandwidth from long, fat networks. This work is similar to LSL in that they preserve the syntax of the well-

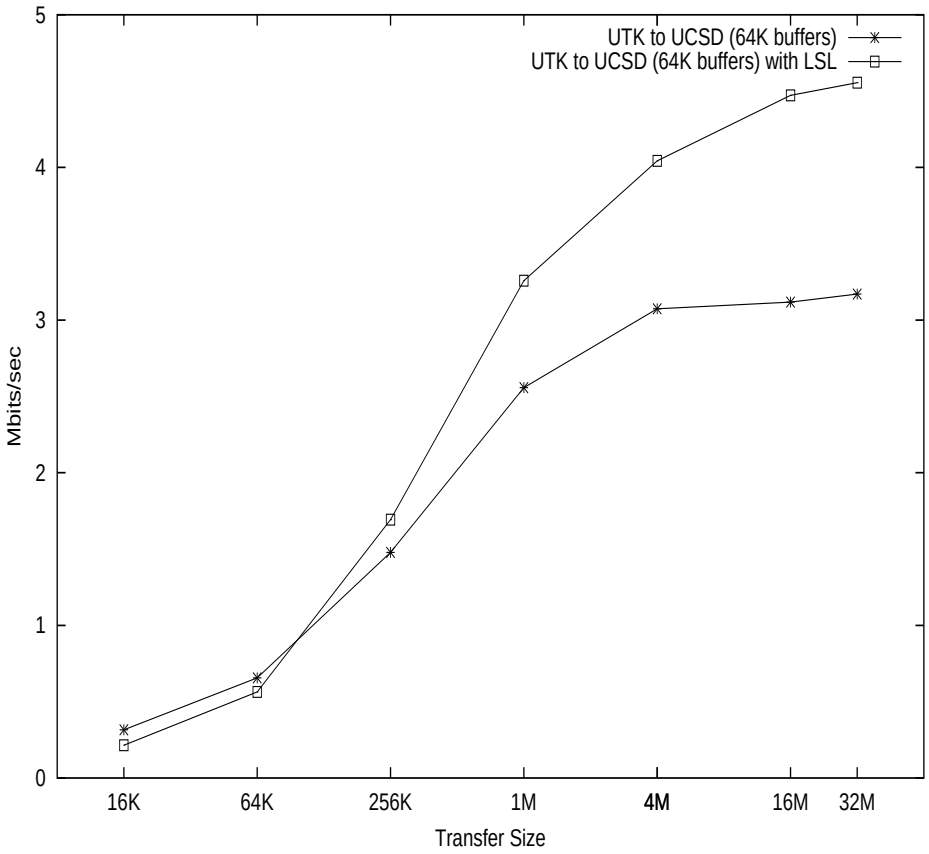

Figure 5. Data transfer from UCSD to UTK with 64KB buffers

known socket interface while taking liberties underneath. They differ in mechanism and spirit, however. This approach to higher bandwidth uses multiple TCP streams and keeps aggregate throughput high by amortizing loss over several streams. The parallel sockets approach is to ignore TCP's inefficiencies in the wide area and aim for better average throughput. However, the streams are then inducing the congestion that the other streams are sensing. So, Psockets provides better average performance but worse link utilization. LSL allows the network to be "articulated" and allows TCP to respond to congestion where it exists without introducing additional load.

The need to control the topology by tunneling from one host to another has been identified by a body of engineering and by the sheer number of Virtual Private Networks that are in use. Even outside the desire to provide an encrypted tunnel, virtual topology systems are abundant [35, 40]. The performance benefits of user-optimization of paths has been discussed [3, 30, 36]. While LSL has the functionality of these systems, our approach is different in that we consider this to be part of a "session" layer of services. Our empirical evidence does establish the viability of this line of research.

Finally, within the networking community, the notion has long existed that the end user will occasionally need to have explicit control over the route that traffic follows. Loose and strict source routing were defined in RFC 791 [29], which defines the Internet Protocol. Store and forward connectivity has been used for quite some time in the networking community and there are many situations in which data transfer need not be synchronous or connection-oriented. SMTP [10], USENET [16] and its successor NNTP [4], all use hop-oriented, connectionless paradigms to send data.

6 Conclusion

With the maturing of network infrastructure, both in terms of ubiquity and quality, comes the possibility of increasing the state held in the network, and the time duration over which it is maintained. Logistical Networking [7] attempts to define the parameters under which the added cost associated with maintaining state is overshadowed by an increase in delivered network performance. The Logistical Session Layer (LSL) is an implementation of Logis-

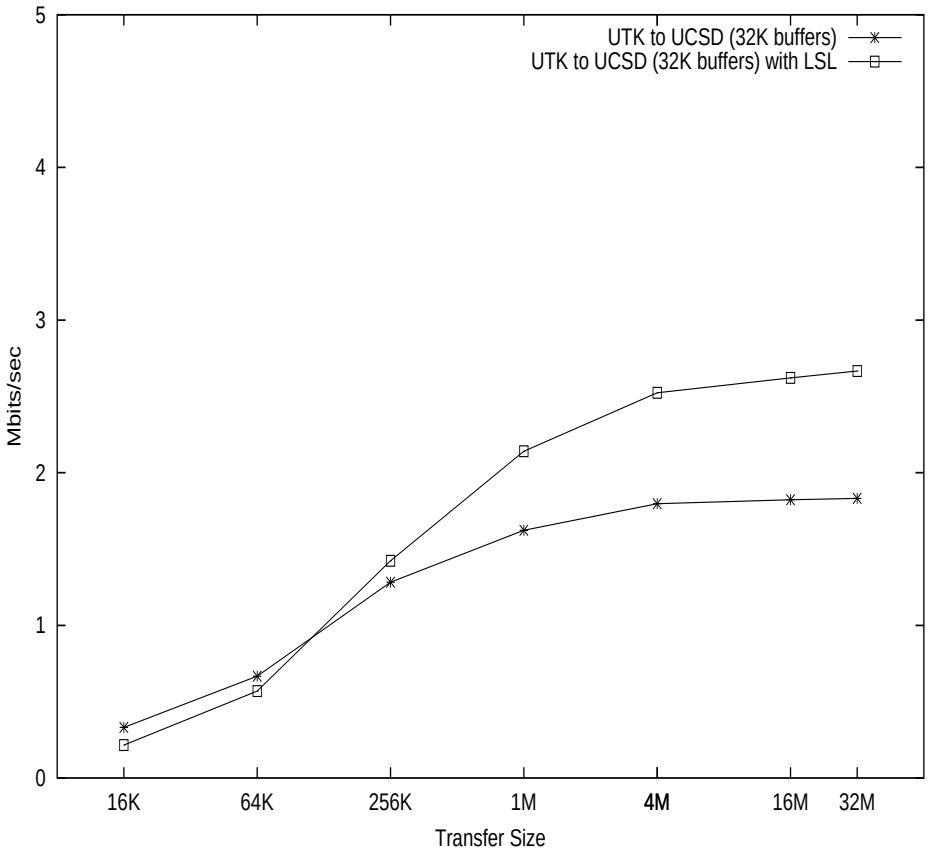

Figure 6. Data transfer from UCSD to UTK with default (32KB) buffers

tical Networking concepts to improve end-to-end communication performance between applications that currently use the TCP socket interface. Our early results show that LSL can result in dramatic throughput performance improvements despite greater protocol processing overhead. In addition, our initial prototype is serving as an architectural framework within which he hope to generalize these results.

Important research questions exist such as the permanence of the buffering, the nature of the flow control mechanisms and alternate architectures for a mechanism such as this. We believe that we have demonstrated the efficacy of this system and that it is novel, viable approach.

References

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

[2] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. *RFC 2581*, page 13, 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] P. L. B. Kantor. A proposed standard for the stream-based transmission of news. RFC 977, February 1986.

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

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

[7] 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.

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

[9] D. D. Clark, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. In *Proc. 13th Conference on Local Computer Networks*, pages 284–291, Minneapolis, Minn., 10-12 1988. IEEE Computer Society.

[10] D. Crocker. Standard for the format of arpa internet text messages. RFC 822, August 1992.

[11] Esnet. http://www.es.net.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

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

[14] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. N. dt, and A. Roy. A distributed resource management architecture that supports advance d reservation and co-allocation. In *International Workshop on Quality of Service*, 1999.

[15] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. Legion: The next logical step towrd a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, 1994.

[16] M. Horton. Standard for interchange of usenet messages. RFC 850, June 1983.

[17] Internet2 end-to-end performance initiative. http://www.internet2.edu/e2eperf/.

[18] I. T. O. S. I. International Organization for Standardization. ISO/IEC 8827, 1990.

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

[20] 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.

[21] J. Kay and J. Pasquale. The importance of non-data touching processing overheads in TCP/IP. In *SIGCOMM*, pages 259–268, 1993.

[22] J. Kay and J. Pasquale. Profiling and reducing processing overheads in TCP/IP. *IEEE/ACM Transactions on Networking*, 4(6):817–828, December 1996.

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

[24] U. Legedza and J. Guttag. Using network-level support to improve cache routing. *Computer Networks and ISDN Systems*, 30(22–23):2193–2201, 1998.

[25] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm, 1997.

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

[27] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[28] J. Plank, A. Bassi, M. Beck, T. Moore, M. Swany, and R. Wolski. The internet backplane protocol: Storage in the network. IEEE Internet Computing (to appear), 2001.

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

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

[31] D. Reed, J. Saltzer, and D. Clark. Commentaries on the active networking and end-to-end arguments, 1998.

[32] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-performance local-area communication with fast sockets. In *USENIX 1997 Annual Technical Conference*, pages 257–274. USENIX, 1997.

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

[34] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[35] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for informed internet routing and transport. Technical Report TR-98-10-05, University of Washington, 1998.

[36] 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.

[37] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *SIGCOMM*, pages 315–323, 1998.

[38] S. Sen, D. Towsley, Z.-L. Zhang, and J. Dey. Optimal multicast smoothing of streaming video over an internetwork. Technical Report UM-CS-1998-077, 1998.

[39] 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.

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

[41] Web100. http://www.web100.org.

[42] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, January 1998. also available from http://www.cs.utk.edu/ rich/publications/nws-tr.ps.gz.

[43] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999. available from `http://www.cs.utk.edu/~rich/publications/nws-arch.ps`

A Path information

A.1 Output of `traceroute` from UTK to ANL and UTK to ORNL

traceroute to pitcairn.mcs.anl.gov

1 r6hm01v150.ns.utk.edu – 0.181 ms
2 128.169.192.241 – 0.711 ms
3 r7dh03g11-0-0.ns.utk.edu – 2.021 ms
4 utk-gatech.ns.utk.edu – 65.490 ms
5 esnet-sox-rtr.sox.net – 48.575 ms
6 orn-gsu.es.net – 48.312 ms
7 nyc-s-orn.es.net – 81.595 ms
8 chi-s-nyc.es.net – 101.062 ms
9 anl-chi-ds3.es.net – 102.939 ms
10 anl-esanl2.es.net – 105.344 ms
11 stardust-msfc-20.mcs.anl.gov – 106.758 ms

12 pitcairn.mcs.anl.gov – 105.946 ms

traceroute to falcon0j.ccs.ornl.gov

1 r6hm01v150.ns.utk.edu – 0.171 ms
2 128.169.192.241 – 0.664 ms
3 r7dh03g11-0-0.ns.utk.edu – 1.657 ms
4 mmesgwya32.ctd.ornl.gov – 3.689 ms
5 192.31.96.225 – 2.061 ms
6 ornlgwy-ext.ens.ornl.gov – 2.736 ms
7 ccsrtr.ccs.ornl.gov – 2.094 ms
8 falcon0j.ccs.ornl.gov – 2.289 ms

A.2 Output of `traceroute` from ANL to UTK and ORNL

traceroute to modulus.cs.utk.edu

1 stardust-msfc-11.mcs.anl.gov – 4.73 ms
2 kiwi.anchor.anl.gov – 0.639 ms
3 abilene-anl.anchor.anl.gov – 5.29 ms
4 atla-ipls.abilene.ucaid.edu – 15.2 ms
5 sox-rtr.abilene.sox.net – 40.2 ms
6 r7dh03a1-0-2.ns.utk.edu – 107 ms
7 * * *
8 128.169.192.242 – 106 ms
9 modulus.cs.utk.edu – 106 ms

traceroute to falcon0j.ccs.ornl.gov

1 stardust-msfc-11.mcs.anl.gov – 0.515 ms
2 kiwi.anchor.anl.gov – 0.279 ms
3 esanl2-anl.es.net – 0.504 ms
4 chi-anl-ds3.es.net – 2.35 ms
5 nyc-s-chi.es.net – 22.7 ms
6 orn-s-nyc.es.net – 54.9 ms
7 ornl-orn.es.net – 75.0 ms
8 192.31.96.225 – 75.7 ms
9 ornlgwy-ext.ens.ornl.gov – 74.9 ms
10 ccsrtr.ccs.ornl.gov – 75.3 ms
11 falcon0j.ccs.ornl.gov – 74.7 ms

A.3 Output of `traceroute` from ORNL to UTK and ANL

traceroute to modulus.cs.utk.edu

1 ccsrtr-003.ccs.ornl.gov – 0.418 ms
2 160.91.0.65 – 0.357 ms
3 orgwy2.ens.ornl.gov – 0.327 ms
4 mmesgwy-ext-fe.cind.ornl.gov – 1.95 ms
5 utk-rtr.ctd.ornl.gov – 3.8 ms
6 * * *
7 128.169.192.242 – 2.19 ms
8 modulus.cs.utk.edu – 2.47 ms

traceroute to pitcairn.mcs.anl.gov

1 ccsrtr-003.ccs.ornl.gov – 0.342 ms
2 160.91.0.65 – 0.796 ms
3 orgwy2.ens.ornl.gov – 0.410 ms
4 ornl-rt3-ge.cind.ornl.gov – 0.503 ms
5 orn-ornl.es.net – 19.6 ms
6 nyc-s-orn.es.net – 52.8 ms
7 chi-s-nyc.es.net – 72.4 ms
8 anl-chi-ds3.es.net – 73.9 ms
9 anl-esanl2.es.net – 74.1 ms
10 stardust-msfc-20.mcs.anl.gov – 76.4 ms
11 pitcairn.mcs.anl.gov – 75.8 ms

A.4 Output of `traceroute` from UTK to UCSD and TAMUS

traceroute to freak.ucsd.edu

1 r5hm01v277.ns.utk.edu – 3.477 ms
2 r7dh03g11-0-0.ns.utk.edu – 2.253 ms
3 utk-gatech.ns.utk.edu – 66.578 ms
4 199.77.193.10 – 67.116 ms
5 hstn-atla.abilene.ucaid.edu – 85.412 ms
6 losa-hstn.abilene.ucaid.edu – 117.387 ms
7 usc–abilene.atm.calren2.net – 117.693 ms
8 UCSD–usc.pos.calren2.net – 120.841 ms
9 sdsc2–ucsd.atm.calren2.net – 121.619 ms
10 cse-rs.ucsd.edu – 122.653 ms
11 freak.ucsd.edu – 122.110 ms

traceroute to i2-dsi.ibt.tamus.edu

1 r6hm01v150.ns.utk.edu – 0.172 ms
2 128.169.192.241 – 0.939 ms
3 192.168.101.3 – 1.197 ms
4 utk-gatech.ns.utk.edu – 65.296 ms
5 atla.abilene.sox.net – 65.430 ms
6 hstn-atla.abilene.ucaid.edu – 84.695 ms
7 link2abilene.gigapop.gen.tx.us – 87.325 ms
8 link2ibt.gigapop.gen.tx.us – 86.420 ms
9 ibtx2-atm10-401.ibt.tamus.edu – 87.494 ms
10 i2-dsi.ibt.tamus.edu – 87.811 ms

A.5 Output of `traceroute` from UCSD to UTK and TAMUS

traceroute to modulus.cs.utk.edu

1 cse-danger-gateway.ucsd.edu – 0.622 ms

2 bigmama.ucsd.edu – 1.224 ms
3 ucsd–sdsc2.atm.calren2.net – 1.542 ms
4 usc–ucsd.pos.calren2.net – 5.961 ms
5 abilene–usc.atm.calren2.net – 4.955 ms
6 hstn-losa.abilene.ucaid.edu – 37.429 ms
7 atla-hstn.abilene.ucaid.edu – 56.922 ms
8 199.77.193.9 – 57.155 ms
9 r7dh03a1-0-2.ns.utk.edu – 124.299 ms
10 192.168.101.40 – 124.302 ms
11 modulus.cs.utk.edu – 122.852 ms

traceroute to i2-dsi.ibt.tamus.edu

1 cse-danger-gateway.ucsd.edu – 0.657 ms
2 nodeb-rs-backbone.ucsd.edu – 2.827 ms
3 nodeB-6500-5500-ge.ucsd.edu – 0.706 ms
4 ucsd-gw-nodeb.ucsd.edu – 0.714 ms
5 198.32.248.185 – 0.673 ms
6 usc–ucsd.pos.calren2.net – 5.037 ms
7 abilene–usc.atm.calren2.net – 5.360 ms
8 hstn-losa.abilene.ucaid.edu – 36.987 ms
9 link2abilene.gigapop.gen.tx.us – 37.596 ms
10 link2ibt.gigapop.gen.tx.us – 38.093 ms
11 ibtx2-atm10-401.ibt.tamus.edu – 39.347 ms
12 i2-dsi.ibt.tamus.edu – 39.469 ms

A.6 Traceroutes from TAMUS to UTK and UCSD

traceroute to modulus.cs.utk.edu

1 ibtx2-atm10-1 – 0.757 ms
2 ibtx1-atm10-401 – 1.239 ms
3 198.32.236.33 – 1.795 ms
4 abilene.gigapop.gen.tx.us – 2.129 ms
5 atla-hstn.abilene.ucaid.edu – 21.754 ms
6 sox-rtr.abilene.sox.net – 21.861 ms
7 r7dh03a1-0-2.ns.utk.edu – 87.372 ms
8 192.168.101.40 – 87.301 ms
9 128.169.192.242 – 87.171 ms
10 modulus.cs.utk.edu – 87.109 ms

traceroute to freak.ucsd.edu

1 ibtx2-atm10-1 –0.759 ms
2 ibtx1-atm10-401 – 1.273 ms
3 198.32.236.33 – 1.852 ms
4 abilene.gigapop.gen.tx.us – 2.425 ms
5 198.32.8.21 – 34.278 ms
6 usd–abilene.atm.calren2.net – 34.526 ms
7 ucsd–usd.pos.calren2.net – 38.284 ms
8 198.32.248.186 – 38.191 ms
9 nodeb-ucsd-gw.ucsd.edu – 38.631 ms
10 nodeb-5500-6500-ge.ucsd.edu – 38.884 ms
11 cse-rs.ucsd.edu – 40.319 ms
12 freak.ucsd.edu – 39.676 ms

B Statistics

B.1 ANL to UTK with 8M buffers

Xfer Size	max	min	average
16K	0.40300	0.331250	0.377521
64K	0.782110	0.724233	0.763622
256K	2.041860	0.823960	1.551935
1M	3.820735	1.582323	2.998547
4M	6.075528	2.964298	4.455999
16M	7.172249	3.019620	4.323620
32M	5.590730	3.874349	4.711681

B.2 ANL to UTK with 8M buffers, using LSL

Xfer Size	max	min	average
16K	0.329159	0.282999	0.324922
64K	0.821404	0.224252	0.778078
256K	2.358894	2.151340	2.260900
1M	6.806367	1.774656	6.182508
4M	15.474262	2.971618	12.317705
16M	22.538489	2.581877	11.362988
32M	24.572602	2.627871	11.447852

B.3 UCSD to UTK with 8M buffers

Xfer Size	max	min	average
16K	0.335828	0.327585	0.332399
64K	0.671300	0.612586	0.666729
256K	1.770170	0.976332	1.643167
1M	3.208707	1.604943	2.577704
4M	4.738750	2.086455	3.742402
16M	6.517446	2.972433	4.154653
32M	5.236016	2.906003	4.031742

B.4 UCSD to UTK with 8M buffers, using LSL

Xfer Size	max	min	average
16K	0.223686	0.033156	0.207426
64K	0.602354	0.522283	0.588151
256K	1.801070	0.755655	1.742145
1M	5.168106	0.976739	4.173235
4M	9.254006	2.622432	4.622810
16M	14.895728	3.986542	6.343063
32M	13.465528	4.960214	6.395069

B.5 UCSD to UTK with 64K buffers

Xfer Size	max	min	average
16K	0.336283	0.240713	0.315922
64K	0.670764	0.598264	0.655689
256K	1.608678	0.688810	1.478414
1M	2.913219	1.108043	2.557631
4M	3.615189	1.866385	3.074215
16M	3.529395	2.790902	3.117535
32M	3.546387	2.623760	3.170910

B.6 UCSD to UTK with 64K buffers, using LSL

Xfer Size	max	min	average
16K	0.217576	0.206132	0.214729
64K	0.599110	0.287751	0.563415
256K	1.720705	1.681558	1.692182
1M	3.533808	1.483824	3.257987
4M	4.855747	1.464077	4.042588
16M	5.212078	3.793412	4.471644
32M	4.994041	3.914145	4.554841