Evaluation of Local Networking in a Lustre-Enabled Virtualization Cluster

Michael A. Murphy[1] and Heather K. Harton
{mamurph, hkeown}@cs.clemson.edu

Final Report for CpSc 852 Project
Prof. James Martin
30 November 2007

Technical Report CU-CILAB-2007-1
Cyberinfrastructure Laboratory
Clemson University School of Computing
Clemson, SC 29634-0974 USA

[1]This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Abstract

Virtualization provides a mechanism by which Virtual Organizations can be isolated from each other and made independent of physical hardware. Such hardware independence may require occasional migration of virtual machines from one system to another, necessitating the use of a networked image store, such as a distributed Lustre filesystem.

In this study, the Local Area Network (LAN) interconnecting a small research cluster was evaluated to estimate both the actual Gigabit Ethernet bandwidth of single links to server nodes and the aggregate performance of the central switch under multiple isolated loads. The primary motivation for this work was to determine if the internal network within the cluster was operating correctly, in preparation for future virtual machine migration research over a planned high-speed Wide Area Network (WAN) connection. Both theoretical and empirical procedures were employed to evaluate the performance of the physical hardware and the Transmission Control Protocol (TCP) used in the interconnection of the Lustre filesystem.

Theoretical calculations were performed to predict the behavior of TCP using both standard-sized and "jumbo" frames. For either frame size, it was determined that two TCP segments would be sufficient to saturate the Category 5 Ethernet cabling between cluster nodes. Additional TCP segments were found to be necessary to saturate switched links using jumbo frames. Independent of frame size, it was determined that larger TCP window sizes would result in more queueing at the switch or the endpoints, which could have the effect of reducing aggregate performance.

Empirical bandwidth measurements were conducted using the freely available *Iperf* and *pathChirp* tools, resulting in an estimated sustainable bandwidth of around 940 $\mathrm{Mb \cdot s^{-1}}$ for an end-to-end switched TCP connection between two nodes in the cluster. Actual TCP window sizes were estimated to be as few as 4 segments, or as many as 90 segments, with switch queueing delays between about 21 μs and 375 μs. Based on ping test results and traced TCP connection data, it was likely that the actual TCP window sizes were substantially larger than necessary to saturate the links. This behavior could have been due to sub-optimal TCP operation, slow forwarding times through the switch, or a combination of the two.

File transfer performance tests also were conducted, yielding a sustainable file transfer bandwidth between 288 and 380 $\mathrm{Mb \cdot s^{-1}}$. This substantially lower "bottleneck" bandwidth was found to be a result of slow transfer speeds at the hard disk drives. Bottleneck bandwidths appeared to vary between the two models of hard disk installed in different cluster nodes. In addition, asymmetric read-write transfer speeds were found to be possible with the model of hard disk used for the Lustre filesystem.

Based on the theoretical calculations and test results, the switch was found to be operating properly in terms of connection isolation, but some further optimization and analysis of the switch was found to be needed. Some disruptive network testing was planned once the cluster could be taken out of operational service, such as changing from standard frames to jumbo frames and re-testing the bandwidth. Additional research questions, such as the utility of link aggregation to increase Lustre node bandwidth, were identified.

Contents

1 Introduction

The use of Virtual Organizations (VO's) to organize scientific communities has become commonplace in the field of grid computing [6]. Virtualization has been proposed to provide secure, scalable Cyberinfrastructure services for sharing physical resources among different organizations [5]. Entire grids consisting of virtual machines have been constructed, which provide the end-user VO with the appearance of dedicated computational resources [1].

Virtual machines offer a variety of advantages over traditional physical machines; these advantages include dynamic reconfigurability [10] and geographical and physical system independence [2]. In order to provide hardware independence for a virtual machine, it is sometimes necessary to migrate a virtual machine to another physical system. This migration may be accomplished over a Local Area Network (LAN) connection within the same physical entity [4], or over a Wide Area Network (WAN) connection between multiple entities. Live migration of virtual machines over the Internet has been shown to be practical by recent WAN migration research [7].

The Cyberinfrastructure Laboratory at Clemson University contains a small research cluster that will be used for virtual machine testing, supporting VO's running entirely on virtual grids. This cluster is scheduled for eventual connection to Oak Ridge National Laboratory via a 10 Gigabit per second optical fiber link. The storage for this cluster is provided by a distributed Lustre file system, which will be used to test virtual machine migration over high-speed WAN connections once the 10 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ connection arrives.

Numerous possible networking issues may affect the operation of the Lustre image store. At the physical layer, the 1 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ links, and perhaps the switch controlling those links, have the potential to form a bandwidth "bottleneck" that limits the efficacy of the 10 Gigabit connection [14]. Alternatively, the location of the bottleneck could be located at the hard disks in each Lustre node. Variability in service quality may occur due to TCP/IP behavior resulting from a heavy-tailed distribution of large file transfers [12]. Also, concurrent traffic from several VO's on the network path, along with traffic originating from system monitoring tools at the physical fabric layer, may affect the cluster's ability to provide effective virtual machine image transfer [15].

To evaluate the correctness of the LAN setup between Lustre storage nodes, a small research project has been undertaken. A subset of the numerous potential networking issues was chosen for analysis as part of this study. Theoretical calculations describing upper bounds of system performance were performed, and the bandwidths of both the Ethernet links and system hard disks were measured through empirical tests. From the results of these experiments, the behavior of the system was analyzed, and conclusions were drawn about the state of the system. Desirable changes to improve system performance for virtual machine migration were identified.

The remainder of this paper is organized as follows: section 2 provides an overview of related work in the area, followed by a discussion of the motivation and objectives for this project in section 3. In section 4, the experimental methodology is described. Section 5 includes analysis of the theoretical data and empirical test results. Finally, conclusions and future work are presented in section 6.

2 Background

In order to diagnose unexpected issues properly and understand the network behavior when transferring large files, it is important to study the issues and characteristics of both the physical networks and the higher-level protocols layered atop the physical links. As a result of the applications in use in the cluster, the Transmission Control Protocol (TCP) — a reliable stream transport protocol — will be the primary focus in an attempt to understand the transport level behavior. Previous research has shown that increasing the bottleneck bandwidth and reducing self-similarity can improve the performance of a network. In addition, understanding the side-effects of concurrent connections and slow start of TCP flows allows for a more informed diagnosis of poor network performance.

2.1 Network Protocol Issues

The "bottleneck" bandwidth of a connection refers to the maximum achievable rate for a transmission from sender to receiver with no other traffic on the network, and it is distinguished from the "available bandwidth," which is characterized by the "best" transmission rate over the same path with normal network traffic. The determiner of the bottleneck bandwidth is the slowest router in the path. Paxson compared two techniques for determining the bottleneck bandwidth: packet pair and packet bunch modes (PBM). PBM, or a stricter packet pair process, should be used for an accurate measure of broadcast bandwidth. [14]

There are strong implications that can be drawn about the overall network performance when the traffic on a network is self-similar. Park et. al. studied the side-effects of self-similar traffic. As the self-similarity of network traffic increases in a UDP connection, the network performance quickly decreases. The same effect is seen with a TCP connection, but the performance decrease is much more gradual. Park et. al. attributed these performance decrease to an increase in lost packets and queue lengths that result from self-similar traffic. [12]

Qiu et. al. studied the effects of concurrent TCP connections. Two assumptions were made: (a) the bottleneck(s) between two end points result from the access link between an autonomous system and the Internet backbone; and (b) the access link bottlenecks will eventually stabilize, and all connections made from behind the same access link share the same bottleneck. [15] These assumptions were used to study concurrent TCP connections in which all connections have the same propagation delay, where there is random overhead, and where the connections have different Round-Trip Times (RTT's). In each of the cases, the capacity of the pipe determined the network behavior. [15]

The TCP Slow Start component of congestion control, introduced by Jacobson and Karels, adds another layer of complexity when attempting to measure bandwidth empirically. The TCP protocol proposed by Jacobson et. al., and used in modern implementations of TCP, includes congestion control. The effect of this addition is "self clocking" behavior, which means that it can adjust automatically to variations in delay and bandwidth. [8] When the slow start duration is calculated more quickly and precisely, network traffic can be significantly reduced [20].

2.2 Lustre Filesystem

Although the bandwidth estimation issues present with TCP/IP networks are important factors in the transfer rates and performance that will be observed, the application requirements needed to make the best use of such a network are equally important. The distributed Lustre filesystem, which relies on the TCP protocol for network transfers, has potential issues that could cause improper or ineffective use of the network. Two important factors are the links between Lustre filesystem nodes and the switch and the bottleneck bandwidth capacity of each Lustre Object Storage Target.

Sufficient network bandwidth between Lustre nodes is critical. Both InfiniBand and Quadrics network technologies have been studied, and both of these systems offer higher bandwidth than is currently available on Gigabit Ethernet. Both technologies are also based on protocols other than IP. IP-over-InfiniBand (IPoIB), which emulates the IP protocol on the InfiniBand links, has been shown to yield lower bandwidth than using InfiniBand directly, due to the additional overhead. [25] Certain Lustre filesystem operations also parallelize more efficiently than others, affecting scalability. In particular, meta-data operations do not scale with an increasing number of Object Storage Servers, or raw data storage nodes. Some performance increases can be obtained by utilizing additional meta-data storage nodes, but Lustre limits the number of MetaData Targets to two per filesystem. [25]

It has been shown that Lustre can be used as a practical wide-area filesystem over a 10 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ link. The Object Storage Server (OSS) were configured with a dual-port Qlogic card, with three Object Storage Targets (OST's — actual disk partitions) connected to each Qlogic port. Each OSS had 10 Gigabit Ethernet cards, effectively resulting in 6 OST's per node with 10 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ bandwidth. This gave roughly 1.67 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ bandwidth to each OST. Test results over the 10-$\mathrm{Gb} \cdot \mathrm{s}^{-1}$ WAN link showed that data rates of 600-700 $\mathrm{MiB} \cdot \mathrm{s}^{-1}$ were possible for single-file reads and writes from and to a 356 TB Lustre filesystem. Due to Lustre's client-side caching policy, file sizes of at least twice the main memory in the client were required to obtain accurate test results. [19]

Harney et. al. posited that a Lustre filesystem to be constructed for virtual machine migration provide both a fixed address for each machine and an IP-interoperable mechanism for communicating with the virtual machine, independent of physical location. Although limited by the slow acceptance of IPv6, Mobile IPv6 was shown to be suitable for providing this mechanism. In order to redirect connections from the virtual machine to the proper physical location, a home agent is required on the virtual machine's "home" server. [7] Mechanisms for locating virtual machines post-migration are still open research topics.

3 Motivations and Objectives

The primary motivation of this project was to validate the networking configuration of the "furnace" research cluster located at Clemson University. This cluster has been designed to support a future striped Lustre filesystem of several terabytes, which will provide an image store for virtual clustering research. Work is in progress to connect this cluster to Oak Ridge National Laboratory via a wide-area 10 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ link.

The Lustre storage portion of this cluster was initially configured with one system dedicated as a MetaData Target (MDT) node and nine systems operating as Object Storage

Targets (OST's). Each of the ten storage nodes was connected, via Gigabit Ethernet, to a central switch (star topology) that also connected fourteen general purpose computation nodes and one head node. The head node functioned as an edge router to the commodity Internet, via a Gigabit Ethernet connection to the School of Computing network.

At the time of initial construction, it was unknown whether the local area network connection was installed properly. In particular, it was not known if the switch was configured and operating correctly. Also, although the "append-mode" initial Lustre installation was known to be inefficient, it was not known if the disk partition scheme selected for Lustre was appropriate. Finally, there were no performance estimates for large file transfer operations into, and out of, individual Lustre nodes. The primary objective of this research project was to find answers to these outstanding questions and to determine if the cluster local networking interconnections were working properly. Since a re-installation of the cluster operating software was already scheduled, it was also desirable to determine if any configuration improvements, such as changing the Lustre partition layout, using link aggregation to increase node Ethernet bandwidth, or enabling jumbo frames, could be made to improve system performance.

Testing the operation of the network and disks on the cluster was complicated due to the shared nature of the system. In order to place some load on the test installation, the cluster was put into service for undergraduate instructional purposes. Since the system had to remain available at all times for these users, a number of restrictions had to be placed on the testing procedures. Specifically, no test that was potentially destructive to any filesystem, which necessitated substantive networking changes, or which required taking any system out of service for exclusive use, could be performed. This limitation was especially significant in that disk performance benchmarks, such as *IOZone* and *Postmark* [25], could not be run.

Due to the "append mode" initial installation of the Lustre filesystem, and the lack of any client device capable of 10 $\mathrm{Gb} \cdot \mathrm{s}^{-1}$ Ethernet transfers, it was not possible to test data transfers on the Lustre filesystem directly. Since entire disks on each Lustre server node had been allocated for exclusive use by the distributed filesystem, data transfer tests had to be performed on a second hard disk of an identical model. Limited free space on each second disk (upon which the operating system was also installed) dictated file size limitations for data transfer tests.

In addition to link bandwidth tests, it would have been desirable to obtain a *PTRANS* benchmark of the physical cluster, which would have enabled better comparison of the cluster's capabilities with other research and production clusters [9]. Although this benchmark was originally planned for completion by an undergraduate class, scheduling difficulties prevented its execution.

4 Methodology

To establish upper bounds on the expected performance of the system, theoretical calculations were performed. Round-trip times (RTT's) were calculated based upon the maximum propagation delay times specified for Category 5 Unshielded Twisted Pair (UTP) Ethernet cable [17]. Based on these RTT's, and assuming either standard 1500-byte Maximum Transmission Unit (MTU) Ethernet frames or 9000-byte MTU "jumbo" Ethernet frames, the behavior of TCP was predicted, including the window size required to saturate the physical

links both with and without the switch present. The bandwidth-delay product was computed as part of the calculation of the TCP window size. Expected application-level throughput was also estimated.

Additional theoretical calculations were performed to determine the "bottleneck bandwidth" [14]. Since large (gigabytes and larger) virtual machine image file transfers were expected to be disk-bound, this bottleneck was expected to occur at the hard disk. Bottleneck bandwidth was computed using the manufacturer's specifications for maximum sustained data transfer rate [18] as an optimistic upper bound.

Empirical tests of the actual system were conducted to narrow the performance estimates to more realistic bounds. The most fundamental of these practical observations involved performing file transfers between Lustre server nodes and other clients within the cluster. Secure copy (`scp`) was used to effect these operations. Transfers were performed in two directions: "inbound" transfers copied the test files from the client to the server, while "outbound" transfers copied the test files from the server to the client (with the transfer operation initiated by a client request). Distinct client-server pairs were used for the tests, with a one-to-one correspondence between client and server, in order to minimize the effects of client system peculiarities. Two separate file sizes were used to study the effects of caching at the endpoints: 1.5 GiB ("gibibytes" – binary gigabytes) and 20 GiB. To validate correct isolation of connections by the switch, the entire test suite was performed twice. First, a "sequential" test was run between each client-server pair, in which there was no other test traffic on the network. A "parallel" test was then performed, in which all client-server pairs performed the transfers simultaneously, with each connection hopefully isolated by the switch. Finally, to discriminate between hard disk performance on the client nodes and the Lustre nodes (hard disk models were not identical between the two node types), the 20 GiB file was transferred between two client nodes, then between two server nodes. For each transfer test, the total wall clock time required to perform the transfer was recorded.

Following the file transfer tests, bandwidth estimation tests were conducted using the freely available *Iperf* [20] and *pathChirp* [16] packages. *Iperf* was used to provide a measurement of available TCP throughput by connecting a single client to a single server using a single TCP stream, thereby estimating the connection bandwidth (throughput plus fixed-size headers). Peak bandwidth was also estimated through UDP "chirps," or temporally-varying bursts of traffic, using *pathChirp*. Once again, *pathChirp* tests were conducted by using a single client to connect to a single server. Each client-server pair was tested both sequentially and in parallel, to provide additional validation of connection isolation in the switch. As an additional check on the *Iperf* upper-bounds, a single discriminatory test was performed using *Iperf* to connect multiple TCP streams from a single client to a single server, thereby sharing the available bandwidth between the two connections.

In order to analyze the actual network RTT's and TCP window sizes, two additional tests were conducted using *Iperf*. In the first of these tests, ping data were collected starting a few seconds before, and continuing for a few seconds after, an *Iperf* test run. These pings were executed from a second client to the oiltank node running the *Iperf* server. The second additional test was a standard *Iperf* test, but the TCP data actually transmitted as part of the test were recorded using *tcpdump* [24] and analyzed using *tcptrace* [11].

To facilitate efficient distribution and execution of the tests, a research application called *Stoker* was employed. *Stoker*, which remains under active development in the Clemson Uni-

versity School of Computing, permitted rapid and easy execution of remote commands, both in sequence and in parallel. Additional scripts were created to post-process data, create visualizations using the open-source *GNUPlot* [22] application, and output preliminary LaTeX[23] formatting for including the visualizations. The *Gnumeric* [21] spreadsheet application was utilized to analyze the results of the data transfer tests. It was important to remember in the analysis steps that other network services were running at the time the tests were conducted, since the cluster was never taken out of production to conduct this study. Conclusions about the operation of the system were drawn, and possible future improvements to the Lustre server setup were identified.

5 Analysis

The analytical and experimental procedures were effected using the "furnace" cluster, located in room 304-D of McAdams Hall at Clemson University. This cluster featured twenty-five 1U rackmount Dell PowerEdge 860 servers, which were connected together in a star topology by means of a Dell PowerConnect 6248 Gigabit Ethernet switch with 10 Gigabit uplink capabilities (see figure 1). One of the servers, with the fully-qualified domain name `furnace.cs.clemson.edu`, functioned as a cluster head node and edge router to the commodity Internet via the School of Computing network. Ten nodes, designated as `oiltank1` through `oiltank10`, formed a distributed Lustre filesystem. The remaining fourteen nodes, designated `flamejet1` through `flamejet14`, served as general-purpose compute nodes and were employed as client hosts in the experimental bandwidth and transfer tests.

Of the ten "oiltanks," the `oiltank1` node was utilized as a Lustre MetaData Target (MDT), and the other nine hosted Lustre Object Storage Target (OST) partitions. In addition, `oiltank10` provided a Network File System (NFS) exported partition that was used for cluster user home directories. Each oiltank node had two Seagate ST3500630 Barracuda ES Serial Advanced Technology Attachment (SATA) hard disk drives, at 500 (decimal) GB each.

Both the oiltank and flamejet nodes featured dual-core Intel Xeon processors and Gigabit Ethernet connectivity. Each flamejet node had 4 GiB (binary gigabytes) of Random Access Memory (RAM), while each oiltank node was supplied with only 2 GiB RAM. Unlike the oiltank nodes, the flamejets each had a single 80 GB Western Digital hard drive. These general purpose flamejet nodes were intended for computation-bound operations, unlike the data-bound oiltank nodes.

5.1 Network Theoretical Bounds

To provide some theoretical bounds for network performance, computations for the maximum expected propagation delay, frame transmit time, *ack* transmit time, and round-trip time for a stop-and-wait protocol were performed. Since Lustre uses the sliding-window TCP protocol, the bandwidth-delay product and optimal TCP window size calculations also were performed.

5.1.1 General Theory

Since the behavior of the network under differing Maximum Transmission Unit (MTU) sizes may vary, the equations needed to predict behavior were first computed for the general case. Gigabit Ethernet, using Category 5 Unshielded Twisted Pair (UTP) cable, was assumed. Propagation delay was calculated from the specification for Category 5 UTP cable, which permits a maximum delay of 548 ns over a single twisted pair per 100 m of cable length [17], or 5.48 ns per meter. Equation 1 parametrizes the cable propagation delay d_c in terms of the cable length λ:

$$d_c(\lambda) \;=\; \lambda \cdot \left(\frac{5.48\,\mathrm{ns}}{\mathrm{m}}\right) \tag{1}$$

Since Gigabit Ethernet is assumed, the time required to send a single frame over the cable (t_f) can be calculated from equation 2. A frame header of 22 bytes and a 32-bit frame checksum are assumed.

$$t_f \;=\; (\mathrm{MTU} + 26) \cdot \left(\frac{8\,\mathrm{b}}{\mathrm{B}}\right) \cdot \left(\frac{1\,\mathrm{s}}{10^9\,\mathrm{b}}\right) \tag{2}$$

As VM migrations are assumed to be unidirectional, the *ack* messages returned to the sender are expected to be free from return data. Regardless of the MTU actually in use, these *ack* replies should consist only of frame, Internet Protocol (IP), and TCP headers, totalling 66 bytes. Equation 3 computes the time required to transmit a single *ack* (t_a).

$$t_a \;=\; 66.0\,\mathrm{B} \cdot \left(\frac{8\,\mathrm{b}}{\mathrm{B}}\right) \cdot \left(\frac{1\,\mathrm{s}}{10^9\,\mathrm{b}}\right) \;=\; 528\,\mathrm{ns} \tag{3}$$

If the switch between the sender and receiver is ignored, the Round-Trip Time (RTT) of a message and associated *ack* over the Category 5 cable can be computed from equation 4.

$$\mathrm{RTT} \;= t_f + t_a + 2 \cdot d_c(\lambda) \tag{4}$$

Since Lustre uses TCP for its communications, it is desirable to compute the necessary window size ω required to saturate the physical cable with data. Equation 6 defines this window size in terms of the Bandwidth-Delay Product (BDP) and the TCP Maximum Segment Size (MSS). MSS is defined as the link MTU minus the TCP header size (20 bytes). A method for calculating BDP is presented in equation 5.

$$\mathrm{BDP} \;=\; \mathrm{RTT} \cdot \left(\frac{10^9\,\mathrm{b}}{\mathrm{s}}\right) \cdot \left(\frac{1\,\mathrm{B}}{8\,\mathrm{b}}\right) \tag{5}$$

$$\omega \;=\; \left\lceil \frac{\mathrm{BDP}}{\mathrm{MSS}} \right\rceil \tag{6}$$

The number of segments ω from equation 6 represents the integer window size needed to saturate the physical link, ignoring the presence of the switch. Since the operation of the switch almost surely affects the operation of the TCP connection between endpoints connected through its ports, it is necessary to account for its behavior. This computation is presented in terms of TCP window size (ω) in inequalities 7 through 11. First, the ceiling

function in equation 6 is decomposed into inequality 7. To avoid a complex edge case, it is assumed that $\omega \geq 2$.

$$(\omega - 1) < \frac{\text{BDP}}{\text{MSS}} \leq \omega \tag{7}$$

Substituting equation 5 for BDP and distributing MSS yields inequality 8:

$$(\omega - 1) \cdot \text{MSS} < \text{RTT} \cdot \left(\frac{10^9 \,\text{b}}{\text{s}}\right) \cdot \left(\frac{1 \,\text{B}}{8 \,\text{b}}\right) \leq \omega \cdot \text{MSS} \tag{8}$$

Simplifying inequality 8 yields inequality 9:

$$\frac{8 \cdot (\omega - 1) \cdot \text{MSS}}{10^9} < \text{RTT} \leq \frac{8 \cdot \omega \cdot \text{MSS}}{10^9} \tag{9}$$

Decomposing inequality 9 by substituting equation 4 for the RTT and accounting for a new two-way switch delay value, d_s, yields inequality 10:

$$\frac{8 \cdot (\omega - 1) \cdot \text{MSS}}{10^9} < t_f + t_a + 2 \cdot d_c(\lambda) + d_s \leq \frac{8 \cdot \omega \cdot \text{MSS}}{10^9} \tag{10}$$

Since the RTT is already great enough to mandate a window size of $\omega > (\omega - 1)$, and the addition of $d_s > 0$ cannot decrease ω, the lower bound can be dropped. Re-arranging the relevant part of inequality 10 yields inequality 11:

$$d_s \leq \frac{8 \cdot \omega \cdot \text{MSS}}{10^9} - t_f - t_a - 2 \cdot d_c(\lambda) \tag{11}$$

The switch delay d_s is a *two-way* (or round-trip) delay. Assuming the switch is of the store-and-forward variety, part of the forward delay will involve re-transmitting the original frame. As the network speed is uniformly Gigabit, with no intermediate slower segments, this re-transmission time is taken to be the same as the original transmit time, t_f. On the return path, the *ack* will need to be stored and forwarded, resulting in a transmit time of t_a. Since the transmitting endpoint is presumed to have data to send at all times, queueing may occur in the switch if data arrives at a rate greater than the forwarding speed of the switch. Such forward queueing delay is represented by the d_q term. Queueing delay is presumed to be negligible on the return path.

$$d_s = d_q + t_f + t_a \tag{12}$$

Substituting equation 12 for d_s in inequality 11 and solving for d_q yields inequality 13, which specifies the maximum acceptable queueing delay for a TCP connection using window size ω.

$$d_q \leq \frac{8 \cdot \omega \cdot \text{MSS}}{10^9} - 2 \cdot t_f - 2 \cdot t_a - 2 \cdot d_c(\lambda) \tag{13}$$

Given the queueing delay at the switch, the end-to-end RTT can be re-computed to include the effects of the switch, using equation 14. The results of this RTT calculation can be "fed back" into equation 5, and the remaining equations and inequalities can be re-solved for the new RTT. This process could be repeated for increasing maximum values of d_q, yielding a set of conditional bounds.

$$\text{RTT} = 2 \cdot t_f + 2 \cdot t_a + 2 \cdot d_c(\lambda) + d_q \tag{14}$$

The set of conditional bounds that can be computed from the revised RTT in equation 14 are of limited utility, since the throughput of the connection is ultimately limited by the throughput of the Category 5 UTP cables, which is limited by physical properties. Theorem 1 formalizes this limitation.

Theorem 1. *If* $d_q > \frac{8 \cdot \omega \cdot \text{MSS}}{10^9} - 2 \cdot t_f - 2 \cdot t_a - 2 \cdot d_c(\lambda)$*, calculated using a switch-less RTT and corresponding* ω*, then the effect on a TCP connection will be additional queueing at the switch or the endpoints, not increased throughput.*

Proof. Equation 4 specifies the minimum RTT for a single transmission-acknowledgement exchange, ignoring the switch. Once the switch is introduced, the RTT increases by a factor of $t_f + t_a + d_q$ (subtract equation 4 from equation 14). Equation 6 specifies the number of outstanding TCP segments required to saturate the physical link, ignoring the presence of the switch. If the RTT increases, as it must once the switch is added, then equations 5 and 6 must be re-computed with the increased RTT. The number of segments ω cannot decrease as the RTT increases, and t_f and t_a are assumed to be constant. Therefore, as d_q increases, ω may increase to a new value ω_s. However, the physical cable link (ignoring the switch) is already saturated at the original ω, and $\omega_s \geq \omega$. If $\omega_s > \omega$, only ω segments can "fit" on the wire, and the remaining $\omega_s - \omega$ segments must be either queued or dropped. Assuming ω_s does not exceed the queue sizes, these extra segments will be queued. Since there is no possible way to increase the wire capacity beyond the saturation level at ω segments, throughput cannot increase. □

The immediate significance of theorem 1 is that it obviates the need to compute a set of fed-back conditional bounds for different values of d_q on the furnace cluster.

5.1.2 Cluster LAN Using Standard Frames

The furnace cluster interconnections consist of single 3.05 m cables running from each host to a central switch, utilizing a star topology. Since the switch specifications do not specify a maximum store-and-forward time [3], the presence of the switch is ignored for these calculations, and the physical connection is treated as a single 6.10 m Category 5 UTP cable.

Under these assumptions, the propagation delay d_p is computed according to equation 15, which follows from equation 1:

$$d_p = d_c(6.10) = 6.10\,\text{m} \cdot \left(\frac{5.48\,\text{ns}}{\text{m}}\right) = 33.4\,\text{ns} \tag{15}$$

The time required to transmit a standard frame, t_f, is calculated in equation 16:

$$t_f = 1526.0\,\text{B} \cdot \left(\frac{8\,\text{b}}{\text{B}}\right) \cdot \left(\frac{1\,\text{s}}{10^9\,\text{b}}\right) = 12.208\,\mu\text{s} \tag{16}$$

Assuming 528 ns for t_a (equation 3), the RTT is computed according to equation 17:

$$\text{RTT} = t_f + t_a + 2 \cdot d_p = 12.803\,\mu\text{s} \tag{17}$$

Dividing the frame transmission time t_f by the RTT yields an efficiency of 95.35% for a pure stop-and-wait protocol, which could sustain a throughput of 953.5 $\text{Mb} \cdot \text{s}^{-1}$. TCP, however, will be using a sliding window protocol, so it is desirable to know the number of segments needed to saturate the link. Following equation 5, the Bandwidth-Delay Product is computed in equation 18:

$$\text{BDP} = \left(\frac{12.803\,\text{s}}{10^6}\right) \cdot \left(\frac{10^9\,\text{b}}{\text{s}}\right) \cdot \left(\frac{1\,\text{B}}{8\,\text{b}}\right) = 1600.375\,\text{B} \tag{18}$$

The TCP window size ω follows by application of equation 6:

$$\omega = \left\lceil \frac{\text{BDP}}{\text{TCP MSS}} \right\rceil = \left\lceil \frac{1600.375\,\text{B}}{1460\,\frac{\text{B}}{\text{segment}}} \right\rceil = 2 \text{ segments} \tag{19}$$

For a direct endpoint-to-endpoint connection, a 2-segment TCP window will saturate the short Gigabit Ethernet link. However, the actual connection contains an intermediate switch, which could require an increased window size to account for switch delay. Since the unknown delay in the store-and-forward switch is considered to be queueing delay, inequality 13 can be applied to find the maximum acceptable queueing delay for the two-segment window size:

$$d_q \leq \frac{8 \cdot 2 \cdot 1460}{10^9} - 2 \cdot 12.280\,\mu\text{s} - 2 \cdot 0.528\,\mu\text{s} - 2 \cdot 0.0334\,\mu\text{s} \leq 11.152\,\mu\text{s} \tag{20}$$

In the case of standard-sized 1500 MTU Ethernet packets, a TCP window size of 2 segments should saturate the entire end-to-end link, including the switch, provided the queueing delay on the switch does not exceed 11.152 μs. Should d_q be greater, additional queueing will occur at the endpoints and/or switch by theorem 1. For the small 2-segment window size (or even for a larger ω dictated by queueing delays), TCP will exit slow-start quickly. Relative to the extended transmission times required for large VM image files, the slow-start procedure should be insignificant.

5.1.3 Cluster LAN Using Jumbo Frames

Assuming the network is using 9000 byte MTU jumbo frames, and that the network interface devices at each endpoint are capable of the full 1 $\text{Gb} \cdot \text{s}^{-1}$ speeds for both transmission and reception, the time to transmit a single jumbo frame, t_f, is calculated according to equation 21. The standard 26 bytes of frame header and checksum are assumed, for a total frame size of 9026 bytes.

$$t_f = 9026.0\,\text{B} \cdot \left(\frac{8\,\text{b}}{\text{B}}\right) \cdot \left(\frac{1\,\text{s}}{10^9\,\text{b}}\right) = 72.208\,\mu\text{s} \tag{21}$$

Although jumbo frames *permit* packet MTU's to exceed 1500 bytes, they do not *require* larger frame sizes. Therefore, the 66-byte *ack* message assumption is still valid, and $t_a = 528\,\text{ns}$. In addition, the cable lengths are exactly the same as in the previous case with standard 1500 MTU frames. Therefore, the total propagation delay over the cable, d_p, is still equal to 33.4 ns. Round-trip time, again ignoring the effects of the switch, is computed according to equation 4, resulting in equation 22:

$$\text{RTT} = t_f + t_a + 2 \cdot d_p = 72.802\,\mu\text{s} \tag{22}$$

Comparing the transmission time for the jumbo frame, t_f, with the total RTT, it is observed that t_f is 99% of the RTT. Thus, the short cable lengths in this situation cause the RTT calculations to be bound by the transmit time of the jumbo frame. A stop-and-wait protocol is fairly efficient in this case, wasting only 594 ns of every 72.802 μs, or 0.8%. Thus, 992 $\text{Mb} \cdot \text{s}^{-1}$ of throughput may be achievable with stop-and-wait.

Since TCP will use a sliding-window protocol, the Bandwidth-Delay Product (BDP), computed from the link speed and RTT, is again utilized to estimate the optimal TCP window size. Equations 5 and 6 are used to compute the window size ω, resulting in equations 23 and 24:

$$\text{BDP} = \left(\frac{72.802\,\text{s}}{10^6}\right) \cdot \left(\frac{10^9\,\text{b}}{\text{s}}\right) \cdot \left(\frac{1\,\text{B}}{8\,\text{b}}\right) = 9100.25\,\text{B} \tag{23}$$

$$\omega = \left\lceil \frac{\text{BDP}}{\text{MSS}} \right\rceil = \left\lceil \frac{9100.25\,\text{B}}{8960\,\frac{\text{B}}{\text{segment}}} \right\rceil = 2 \text{ segments} \tag{24}$$

From equation 24, a TCP window size of 2 segments should be able to keep the link operating at capacity with the RTT computed in equation 22. Since these calculations assume zero delay at the switch, and some delay is almost surely going to occur due to the store-and-forward algorithm in the switching process, it is necessary to determine the maximum tolerable switching delay for the calculated TCP window size. Initially, the two-way total switching delay d_s will be calculated in this case. Inequality 11 will be used, resulting in inequality 25.

$$d_s \leq \frac{8 \cdot 2 \cdot 8960}{10^9} - 72.208\,\mu\text{s} - 0.528\,\mu\text{s} - 2 \cdot 0.0334\,\mu\text{s} \leq 70.557\,\mu\text{s} \tag{25}$$

Substituting d_s into equation 12, and substituting values for t_f and t_a yields equation 26:

$$70.557\,\mu\text{s} = d_q + 72.208\,\mu\text{s} + 0.528\,\mu\text{s} \tag{26}$$

Since $d_q \geq 0$, equation 26 is a contradiction. Therefore, a TCP window size of 2 segments is insufficient to saturate the switch link when jumbo frames are used. Assuming no queueing delay, a revised calculation for RTT can be made by application of equation 14, resulting in equation 27:

$$\text{RTT} = 2 \cdot 72.208\,\mu\text{s} + 2 \cdot 0.528\,\mu\text{s} + 2 \cdot 0.0334\,\mu\text{s} + 0 = 145.539\,\mu\text{s} \tag{27}$$

A new TCP window size ω is computed by re-computing the BDP (equation 5) and using equation 6. Equations 28 and 29 illustrate these calculations:

$$\text{BDP} = \left(\frac{145.539\,\text{s}}{10^6}\right) \cdot \left(\frac{10^9\,\text{b}}{\text{s}}\right) \cdot \left(\frac{1\,\text{B}}{8\,\text{b}}\right) = 18192.35\,\text{B} \tag{28}$$

$$\omega = \left\lceil \frac{\text{BDP}}{\text{MSS}} \right\rceil = \left\lceil \frac{18192.35\,\text{B}}{8960\,\frac{\text{B}}{\text{segment}}} \right\rceil = 3 \text{ segments} \tag{29}$$

Back-substituting the revised RTT value and window size into inequality 13 provides the maximum queueing delay. This calculation is performed as inequality 30:

$$d_q \leq \frac{8 \cdot 3 \cdot 8960}{10^9} - 2 \cdot 72.208\,\mu\text{s} - 2 \cdot 0.528\,\mu\text{s} - 2 \cdot 0.0334\,\mu\text{s} \leq 69.501\,\mu\text{s} \tag{30}$$

From equation 24, 2 TCP segments are sufficient to saturate the link between the endpoints, assuming no switching delay. However, a minimal switching delay assumption requires 3 TCP segments to saturate the end-to-end link. By theorem 1, queueing *must* occur at the switch and/or endpoints. Provided the queueing delay does not exceed 69.5 μs, no additional TCP segments will be needed to saturate the link, and therefore no additional queueing should occur beyond that required by the switch.

As shown in inequality 20 for standard frame sizes, a maximal queueing delay of 11.152 μs is acceptable with standard 1500 byte MTU frames, in order to prevent additional queueing. For jumbo frames, a 69.501 μs delay is acceptable at the sufficient window size. This jumbo frame d_q is 6.23 times the value of the standard frame d_q, with a frame MTU that is 6 times larger. Therefore, provided that the queueing operation is no worse than $O(N)$ (and does not have a large constant term or coefficient), the relative queueing time of jumbo frames should not be significantly greater than that of regular frames. However, the necessity of additional queueing demanded by theorem 1 could make the actual end-to-end communication times for jumbo frames longer than for regular frames, owing to the longer absolute queueing delays. Even so, TCP slow start should be exited quickly with jumbo frames, making the slow start behavior insignificant in comparison to the long VM image transmit times.

5.2 File Transfer Performance

Empirical file transfer tests were conducted using secure copy (`scp`) to transfer large files between a single client node in the cluster and a single Lustre node. Two different file sizes were used: a 1.5 GiB file actually containing a virtual machine image, and a 20 GiB random data file representing a larger VM image. Each file was first copied from the client machine (a "flamejet" node) to the corresponding Lustre server (an "oiltank" node with the same number) in an "inbound" (to the server) copy test. In a later test, the files were deleted from the client nodes and copied back from the servers (an "outbound" transfer). Copy operations were initiated by the client in each case.

Each set of copy operations (inbound and outbound) was performed twice. In the first test suite, each client-server pair was tested sequentially, so that only 1 copy operation was occurring at any given time on the LAN. To verify that connections were properly isolated from each other by the switch, a second parallel test suite was run, in which each client-server pair was tested simultaneously. Results of the tests are summarized in tables 1 through 3 and figures 2 through 12.

As shown in table 1 and figure 2, transfer times for the 1.5 GiB file tended to be relatively short, averaging about 34 seconds inbound and about 32 seconds outbound. The differences between sequential and parallel execution were minor, and much of this difference is likely due to overhead in *Stoker*, which in its present form is not particularly efficient. Standard deviations for both outbound tests, in which files were copied from the server to the client, were relatively much lower than those for the inbound tests.

For the 20 GiB file size, outbound transfer times were substantially higher than inbound times (table 2 and figure 3). More variation between outbound tests was observed than for inbound tests, as evidenced by a larger standard deviation in the results. This relative

configuration of transfer times is opposite that of the small transfer times and may be due to differences in the hard drives and system memory in the different machines. Each flamejet node uses a lower-performance hard disk than each oiltank node, but each flamejet also has twice the memory of each oiltank. The 1.5 GiB files are small enough to be mostly or completely buffered in memory during the transfers; such complete buffering is not possible for the larger 20 GiB files.

The distribution of transfer times for the smaller file size, as a function of client-server pair, were fairly random with small temporal variations between client-server pairs (figures 4 through 7). Maxima occurred at client-server pairs flamejet4-oiltank4 and flamejet7-oiltank7 inbound, and flamejet1-oiltank1 and flamejet10-oiltank10 pairs outbound. However, these maxima were not particularly significant, given the short total transfer times. For the large-file transfers, interesting minima were observed with client-server pairs flamejet6-oiltank6 and flamejet7-oiltank7 (figures 8 through 11) on both inbound and outbound transfers. Another transfer minimum was also noted at the flamejet1-oiltank1 pair, which could be explained by the fact that the oiltank1 node is running as a Lustre MetaData Target (MDT) instead of as an Object Storage Target (OST), which might have reduced intra-Lustre communications at that node. The minima in the other two client-server pairs occurred for unknown reasons and may warrant further investigation.

Since each type of transfer endpoint (flamejet or oiltank) contained somewhat different hardware, an additional transfer test was conducted using the 20 GiB file to identify performance differences between the system types. Each flamejet node was equipped with an inexpensive hard disk of lower performance specification than each oiltank. As expected, a file copy operation between one flamejet and another flamejet required more time than the same copy operation between two oiltanks (figure 12).

Based on the average transfer times for the 20 GiB file in table 2, the end-to-end bandwidth of the entire system including the hard drives was estimated by dividing the 20 GiB file size by the transfer times. An inbound mean throughput of 47,510,164 $\mathrm{B} \cdot \mathrm{s}^{-1}$, or approximately 380 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$, was observed. The outbound mean throughput was calculated to be 35,992,001 $\mathrm{B} \cdot \mathrm{s}^{-1}$, or approximately 288 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$. Both results were calculated using the sequential transfer test means, and the difference of nearly 100 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$ indicates a substantial performance discrepancy exists between inbound and outbound transfers for files too large to be buffered in memory. This disparity could be due to non-uniform read-write speeds in the drives (i.e. a drive could take longer to read data than to write data, or vice-versa) or to bottleneck bandwidth differences between drive models (see section 5.4).

Since the flamejet-flamejet copy operation required more time than the inbound flamejet-oiltank operations, a disparity in hard disk performance between the two node types is likely. Furthermore, since the oiltank-oiltank transfer time was also higher (see figure 12) than the inbound flamejet-oiltank operations, there is an increased chance that the oiltank node hard disks have greater write performance than read performance. Therefore, both bottleneck bandwidth differences and non-uniform read-write speeds are possible within the cluster. Such differences have the potential to result in asymmetric migration speeds: migration of virtual machines in one direction (into or out of the cluster) may occur at a different rate than migrations in the other direction.

5.3 Measured Available Bandwidth

Bandwidth measurements were performed using both *Iperf* and *pathChirp*, which are two packages designed for end-to-end network bandwidth estimation. Each of these tests was performed between a pair consisting of a flamejet client and an oiltank server. Two suites of each test were performed: one sequential suite in which only one bandwidth measurement was occurring at any time on the LAN, and one parallel suite in which each client-server pair was tested simultaneously.

5.3.1 Iperf Bandwidth Tests

Iperf tests were completed first and reported an available end-to-end bandwidth of approximately 940 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$ (including the switch). Figures 13 through 32 visualized these results, which showed consistent available bandwidth for the majority of the time, with occasional downward spikes. These spikes were attributed to communications between Lustre server nodes, communications between the Lustre client nodes and the Lustre file system generally, the Ganglia monitoring system, and other routine monitoring and synchronization systems running on the cluster nodes. There was no significant difference between the sequential and parallel test results, which indicated proper connection isolation by the switch.

As further validation of the available bandwidth, an *Iperf* test was run in which two clients connected to a single server. The results of this test were visualized in figures 33 through 36. In figures 35 and 36, the maxima in bandwidth usage by one connection corresponded to the minima in bandwidth usage for the competing connection. This behavior was expected, since the sum of the bandwidth used by both connections could not exceed the physical capacity of the Gigabit Ethernet connection.

5.3.2 Iperf Window Sizes and RTT's

In order to estimate the effects of the switch on the connection, another *Iperf* test was conducted in which a second client node repeatedly pinged the *Iperf* server machine before, during, and after the test. The presence of the ping operation did not affect the *Iperf* results (figure 37), but the ping RTT's were greatly affected by the *Iperf* operation (figure 38). Prior to the start of the *Iperf* run, approximately 25 ping RTT's were observed, with the reported RTT close to 100 μs. Once the *Iperf* communications began, the measured RTT increased to over 300 μs, indicating substantial amounts of queuing time at the switch. The ping RTT's returned to approximately 100 μs once the *Iperf* test ended, suggesting a queueing delay d_q of around 200 μs induced by the *Iperf* TCP connection.

A final *Iperf* test was conducted with the client communications recorded using *tcpdump* and analyzed with *tcptrace*. The behavior of *Iperf* under the recording operation did not appear to vary substantially from the un-traced behavior (figure 39). Results of the test were presented in table 4. A considerable amount of data – nearly 33 GiB – was transferred during this test. Over 2 million individual IP packets were sent, averaging 16,646 bytes (close to the maximum per packet). Based on the window advertisements, when these IP packets were divided into frames, 5888-byte windows were utilized for the TCP communications, which corresponds to 4.03 segments. However, an average of 54,745 bytes of data were allowed to be un-acked at any time (with a maximum of 130,721 bytes). If TCP/Reno-style sliding windows

were used, these outstanding data sizes corresponded to window sizes of 38.18 segments and 89.53 segments, respectively.

Traced RTT's ranged from 0.1 to 11.6 ms, with an average of 0.4 ms. These times may be unreliable, however, as *tcpdump* is sensitive to delays within the kernel at the tracing endpoint, as well as clock skew in the event that timestamping was employed during the trace procedure [13]. It is known that the cluster node clocks are definitely *not* synchronized, as synchronization operations tend to break the PBS daemons on the oiltank nodes. Assuming endpoint delays and clock skew have not affected the RTT calculations, the application of equations 6 and 14 results in a window size of 34.25 (integer value 35) and a maximum queueing delay d_q of 374.46 μs. In this case, queueing delay accounts for almost 94% of the total 400 μs RTT.

Alternatively, if the advertised window size of 5,888 bytes is considered, an alternate RTT estimate can be calculated by decomposition of equations 6 and 5. The first step in the decomposition is recognizing that equation 6 requires an integer number of segments for ω as a result of the ceiling operation, but the actual TCP implementation operates in bytes instead of segments. Since 5,888 bytes corresponds to 4.03 segments, and increasing to 5 segments for the fixed MSS would result in an increased BDP (and therefore an inflated RTT value), the ceiling operation is dropped and 5,888 bytes is used in the numerator. Equation 31 presents this setup for the window size in bytes, ω_B:

$$\omega_B = \frac{5,888}{1,460} \tag{31}$$

Thus, for back-solving equation 5, the window size of 5,888 bytes is substituted for the BDP. The resulting RTT from this calculation is 47.104 μs. Solving equation 14 for this RTT value yields a queueing delay d_q of 21.57 μs.

There is, of course, a disparity between the theoretical calculations and the observed RTT and window size values. From theorem 1, a window size larger than 2 TCP segments must result in queueing when only the wire is considered. However, the store-and-forward nature of the switch demands queueing, unless the switch can forward a packet in under 11.152 μs (inequality 20). The evidence indicates that a window size of more than 2 segments, perhaps significantly more than 2 segments, is in use in the observed TCP communications, which implies queueing. However, a key limiting factor in the theoretical calculations is that TCP/Reno is assumed. It is likely that the relatively recent Linux kernels in use on the cluster are using a more enhanced version of TCP, with features such as delayed *ack* messages and stateful "learning" mechanisms that can optimize new connections over the same link using observations from prior connections. Since the cluster was never taken out of service to effect this study, a substantial history of TCP connections would have been observed by the kernels on each machine. Finally, the managed switch does appear to have link and path learning capabilities, as evidenced by QoS and spanning tree settings found to be present in the configuration interface. It is thus possible that the switch has monitoring and management capabilities above the network layer, which could potentially affect the handling of TCP connections.

Another TCP concern is the wide range of possible observed RTT values. Observed round-trip times range from a potential low of 47.104 μs based on the *tcptrace* window advertisement information, to an average high of 0.4 ms (400 μs), also calculated by *tcptrace*. The observed

RTT's of over 300 μs from the ping test reinforce the possibility of the higher estimate. Since a theoretical RTT for saturating the Category 5 UTP cables is an order of magnitude lower (equation 17), there is a potential that switch store-and-forward performance is poor, or that the TCP protocol is behaving sub-optimally on the cluster. Both of these issues could be occurring, confounding the analysis: a longer queueing delay on the switch will require TCP to increase its window size to account for the longer RTT, while an increased window size will result in additional queueing (theorem 1) and therefore increased queuing delay and total round-trip time. Since no packets were dropped in the tests (table 4), TCP was at least able to avoid overrunning the queues, permitting sustained transfer speeds without falling into congestion avoidance (see figures 13 through 32).

5.3.3 pathChirp Bandwidth Tests

Following the *Iperf* tests, two suites of *pathChirp* tests were executed with the same configuration: a sequential suite and a parallel suite, each with *pathChirp* running on a single client-server pair. Figures 40 through 59 summarize the results of these tests, which generally confirm the *Iperf* bandwidth estimates of around 900 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$. However, *pathChirp* required manual tuning of packet size (1500 bytes) and interrupt coalescence (5 actual packets per interrupt) to report "correct" results, which makes the results of these tests somewhat suspect. Furthermore, *pathChirp* occasionally reported unbelievable values for the bandwidth, such as the spike over 1050 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$ shown in figure 55. A *tcpdump* and UDP-mode *tcptrace* of the *pathChirp* data did show that many fewer packets were transmitted to perform the test (around 72,000 UDP packets total), as was claimed by the authors of the tool [16]. However, the tuning issues and questions about the validity of the interrupt coalescence calculations render *pathChirp* less useful for realistic bandwidth estimation purposes since *a priori* estimates of the bandwidth (from *Iperf* were needed to perform the tuning.

5.4 Hard Disk Bottleneck Bandwidth

At the start of the study, the location of the "bottleneck bandwidth" in the system was believed to be at the hard drives. Since the Lustre filesystem nodes are the interesting machines in the cluster for virtual machine migration purposes, the specifications for the Seagate ST3500630NS 500 GB Barracuda ES drives [18] were used to perform the theoretical calculations. According to the manufacturer's specification, these drives are supposedly capable of a sustained data transfer rate of 72 decimal megabytes per second. Equation 32 converts this figure into network-bandwidth units:

$$\text{Bottleneck Bandwidth} = 72 \frac{\mathrm{MB}}{\mathrm{s}} \cdot \left(\frac{8\,\mathrm{b}}{\mathrm{B}} \right) = 576 \frac{\mathrm{Mb}}{\mathrm{s}} \tag{32}$$

As observed in the data transfer tests (tables 1 and 2), the actual transfer rates for data stored on the hard drives were between 288 and 388 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$ (see section 5.2) — well below the manufacturer's advertised specifications. Since the *Iperf* tests identified available network bandwidth around 940 $\mathrm{Mb} \cdot \mathrm{s}^{-1}$, the bottleneck almost surely lies at the drive. It is possible that the manufacturer quotes the specified transfer rate as the theoretical peak transfer rate between the hard drive cache and system memory, as opposed to the actual transfer rate of

data to and from the platters, which would explain the differences in observed and theoretical bottleneck bandwidth.

6 Conclusions and Future Work

Conclusions are presented regarding the correctness of the cluster networking setup and expected performance bounds in section 6.1. Link aggregation and jumbo frame options are also explored. Following the general conclusions, threats to the validity of the conclusions (caused by weaknesses in the experimental methodology) are presented in section 6.2. Finally, future changes to the cluster, along with planned and potential extensions of the current study, are discussed in section 6.3.

6.1 Conclusions

Based on the observed isolation of simultaneous connections between different system pairs and the observed end-to-end TCP bandwidth around 940 $\text{Mb}\cdot\text{s}^{-1}$, the switch appears to have sufficient internal capacity to handle the multiple loads of a virtualization research cluster. Although the settings on the switch may not yet be optimal, it does appear to be operating correctly under the test conditions. The Category 5 UTP cables between each of the Lustre storage nodes and the switch, and between the ten utilized client nodes and the switch, are also apparently in operational condition.

End-to-end bandwidth is 94% of the theoretical peak bandwidth, even using standard-size 1500 MTU frames for large transfers. Enabling jumbo frames has the potential to improve performance by decreasing overhead; however, the opposite result could also occur since jumbo frames could lead to more queueing and thus longer queue delays. Empirical tests will be needed to determine the true benefits or drawbacks to enabling jumbo frames.

For large file transfers, the bandwidth bottleneck lies at the hard drive and is in the approximate range of 288 to 380 $\text{Mb}\cdot\text{s}^{-1}$ for a single drive. Doubling the number of Lustre Object Storage Targets by striping data across both disks in each server node has the potential to double the bottleneck bandwidth and increase transfer performance, provided bus capacity is sufficient and stripe overhead is not excessive. For smaller file transfers, the system memory at each endpoint can be used as a buffer to reduce the impact of the bottleneck. This buffering should be beneficial for small filesystem operations that result from executing virtual machines directly from the distributed storage. Large transfer bottlenecks should be limited to machine migration operations.

Link aggregation is of uncertain utility in the research cluster. In the case of large transfers conducted at migration time, a bottleneck bandwidth of at most 760 $\text{Mb}\cdot\text{s}^{-1}$ (twice the single-drive bottleneck once data are striped over both drives in each node) will still exist at each node. Based on the *Iperf* and *pathChirp* tests, a single Ethernet link can support sustained data transfer rates of 940 $\text{Mb}\cdot\text{s}^{-1}$. Thus, maximum bottleneck bandwidth is only about 80% of the measured single-link bandwidth. Even if multiple simultaneous migrations occur, the hard disk bottleneck bandwidth will have to be shared among the connections, so increasing the total available Ethernet bandwidth via link aggregation will not yield any benefits for disk-bound migration operations. Link aggregation does have the potential to

improve aggregate performance for multiple simultaneous connections that are not bound by the hard disk bottlenecks, however.

With the increased bottleneck bandwidth resulting from planned Lustre partitioning changes, the distributed filesystem should be able to provide acceptable performance both for conducting virtual machine migration tests and for distributed storage of operating virtual machines. As a result, the research cluster in 304-D McAdams Hall at Clemson University should provide a state-of-the-art platform for enabling full virtualization of compute resources. Such virtualization will allow multiple Virtual Organizations to share the physical compute fabric in a secure manner.

6.2 Threats to Validity

Although the results of the cluster LAN research are encouraging, several factors are present that could threaten the validity of the results. In particular, the research was conducted under fairly stringent limitations since the cluster was not taken out of service for the purpose of testing. As a result, the bottleneck bandwidths are at best estimates of the true hard drive bandwidth. It is possible that disk or filesystem benchmarks could show a greater bottleneck bandwidth than the secure copy tests were able to demonstrate. Furthermore, secure copy may introduce overhead not present in direct Lustre operations, which would also result in an increase in available bottleneck bandwidth. A fairly modest increase in hard disk bandwidth, perhaps from 380 to 450 $\mathrm{Mb \cdot s^{-1}}$, could invalidate the conclusion that link aggregation would not yield performance benefits for virtual machine migrations. Furthermore, the effects of jumbo frames on all the empirical test results are unknown.

Conversely, it is also possible that the estimates obtained as a result of this study are too optimistic. Striping and filesystem overhead in Lustre could be greater than secure copy overhead, which would have the effect of reducing the estimated bottleneck bandwidth. Poor virtual machine migration performance, and perhaps even poor virtual disk I/O performance, could result in such a case. However, this case is considered less likely than the case of performance under-estimation, since the cluster network tests were conducted while the system was in production and under Ethernet load from system services such as Ganglia.

Effective network performance could be contextually over-estimated in this study, since the effects of virtual networks between virtual machines on the cluster were not considered in the total bandwidth analyses. Although the Lustre filesystem nodes may be able to provide suitable independent bandwidth, virtual machines running on client flamejet nodes may cause bottlenecks to appear at the client-side. Since the oiltank nodes will not be running hypervisors, virtual networking should not affect the available bandwidth between each oiltank and the switch. However, this bandwidth could be under-utilized by Lustre clients in the event that large amounts of disk-independent virtual network traffic are saturating the physical links between the flamejets and the switch. Link aggregation, client Quality of Service settings, or other remedial actions may be needed to ensure that a new bottleneck does not form.

Finally, it is likely that the theoretical estimates of window sizes for TCP transfers will be difficult to relate to the actual networking operation. For computational and cognitive tractability, the relatively simple TCP/Reno implementation was used as the basis for the theoretical analysis. The actual TCP implementation in the Linux kernel is expected to differ

from TCP/Reno. Furthermore, the TCP/IP implementations between flamejet and oiltank nodes may have varied from each other during the empirical testing, since two different kernel versions were in use on the different node types.

6.3 Future Work

In December 2007, the cluster will be taken out of service for re-installation of the operating system software on all nodes. A uniform Linux distribution and kernel version will be installed on each system, so that different kernels are not in use across the physical compute layer. The kernel will be customized, permitting the selection of specific TCP/IP extensions. System clocks will be kept synchronized in the upgraded cluster, so that clock skew should be minimal (or perhaps even insignificant) between nodes.

Prior to the re-installation procedure, the switch will be changed to support jumbo frames. The *Iperf* and file transfer tests will be repeated with the 9000-byte MTU frames enabled, and the performance differences will be analyzed in relation to the standard-size frame performance reported in this study. While the systems are down in preparation for re-installation, additional wiring will be installed to enable link aggregation on each oiltank node. However, aggregation of Ethernet bandwidth may remain disabled on the switch unless and until saturation of the single links is actually observed in the network.

As part of the upgrade procedure, the switch settings will be analyzed so that optimal adjustments may be made to the switch management features. Additional *Iperf* tests may be used to evaluate the effects of management settings changes, so that end-to-end bandwidth can be maximized and switch overhead minimized. Domain expertise for understanding switch configuration options will be sought from a certain networking professor.

One part of the new software load to be installed on the cluster will be better passive instrumentation for networking monitoring. Since *Iperf* is rather disruptive in terms of the amount of bandwidth used, passive monitoring of Linux kernel parameters through the Web 100 kernel patch is planned [20]. This extra instrumentation should be useful for fine-tuning switch and network configuration settings post-installation, and the additional monitoring capabilities should be helpful in determining the impact of virtual networks layered upon the physical LAN.

Beyond the software re-installation, there are several potential major research projects that could be conducted as an extension of this study. In particular, the effects of an overlaid virtual network on top of the physical layer could be considered. Differentiated levels of service, provisioning of limited physical link capacity over virtual network links, and prioritization of physical-level management traffic over virtual VO traffic are a few of the additional research topic areas that remain open for further investigation.

A Tables

Table 1: 1.5 GiB File Transfer Times in Seconds

	Inbound		Outbound	
System Pair	Sequential	Parallel	Sequential	Parallel
1	32.446	32.87	32.71	32.79
2	33.181	32.54	31.743	32.19
3	34.819	32.45	31.901	32.67
4	37.449	37.57	31.941	32.13
5	32.902	32.4	32.021	32.66
6	34.047	33.39	32.124	33.06
7	39.12	38.79	31.822	32.43
8	32.756	33.46	32.271	32.49
9	34.592	34.78	31.964	32.34
10	35.596	33.02	32.671	32.9
Mean	34.6908	34.127	32.1168	32.566
Std Dev	2.1794	2.2640	0.3366	0.3041
Median	34.3195	33.205	31.9925	32.575

Table 2: 20 GiB File Transfer Times in Seconds

	Inbound		Outbound	
System Pair	Sequential	Parallel	Sequential	Parallel
1	424.806	424.45	561.165	537.43
2	450.777	450.23	595.677	611.11
3	469.475	460.63	604.786	594.11
4	460.542	447.99	590.421	586.92
5	471.624	471.41	638.563	638.14
6	426.683	426.61	545.708	534.08
7	423.235	425.54	545.791	546.55
8	452.73	452.14	606.582	616.56
9	472.163	473.49	644.974	627.52
10	468.016	469	632.891	638.8
Mean	452.0051	450.149	596.6558	593.122
Std Dev	20.1009	19.1218	36.5921	40.7774
Median	456.636	451.185	600.2315	602.61

Table 3: Large File Transfer Within Groups

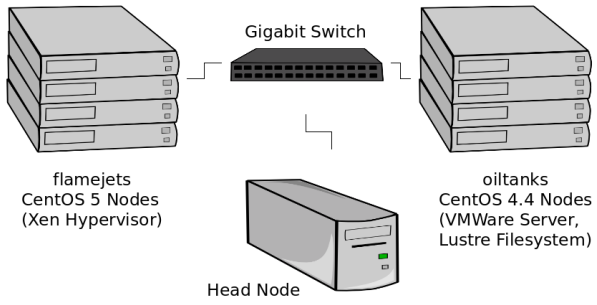
Node Group	Time (sec)
flamejets	627.684
oiltanks	526.132

Table 4: Iperf TCP Statistics

Parameter	Value	Units
Data Packets Sent	2,120,239	packets
Total Data Sent	35,294,122,288	bytes
Maximum Window Advertised	5,888	bytes
Minimum Window Advertised	5,888	bytes
Maximum Un-acked Data	130,721	bytes
Average Un-acked Data	55,745	bytes
Throughput	117,640,771	$\mathrm{B} \cdot \mathrm{s}^{-1}$
Re-transmitted Packets	0	packets
Minimum Observed RTT	0.1	ms
Average Observed RTT	0.4	ms
Maximum Observed RTT	11.6	ms

B Figures

Figure 1: Topology of the furnace Research Cluster



B.1 Transfer Test Figures

Figures 2 and 3 summarize the results of the secure-copy file transfer tests. The terminology convention used in these figures is that "inbound" copy operations are copies *into* the Lustre (oiltank) nodes. That is, inbound operations are copies from the flamejet client to the oiltank server. "Outbound" operations are the reverse: a flamejet client requests the remote file on the oiltank server. Operations marked as "sequential" are carried out one transfer at a time; "parallel" transfers occur with all 10 client-server pairs performing the copy operation at the same time (but isolated from each other by the switch). Details of each type of transfer for each size of file are shown in figures 4 through 11.

Figure 12 describes the difference between a copy operation among two flamejet nodes and a copy operation among two oiltank nodes. The purpose of this test was to identify performance differences between the models of hard drive used in each type of node.

Figure 2: Small File Transfer Comparison

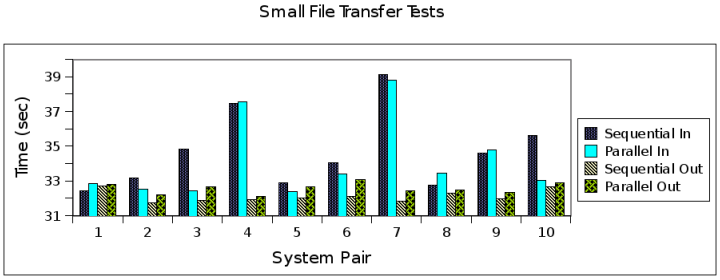


Figure 3: Large File Transfer Comparison

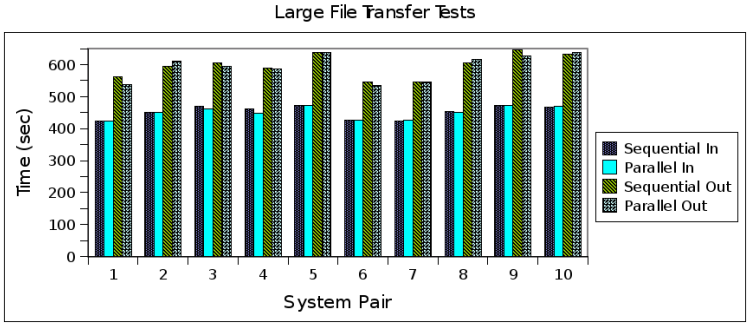


Figure 4: 1.5 GiB Sequential Inbound Transfer

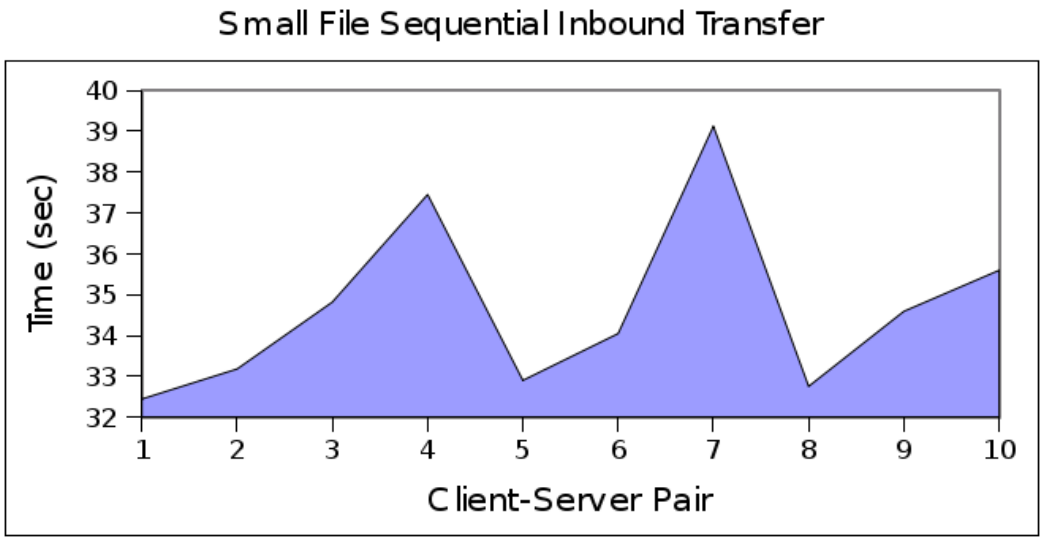


Figure 5: 1.5 GiB Parallel Inbound Transfer

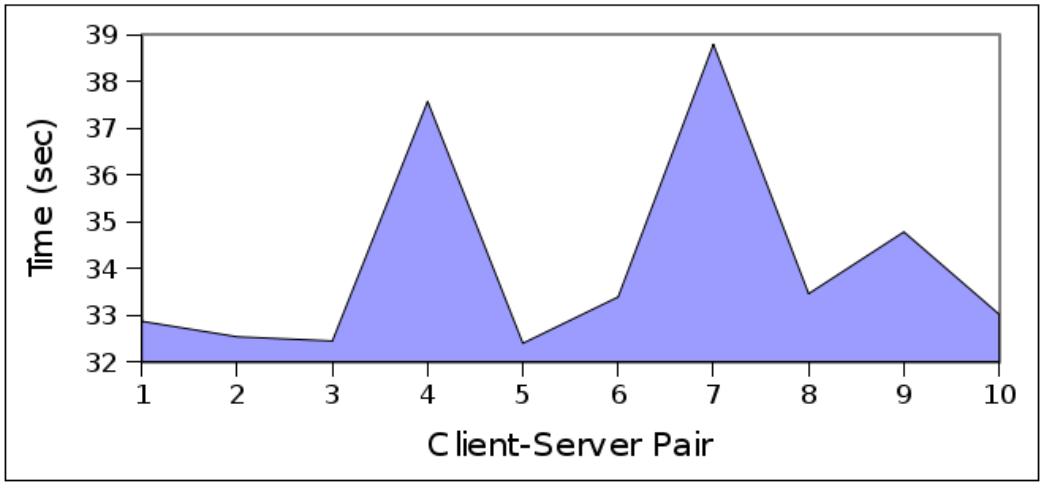


Figure 6: 1.5 GiB Sequential Outbound Transfer

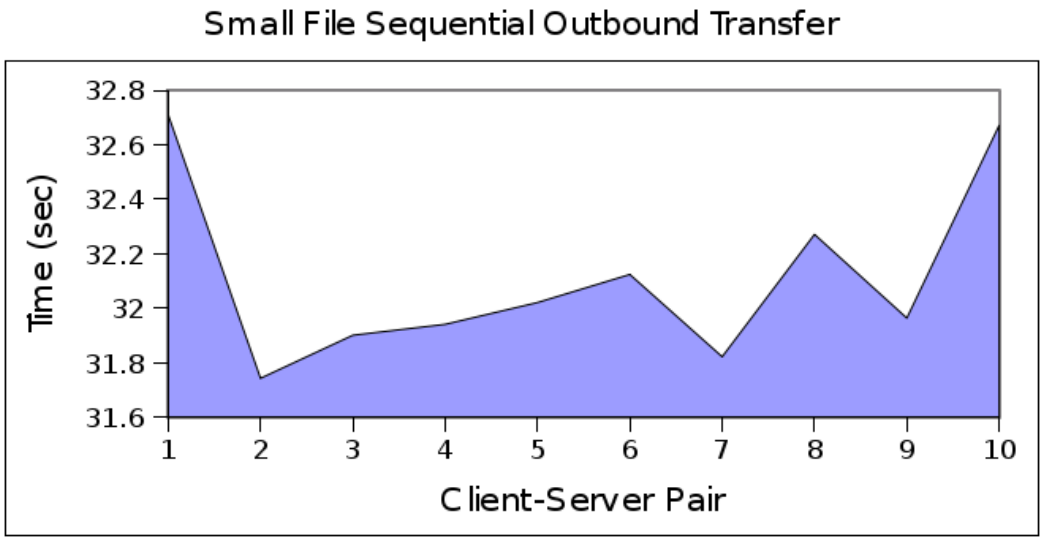


Figure 7: 1.5 GiB Parallel Outbound Transfer

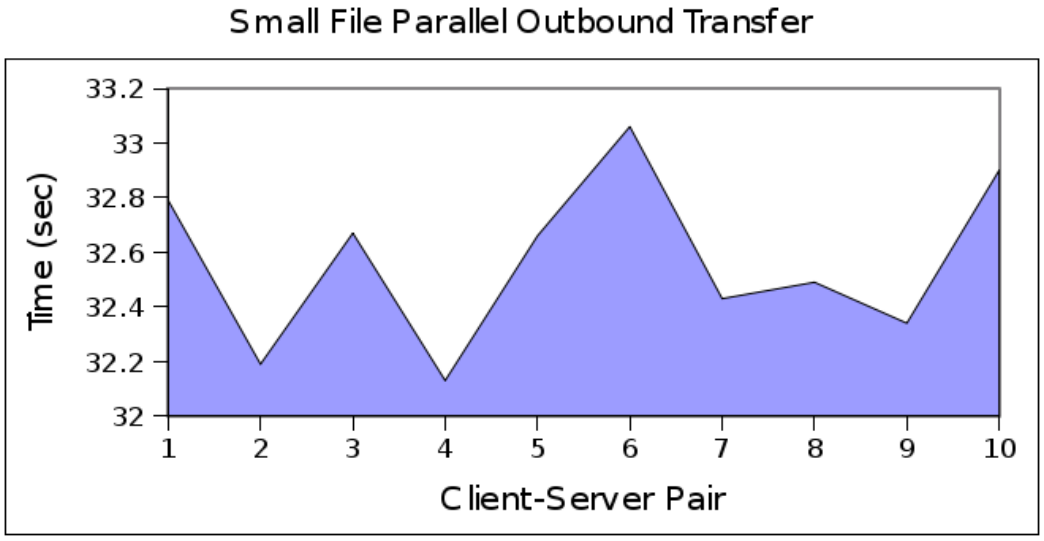


Figure 8: 20 GiB Sequential Inbound Transfer

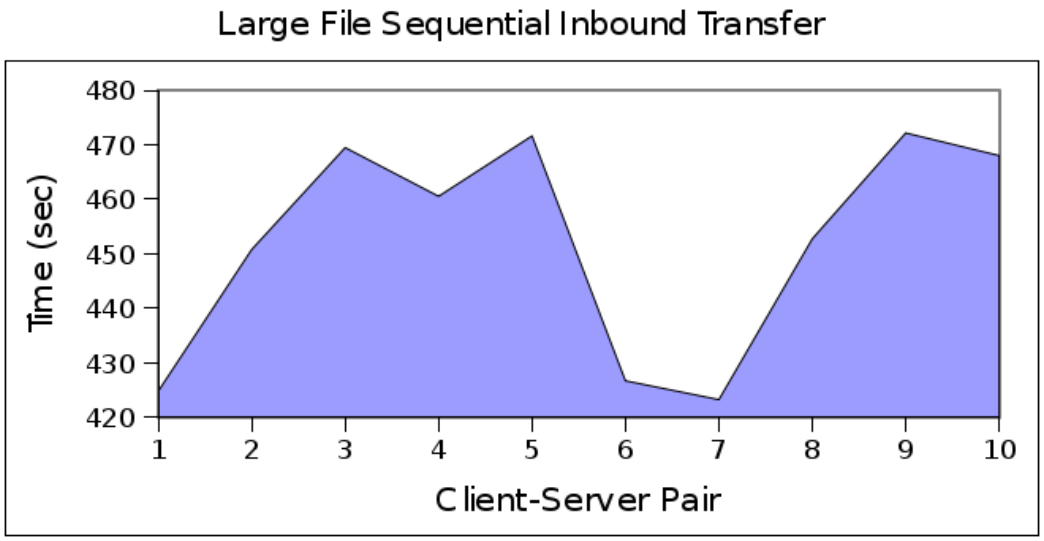


Figure 9: 20 GiB Parallel Inbound Transfer

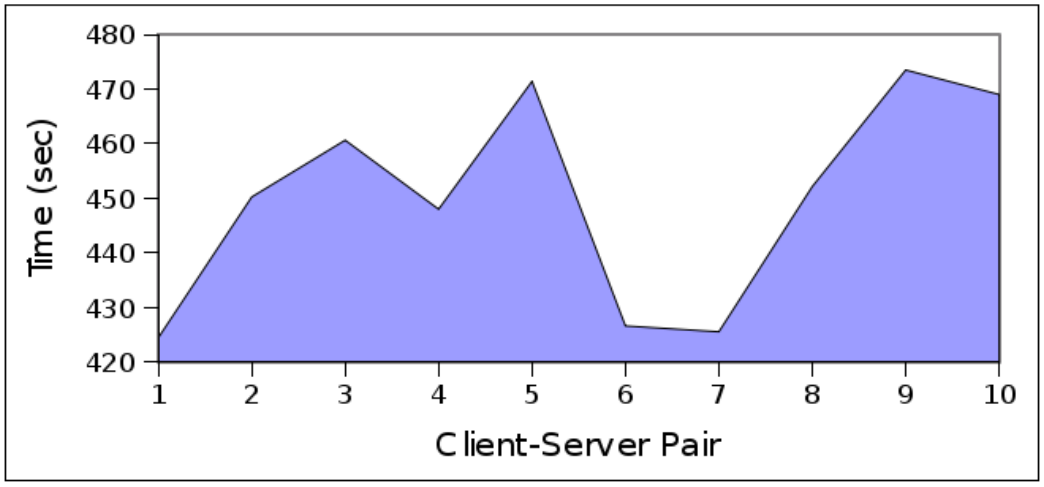


Figure 10: 20 GiB Sequential Outbound Transfer

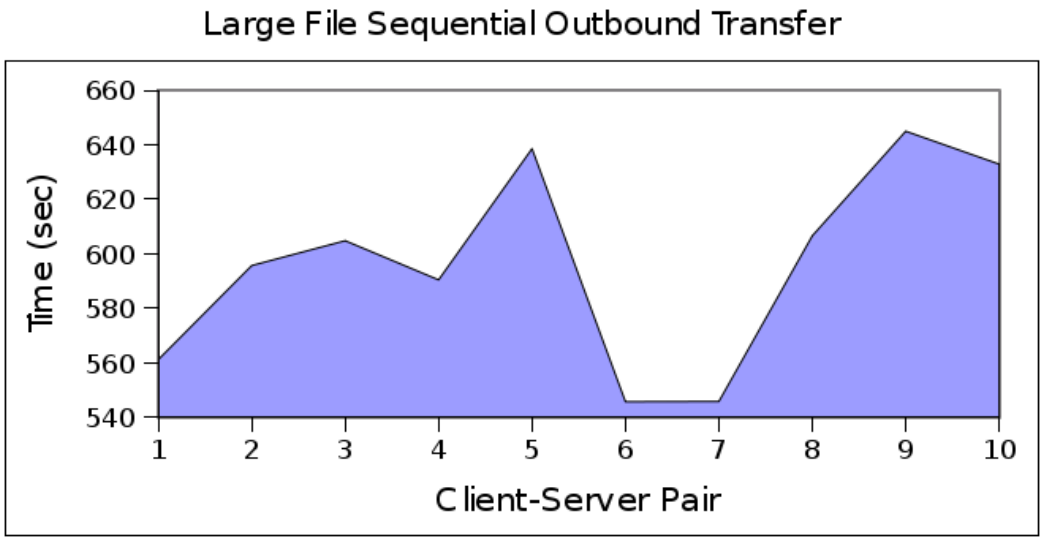


Figure 11: 20 GiB Parallel Outbound Transfer

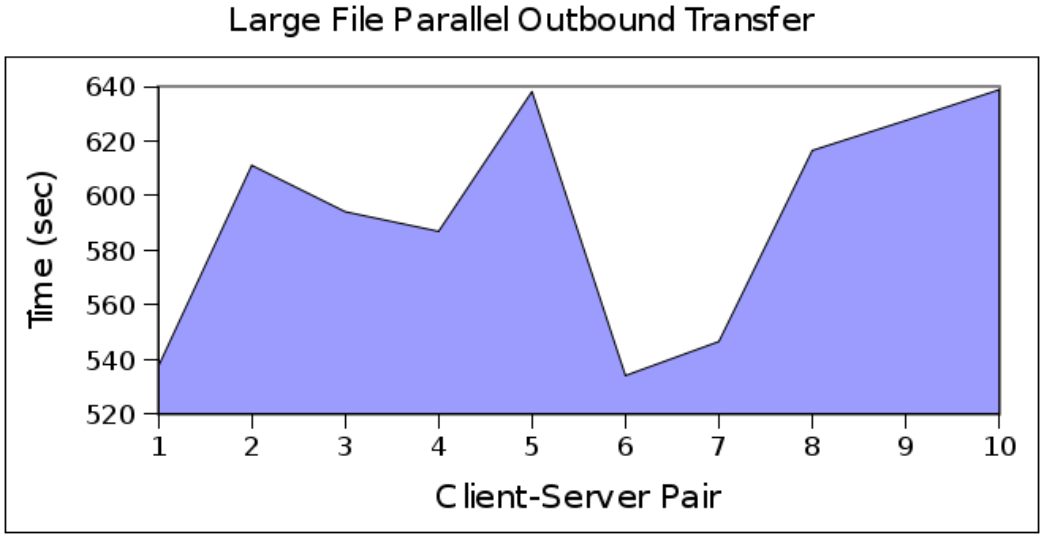
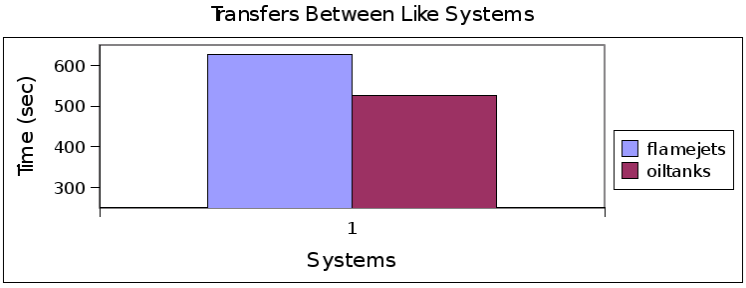


Figure 12: Comparison Between flamejet and oiltank Nodes



B.2 Iperf Test Figures

The results of the *Iperf* bandwidth measurement tests are illustrated in figures 13 through 32. Each figure shows the results of a TCP bandwidth-probing test between a flamejet client and a corresponding oiltank server. Two tests among each client-server pair were performed: a "sequential" test, in which only 1 pair was tested at any given time; and a "parallel" test, in which all 10 pairs were tested at once (but isolated from each other by the switch).

Figures 33 through 36 illustrate the TCP behavior when two clients (flamejets 1 and 2) connect to a single server (oiltank3). Bandwidth is shared between the two clients. Figures 37 and 38 visualize the effects of a concurrent *Iperf* connection and ping test to a single server. The *Iperf* results of a traced client-server bandwidth estimation test are visualized in figure 39.

Figure 13: flamejet1-oiltank1 Sequential Test

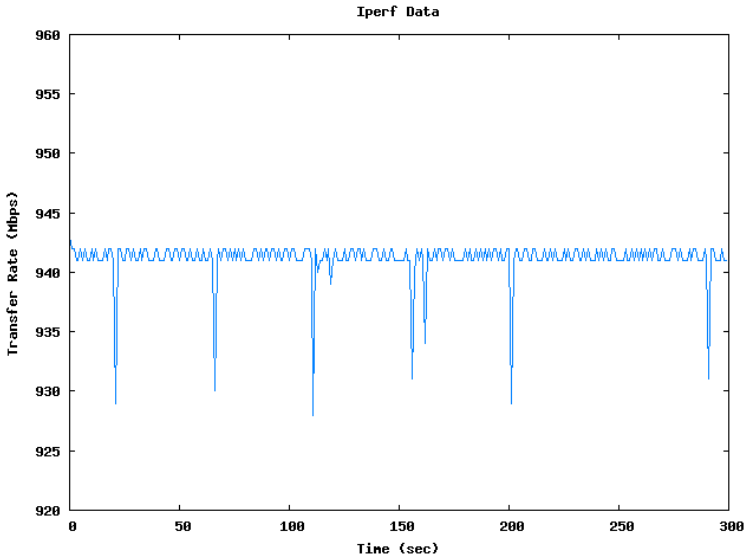


Figure 14: flamejet1-oiltank1 Parallel Test

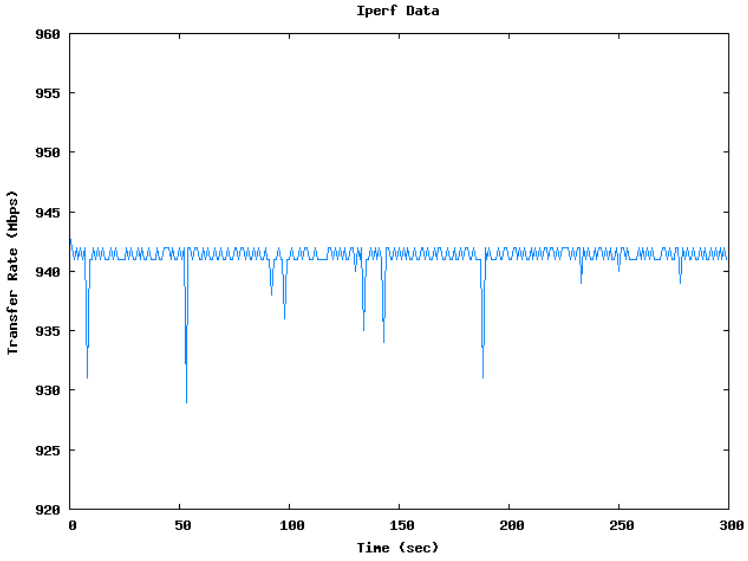


Figure 15: flamejet2-oiltank2 Sequential Test

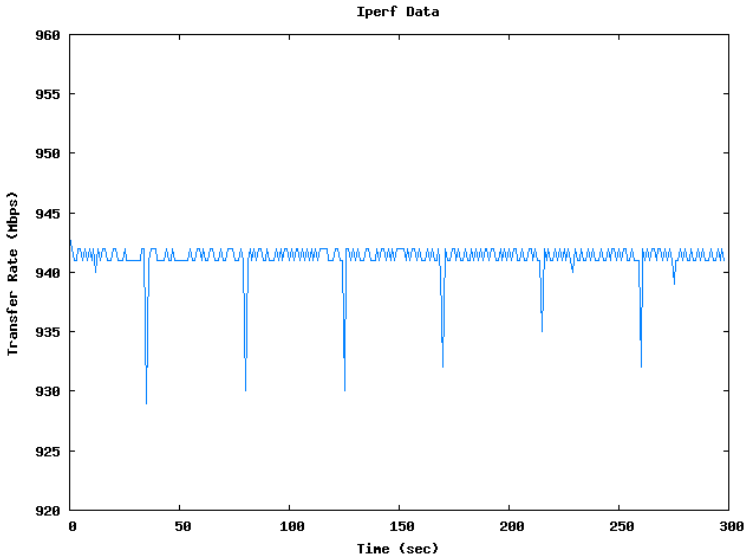


Figure 16: flamejet2-oiltank2 Parallel Test

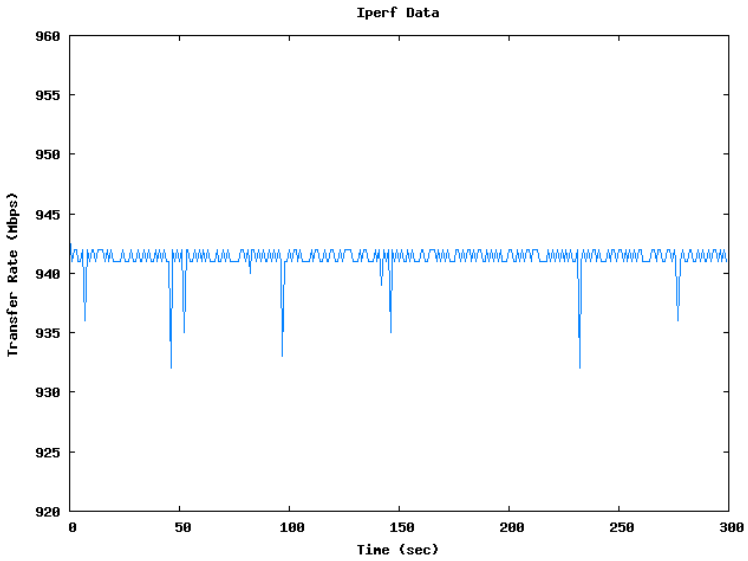


Figure 17: flamejet3-oiltank3 Sequential Test

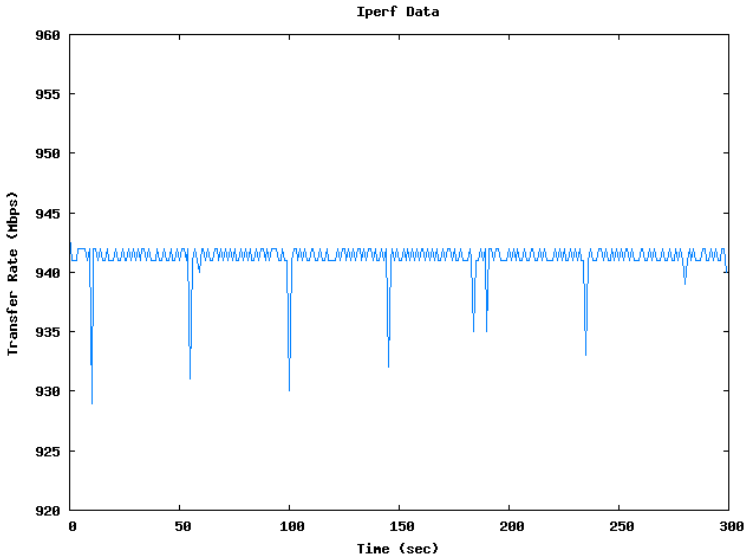


Figure 18: flamejet3-oiltank3 Parallel Test

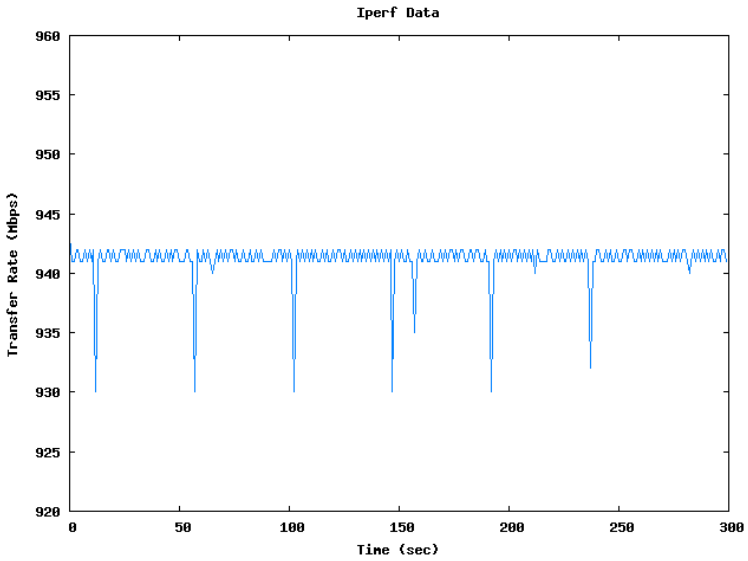


Figure 19: flamejet4-oiltank4 Sequential Test

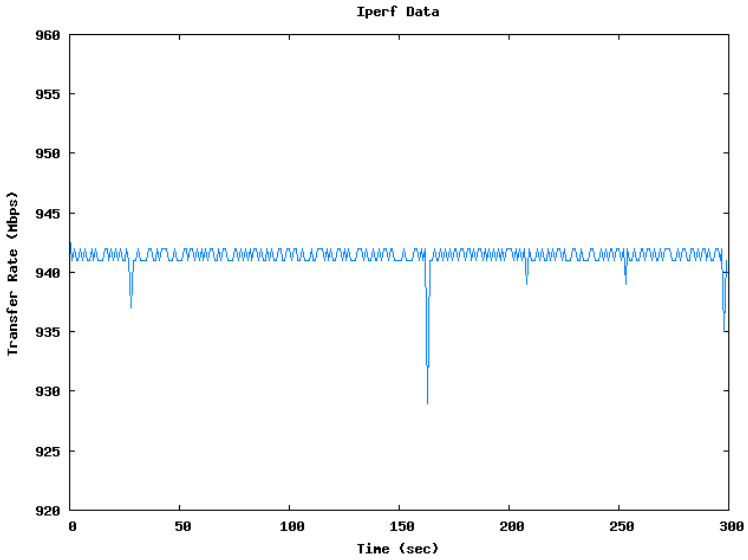


Figure 20: flamejet4-oiltank4 Parallel Test

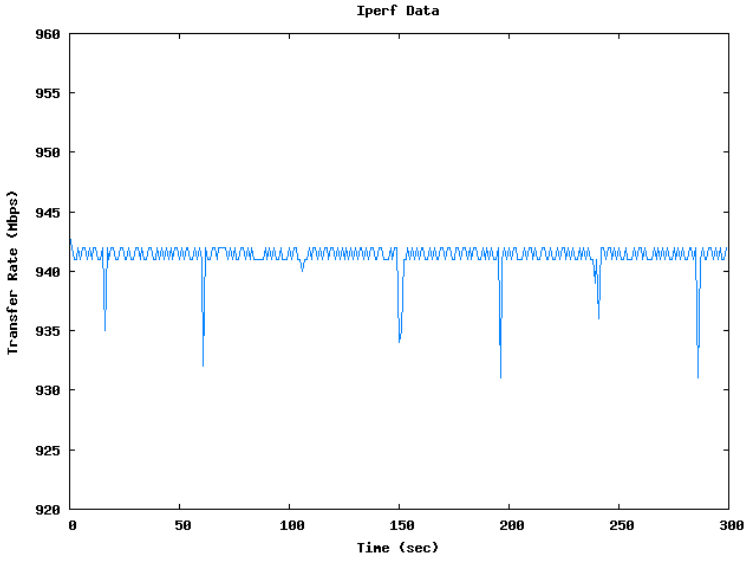


Figure 21: flamejet5-oiltank5 Sequential Test

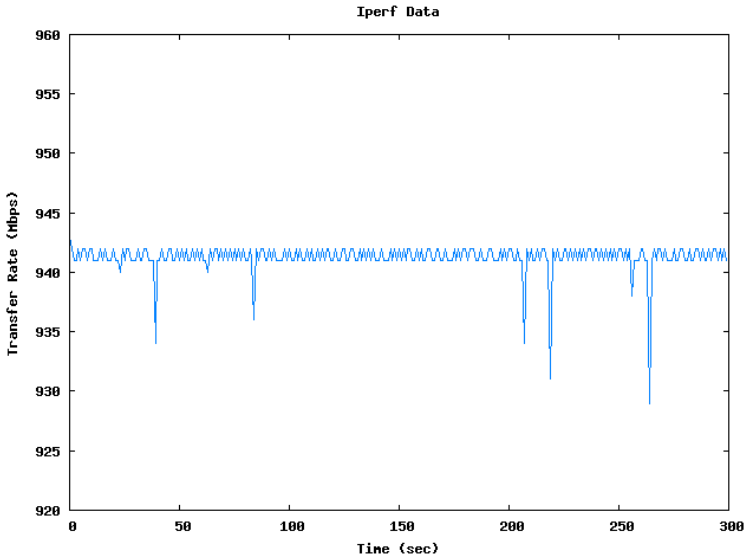


Figure 22: flamejet5-oiltank5 Parallel Test

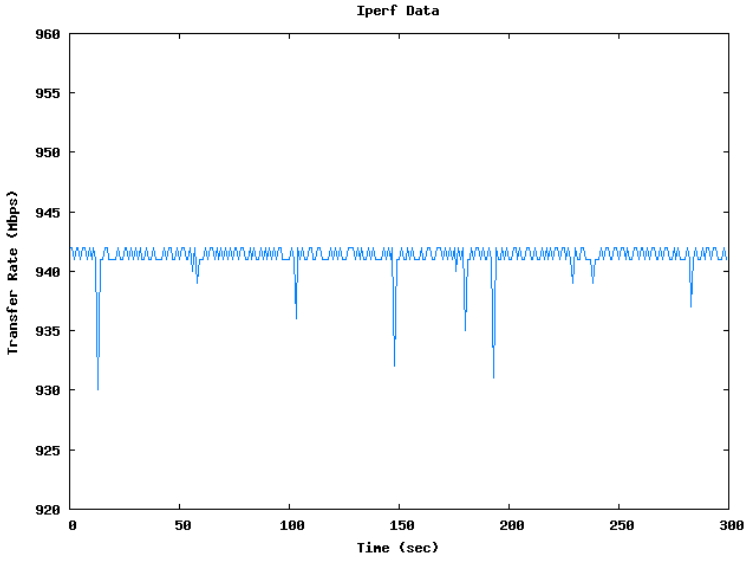


Figure 23: flamejet6-oiltank6 Sequential Test

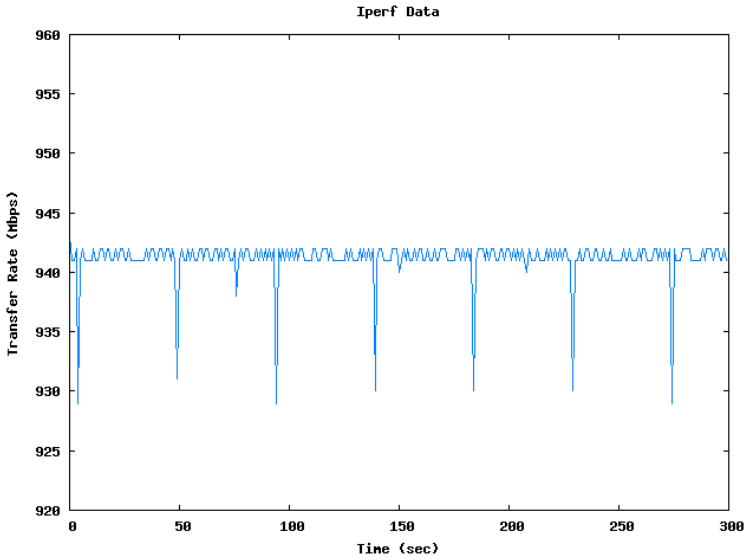


Figure 24: flamejet6-oiltank6 Parallel Test

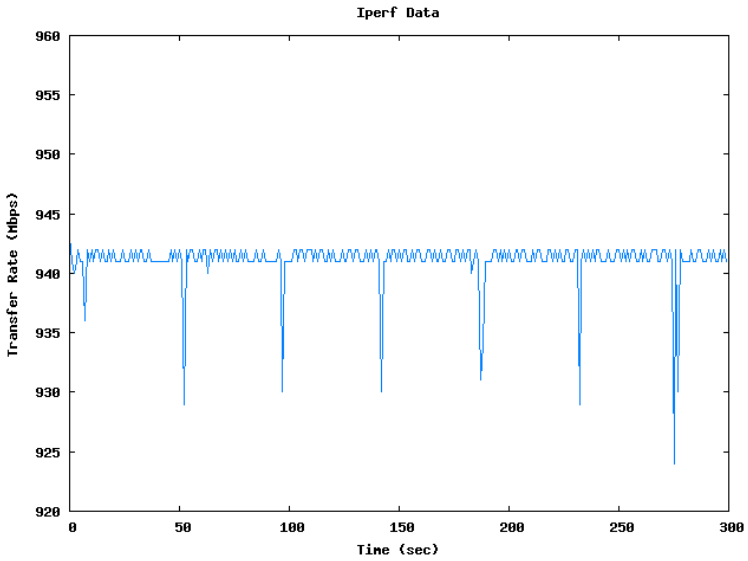


Figure 25: flamejet7-oiltank7 Sequential Test

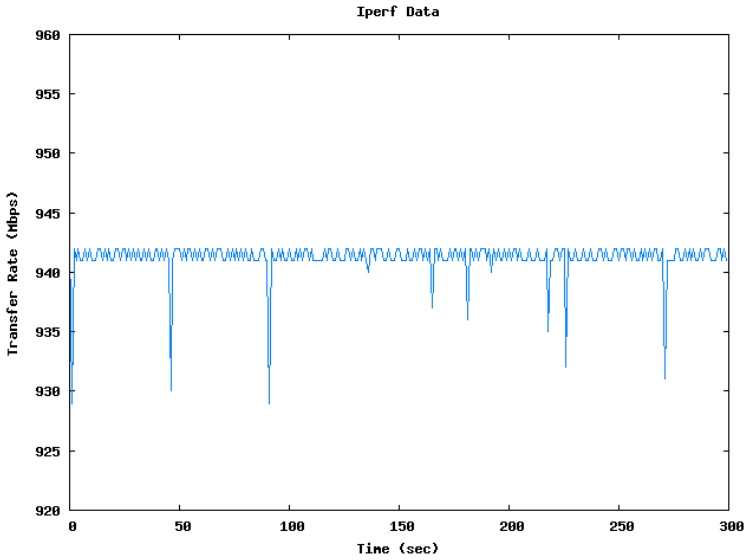


Figure 26: flamejet7-oiltank7 Parallel Test

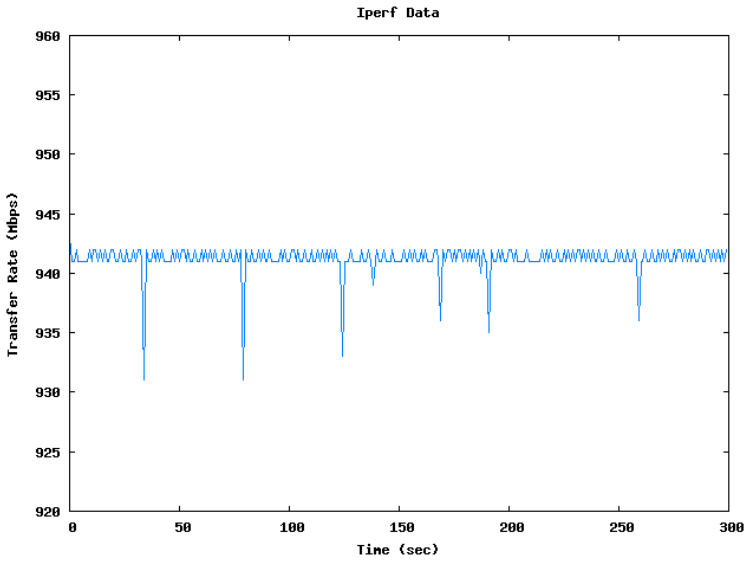


Figure 27: flamejet8-oiltank8 Sequential Test

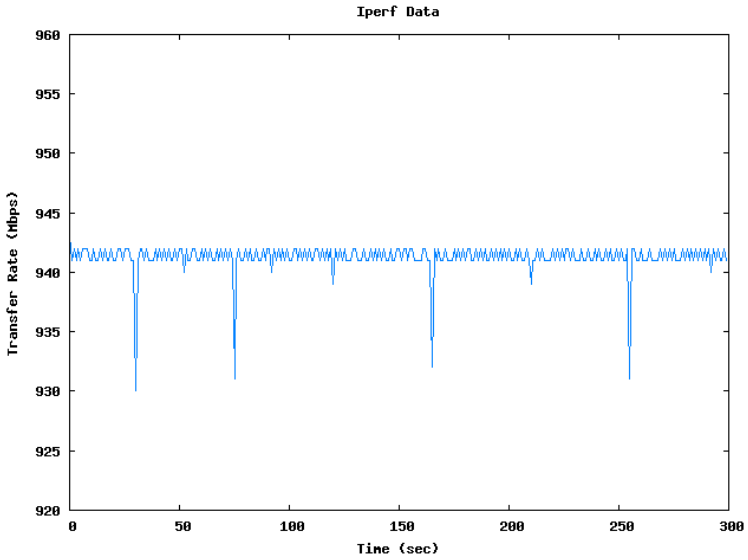


Figure 28: flamejet8-oiltank8 Parallel Test

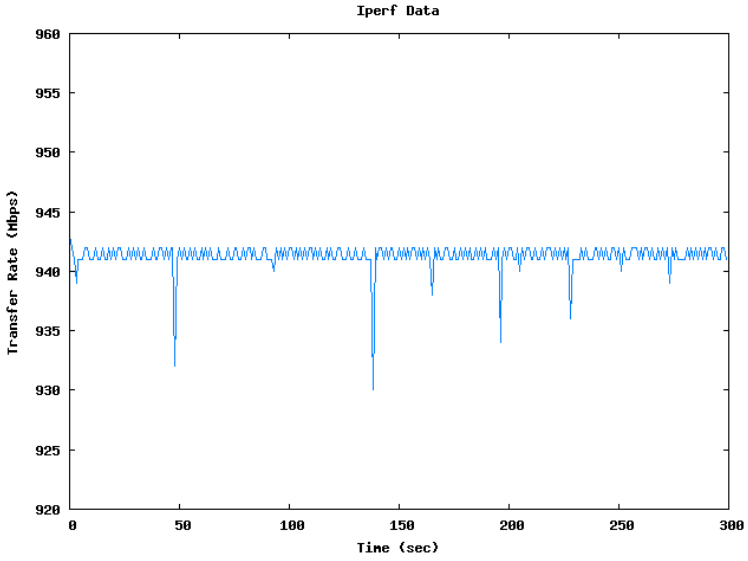


Figure 29: flamejet9-oiltank9 Sequential Test

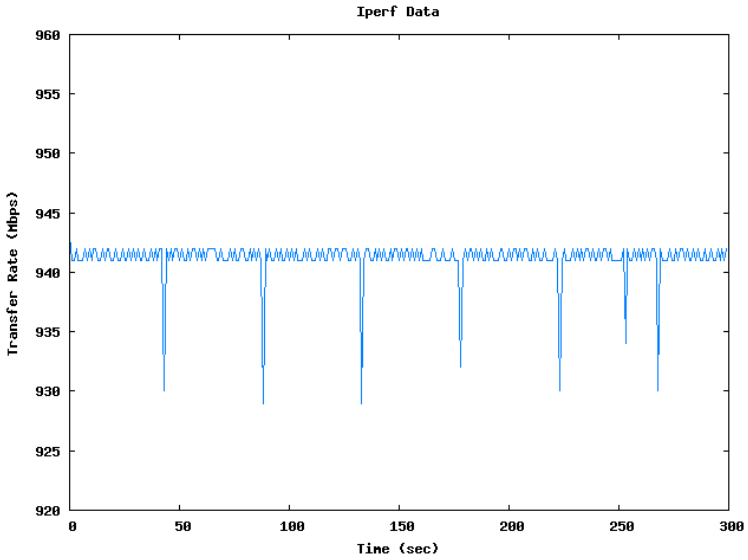


Figure 30: flamejet9-oiltank9 Parallel Test

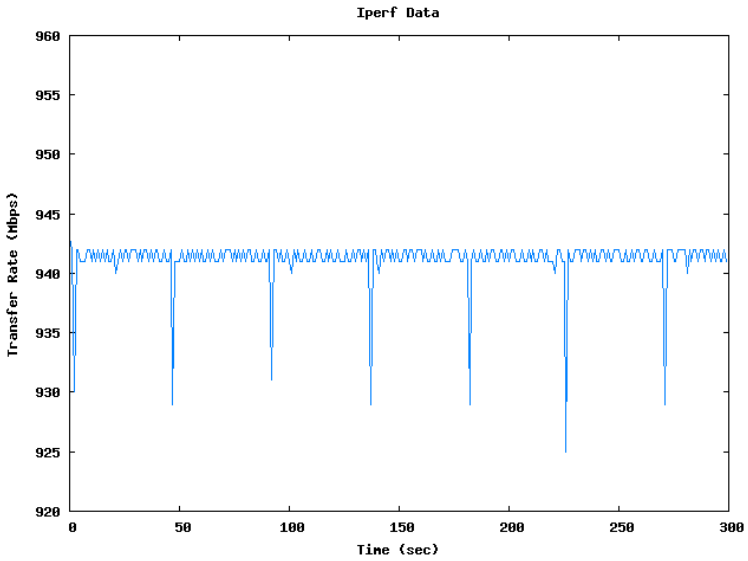


Figure 31: flamejet10-oiltank10 Sequential Test

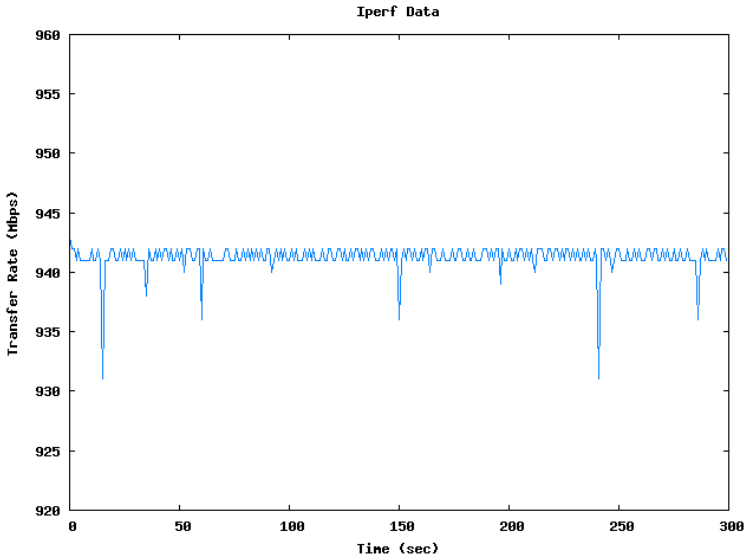


Figure 32: flamejet10-oiltank10 Parallel Test

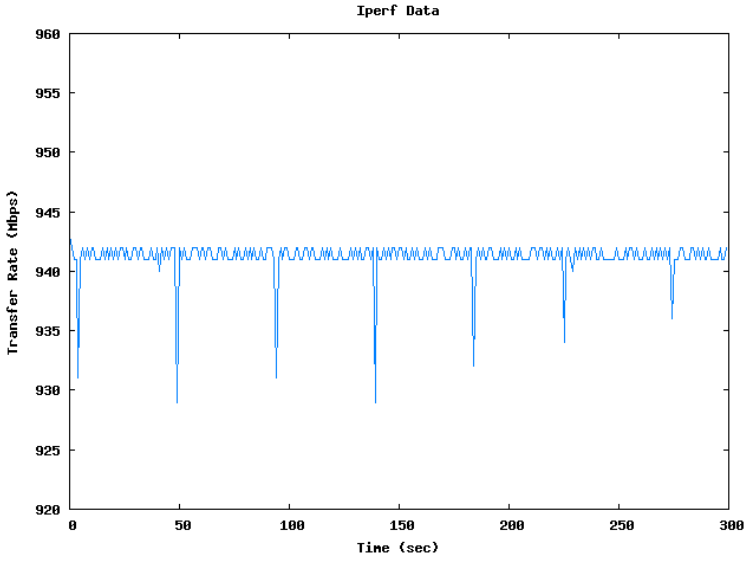


Figure 33: Simultaneous Iperf Test: Long-Running Client

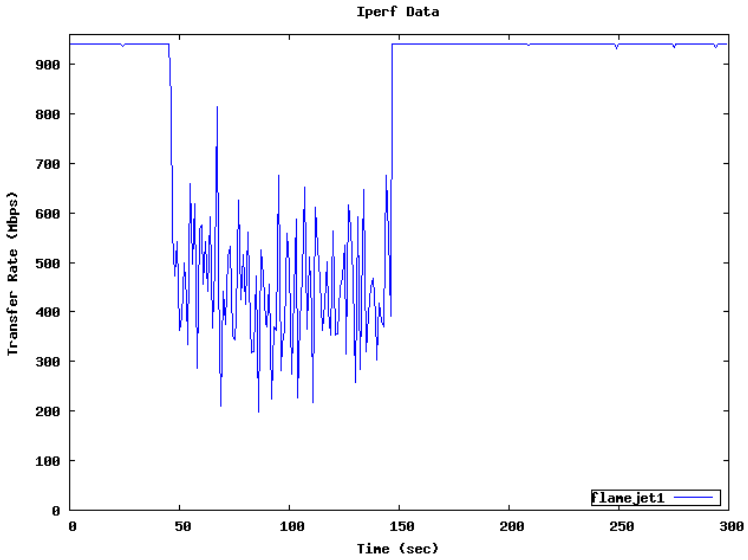


Figure 34: Simultaneous Iperf Test: Short-Running Client

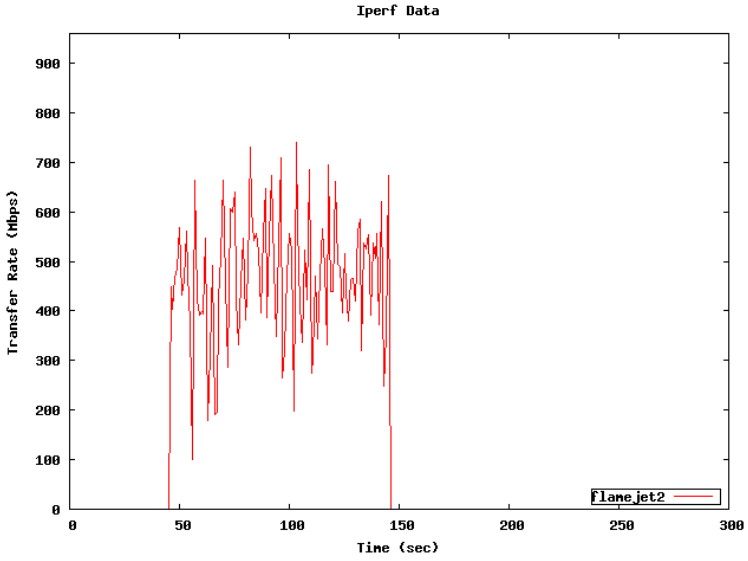


Figure 35: Simultaneous Iperf Test: Both Clients

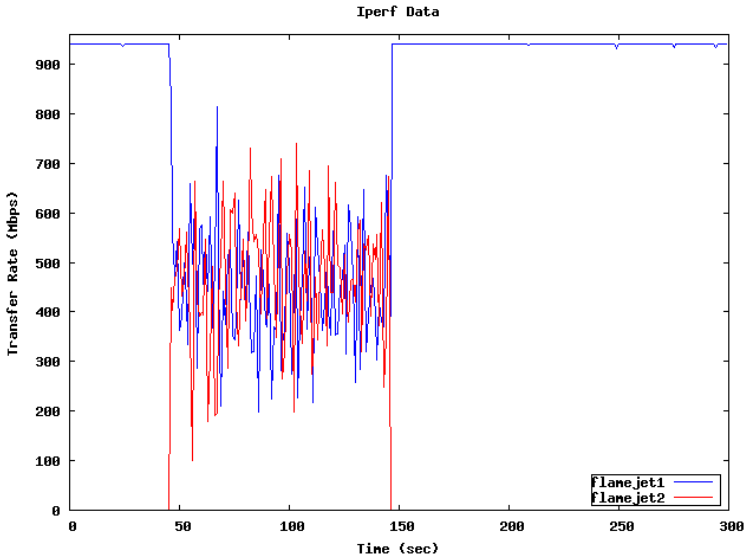


Figure 36: Detail of Simultaneous Iperf Test Client Overlap

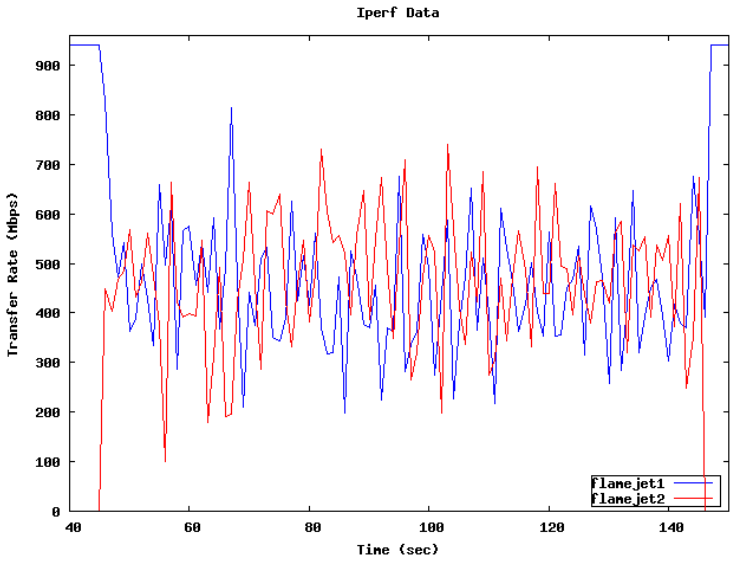


Figure 37: Iperf Ping Test: Iperf Results

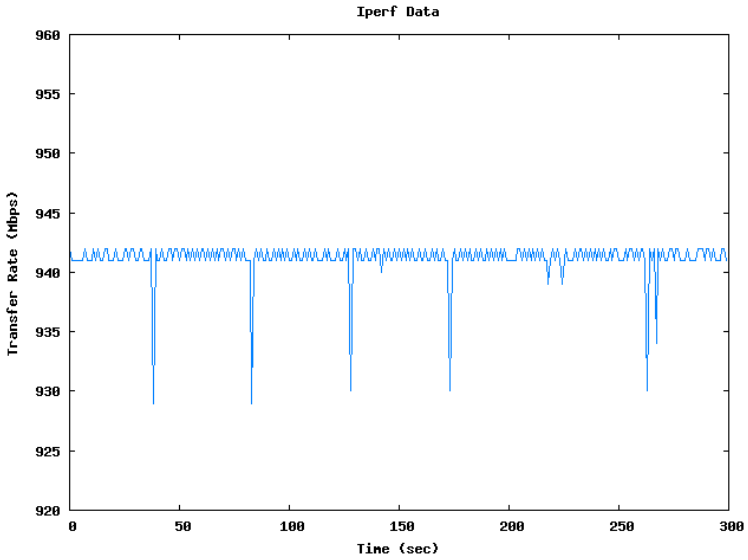


Figure 38: Iperf Ping Test: Ping Results

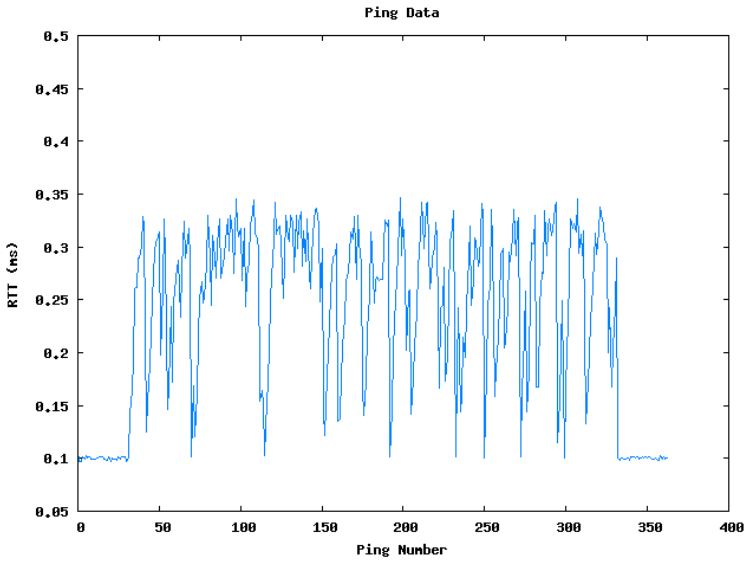
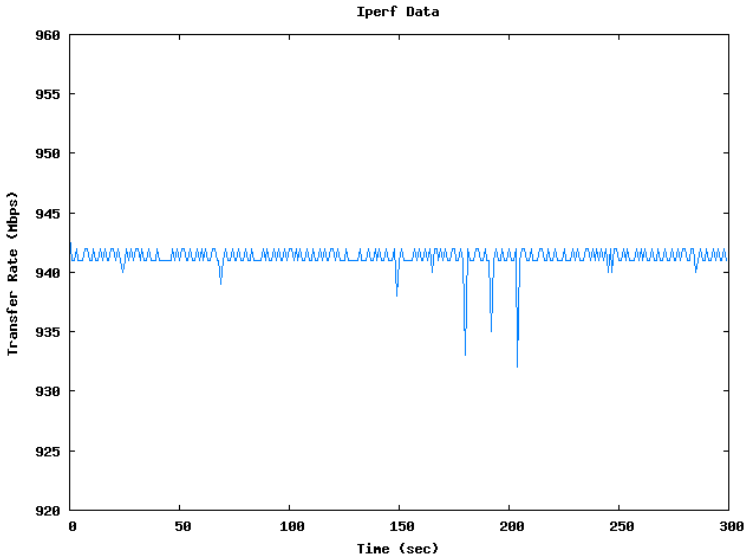


Figure 39: Traced Iperf Test



B.3 pathChirp Test Figures

The results of the *pathChirp* tests are visualized in figures 40 through 59. Each test utilized a connection between a client on a flamejet node and a server on the corresponding oiltank node. Tests were conducted twice for each client-server pair: once in a "sequential" test run (one test at a time), and once in a "parallel test run" (all client-server pairs at once, but isolated from each other by the switch).

To perform the tests, *pathChirp* sent out test "chirps" at various intervals and in bursts of varying sizes using UDP [16]. Thus, the measured instantaneous bandwidth varied over time, peaking around the maximum available bandwidth between the client and server. Each chirp test was run for five minutes.

Figure 40: flamejet1-oiltank1 Sequential Test

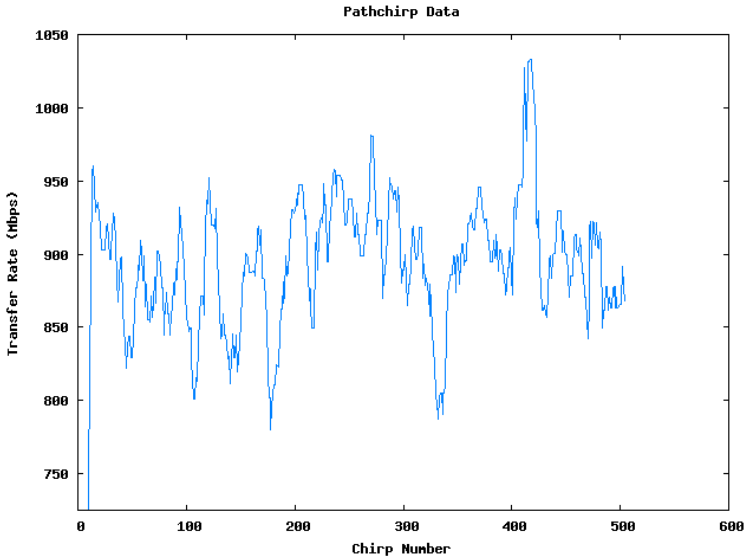


Figure 41: flamejet1-oiltank1 Parallel Test

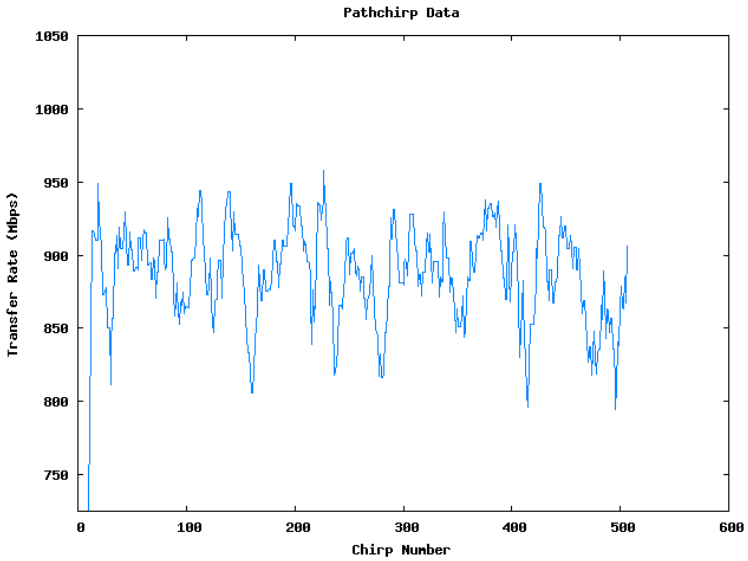


Figure 42: flamejet2-oiltank2 Sequential Test

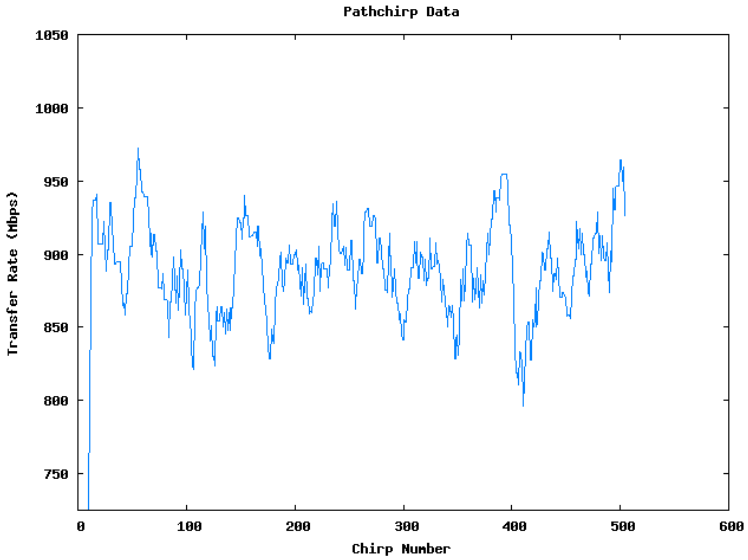


Figure 43: flamejet2-oiltank2 Parallel Test

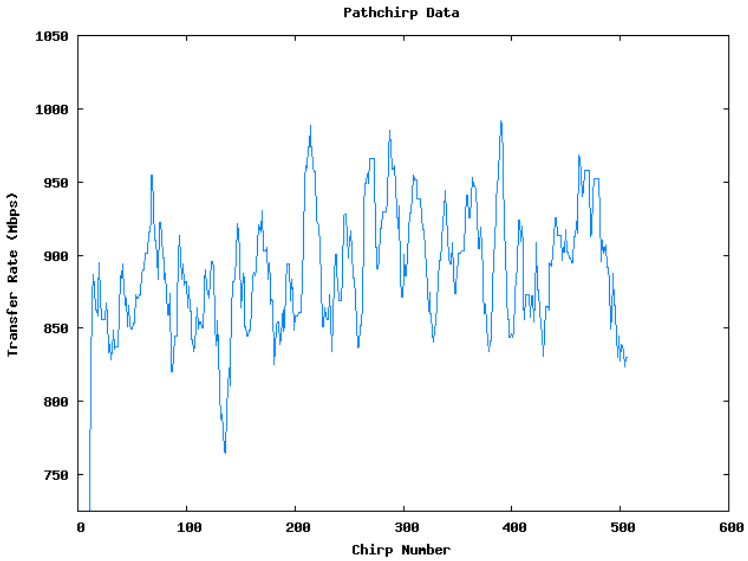


Figure 44: flamejet3-oiltank3 Sequential Test

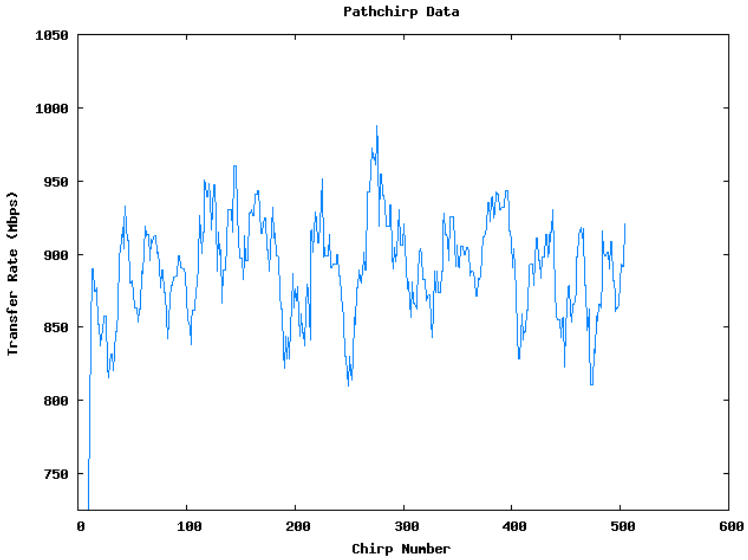


Figure 45: flamejet3-oiltank3 Parallel Test

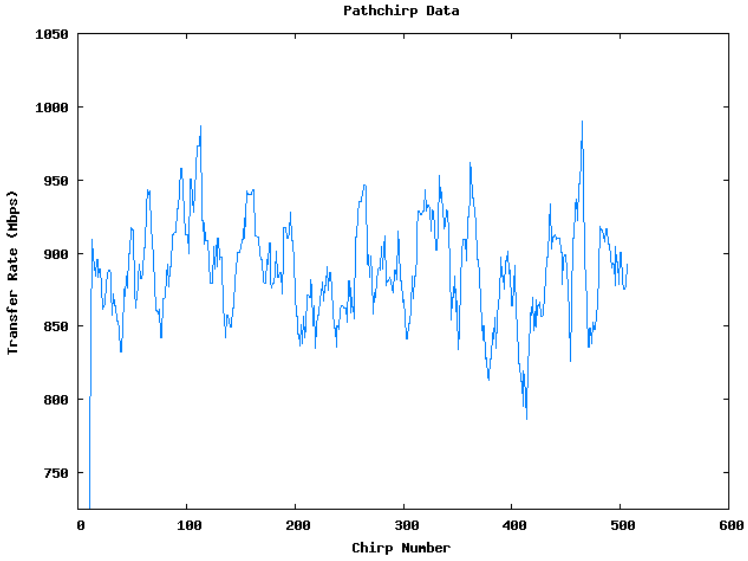


Figure 46: flamejet4-oiltank4 Sequential Test

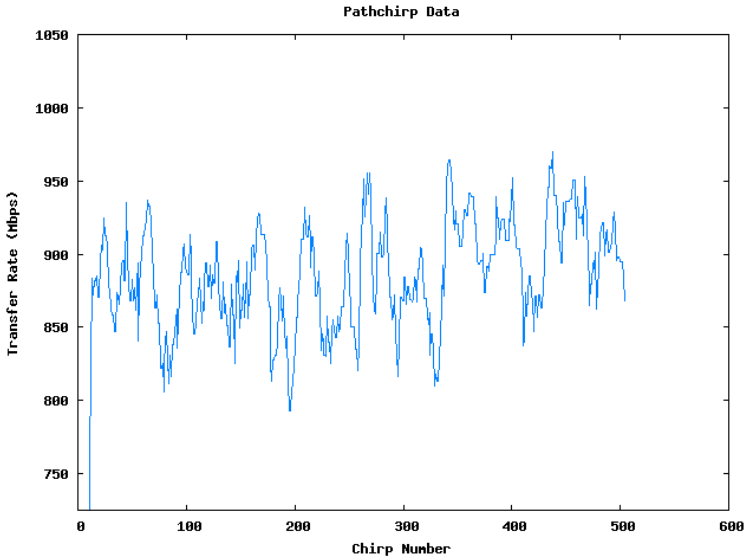


Figure 47: flamejet4-oiltank4 Parallel Test

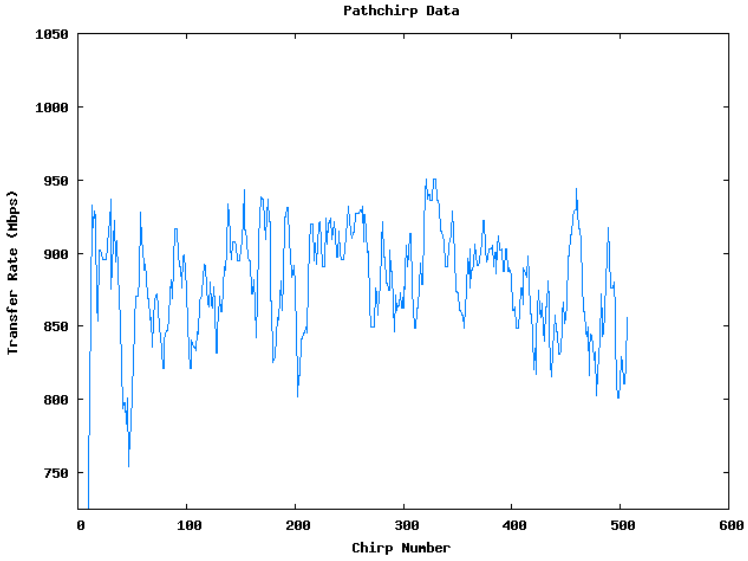


Figure 48: flamejet5-oiltank5 Sequential Test

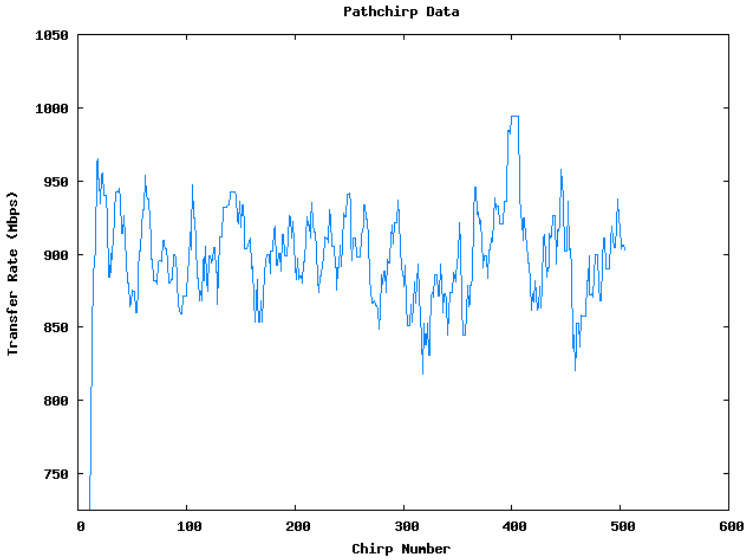


Figure 49: flamejet5-oiltank5 Parallel Test

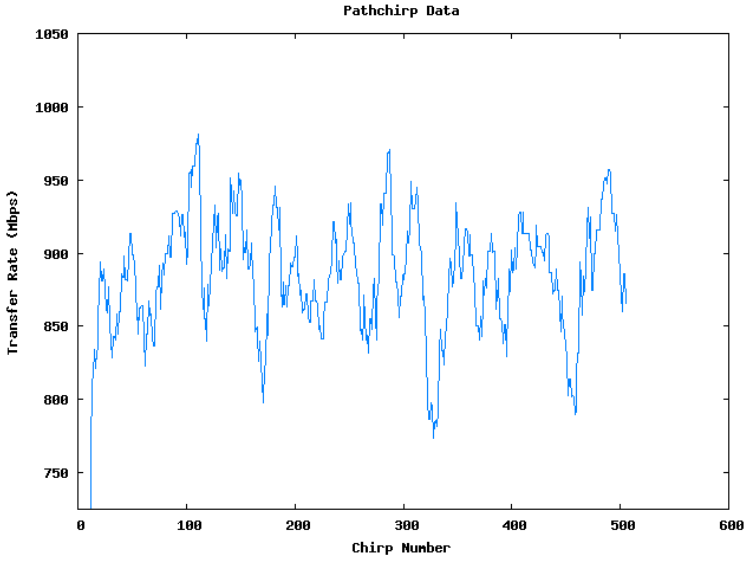


Figure 50: flamejet6-oiltank6 Sequential Test

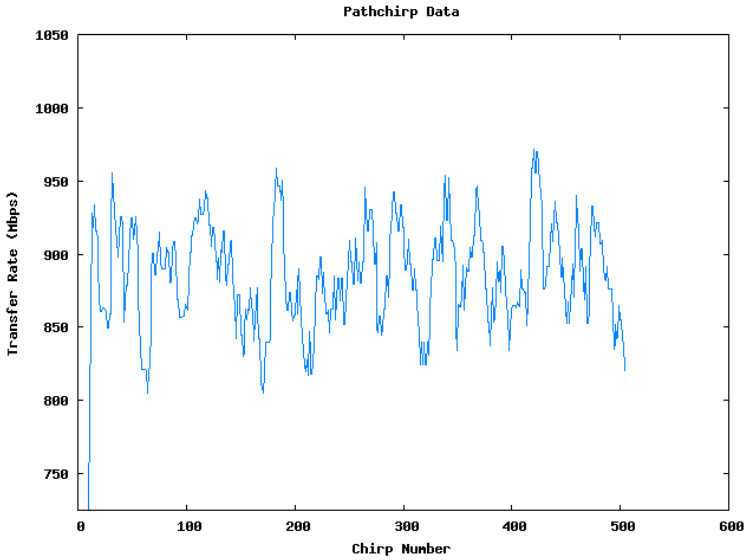


Figure 51: flamejet6-oiltank6 Parallel Test

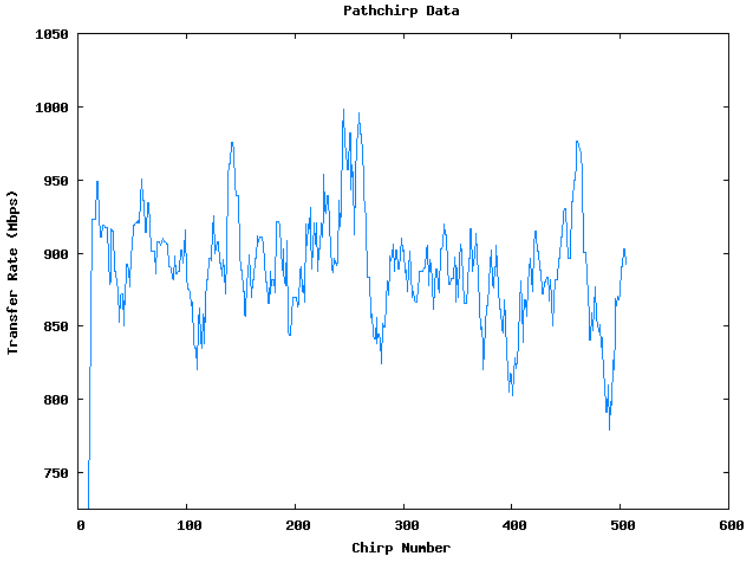


Figure 52: flamejet7-oiltank7 Sequential Test

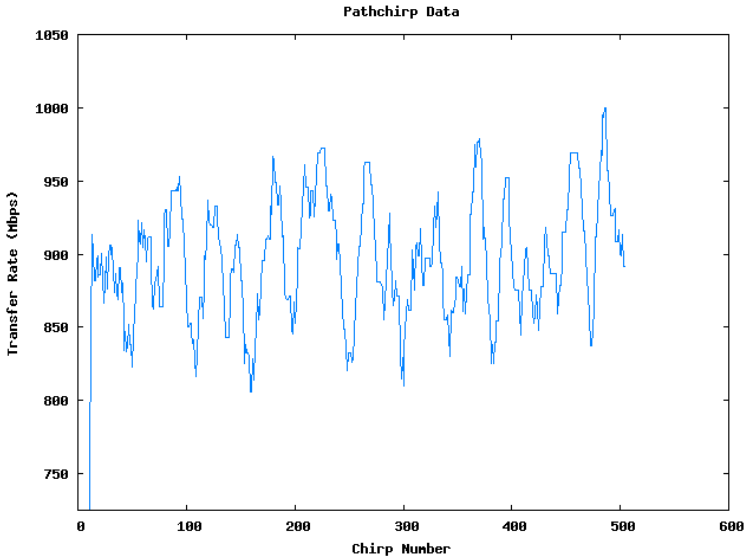


Figure 53: flamejet7-oiltank7 Parallel Test

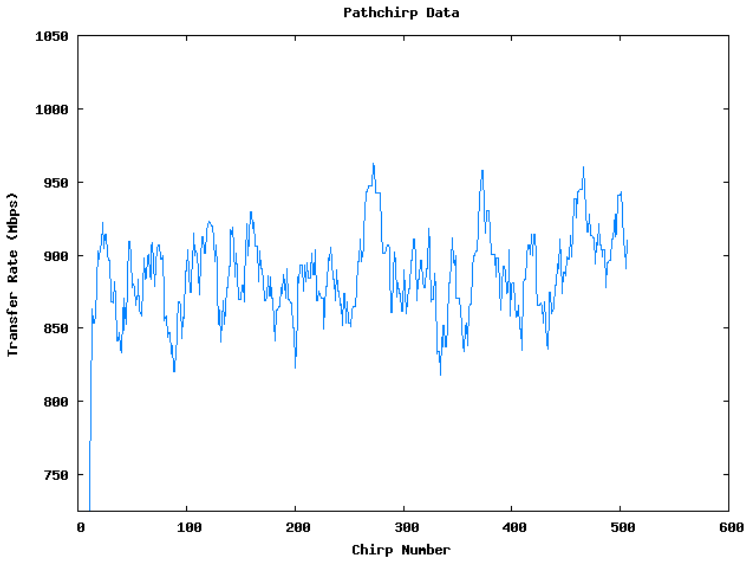


Figure 54: flamejet8-oiltank8 Sequential Test

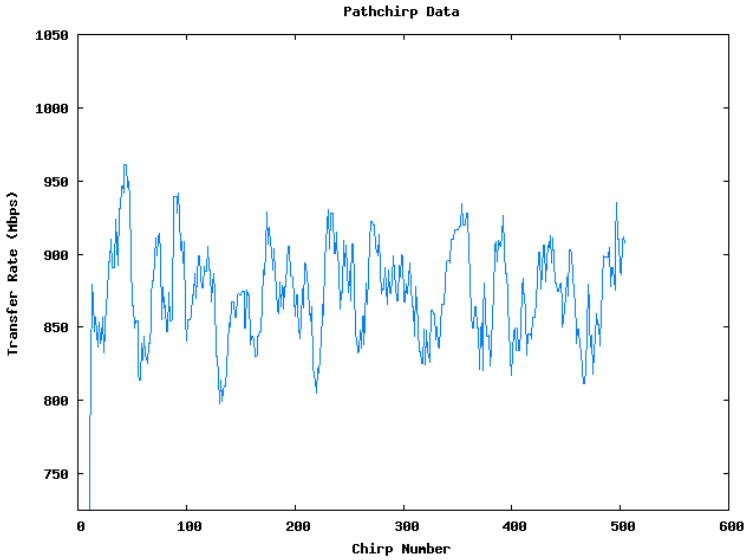


Figure 55: flamejet8-oiltank8 Parallel Test

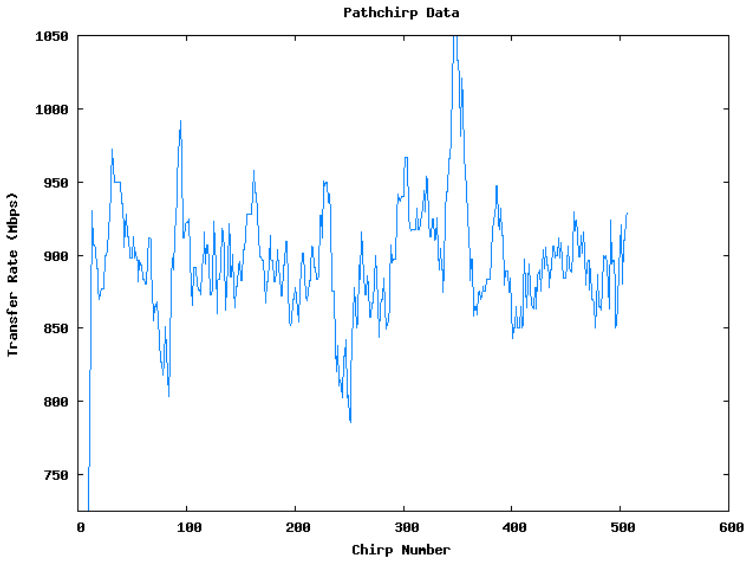


Figure 56: flamejet9-oiltank9 Sequential Test

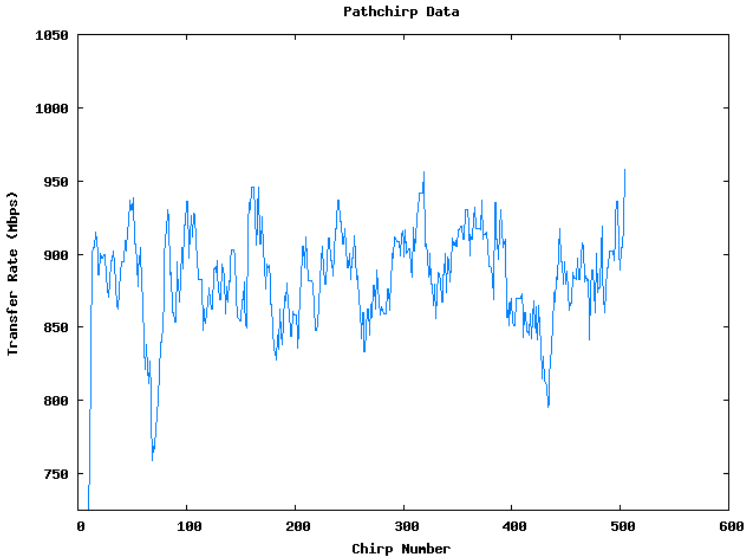


Figure 57: flamejet9-oiltank9 Parallel Test

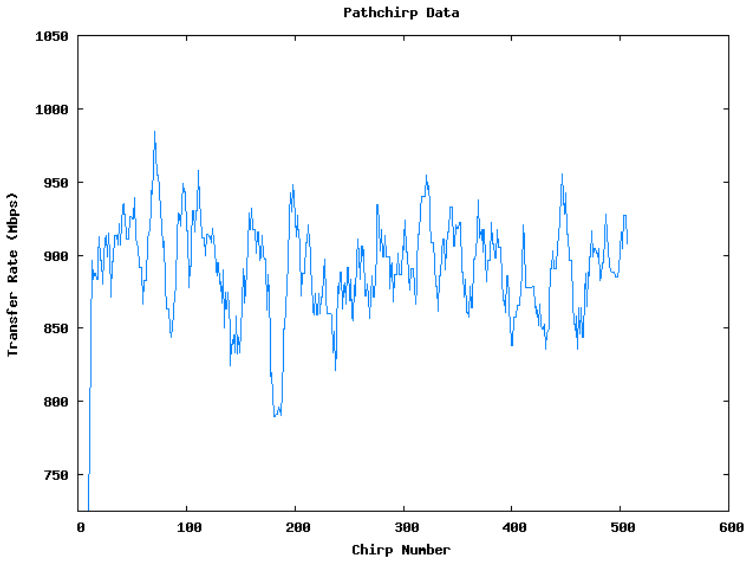


Figure 58: flamejet10-oiltank10 Sequential Test

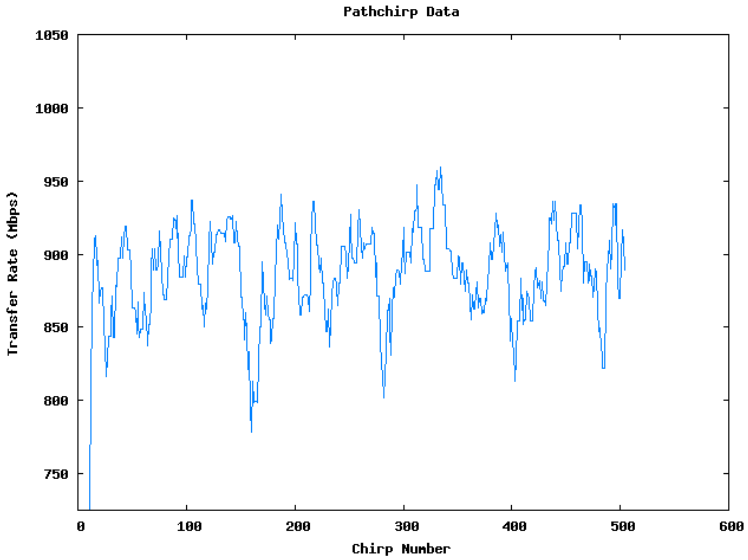
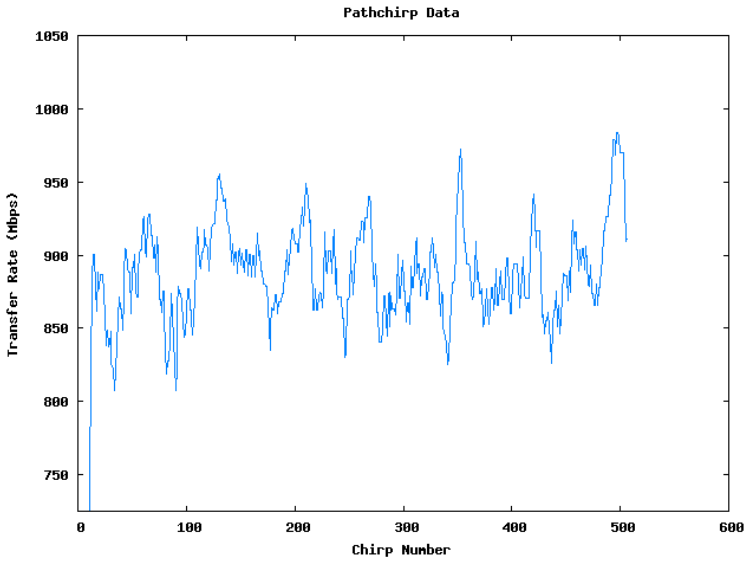


Figure 59: flamejet10-oiltank10 Parallel Test



C Scripts and Programs

This appendix is a verbatim copy of the `README` file included in the submitted source and data package.

```
README file for CpSc 852 Project, Fall 2007
Mike Murphy and Heather Harton

1. Package Contents

data/              Raw Iperf, pathChirp, and file transfer data

   single.X        Sequential Iperf test data
   mult.X          Parallel Iperf test data
   seq-in          Sequential inbound transfer times (order 1, 10, 2, 3, ...)
   seq-out         Sequential outbound transfer times (order 1, 10, 2, 3, ...)
   inm-time1.X     Inbound parallel transfer times, small file
   inm-time2.X     Inbound parallel transfer times, large file
   outm-time1.X    Outbound parallel transfer times, small file
   outm-time2.X    Outbound parallel transfer times, large file
   discrim         flamejet-flamejet and oiltank-oiltank transfer times

   flamejetX_oiltankX_T.instbw pathChirp data at time T
                                        later times are parallel, earlier seq

   'X' is an integer representing flamejet-oiltank pair number

ipdiscrim/         Iperf window/ping trace and pathChirp trace tests

   fjot1-pcdumped       pathChirp data for pathChirp UDP trace
   tcpdump.pc.gz        pathChirp UDP trace data (from tcpdump)

   iperf.wdowchk        Iperf data from tcpdump-traced Iperf test
   tcpdump.all.gz       tcpdump trace data for traced Iperf test

   iperf.pings          Iperf data from simultaneous Iperf/ping test
   oiltank1.pings       Ping data from simultaneous Iperf/ping test
   pings.step1          First-step post-processing for ping data
   proc_pings.py        Post-processor script for pings.step1

   *.dat                Data files suitable for plotting
   plot-*               GNUPlot scripts

iperf-XXYYZZ       Source code for Iperf

pathchirp-2.4.1/   pathChirp sources, binaries, and added scripts
```

```
   Bin/x86_64/pc_single_new      Sequential pathChirp execution script
   Bin/x86_64/pc_mult_new        Parallel pathChirp execution script

plot/              Plotting scripts and plot outputs (Iperf/pathChirp data)

   plot-all        Script to plot all data in the data/ directory
   plot-single     Dependency script to plot a single data set
   plot-iperf      GNUPlot script for Iperf plot data
   plot-pc         GNUPlot script for pathChirp plot data
   preprocess.py   Preprocessor for raw Iperf/pathChirp data to GNUPlot format

report/            LaTeX source for the report

   furnace-setup.odg     OpenOffice.org Draw graphic of furnace cluster setup
   phase1.bib            Bibliography file for report references
   report.tex            Report LaTeX source

simul/             Simultaneous (mult. cxns to same server) Iperf data

   simul.13        Raw data for flamejet1-oiltank3 connection (1st cxn)
   simul.23        Raw data for flamejet2-oiltank3 connection (2nd cxn)
   *.dat           Post-processed data suitable for GNUPlot
   plot-*          GNUPlot scripts
   proc_iperf.py   Post-processing script for Iperf data

stoker-0.30/       Sequential version of Stoker

   stoker.py       Main Stoker executable
   stoker.conf     Stoker configuration file
   furnace.scf     furnace cluster configuration file
   plugins/        Stoker plugins
   copy-in-par     Parallel inbound copy script
   copy-in-seq     Sequential inbound copy script
   copy-out-par    Parallel outbound copy script
   copy-out-seq    Sequential outbound copy script
   iperf-para      Parallel Iperf script
   iperf-seq       Sequential Iperf script
   (remainder)     Dependencies of Stoker or of above scripts

stoker-0.31/       Parallel version of Stoker

   stoker.py       Main Stoker executable
   stoker.conf     Stoker configuration file
   furnace.scf     furnace cluster configuration file
```

```
   plugins/        Stoker plugins
   iperf-install   Iperf installation script
   pathchirp-comp  pathChirp compilation script
   (remainder)     Dependencies of Stoker

ttimes/            Transfer time data, spreadsheet, and visualizations

   all-times       Processing script for converting time data for import
   times.py        Python processing script for a single time data file

   transfer_times.gnumeric      Gnumeric spreadsheet containing times/plots

   (remainder)     Raw time data/plots

cu-cilab-2007-1.pdf      PDF version of the report


2. Build Instructions

   2.1. Iperf

   In the Iperf source directory:

   ./configure
   make
   make install

   For installation on the cluster, the iperf-install script, in stoker-0.31/,
   was used:

   stoker-0.31/stoker.py all copy iperf*.tar.gz /root
   stoker-0.31/stoker.py all exec -s iperf-install

   2.2. pathChirp

   In the pathChirp source directory:

   ./configure
   make

   For compilation on the cluster, the pathchirp-comp script, in stoker-0.31/,
   was used:

   stoker-0.31/stoker.py all copy pathchirp-2.4.1.tar.gz /root
   stoker-0.31/stoker.py all exec -s pathchirp-comp
```

2.3. Scripts

There is no build procedure for included scripts, but recent versions of bash, GNUPlot, and Python are needed.

3. Executing Iperf

To execute an Iperf test, Iperf must be running on a remote server node with the command:

```
iperf -s
```

On the client node, run:

```
iperf -c <server> -f m -i 1 -t 300 > <data_file>
```

where <server> is the hostname or IP of the Iperf server node, and <data_file> is the desired output file for the results

For simplicity of execution across the cluster, the iperf-seq and iperf-para scripts in the stoker-0.30 directory were executed to initiate clients. Stoker 0.31 was used to initiate the servers, with the command:

```
stoker-0.31/stoker.py oiltanks exec iperf -s
```

Terminating the servers required CTRL+C to abort Stoker, followed by:

```
stoker-0.31/stoker.py oiltanks exec killall iperf
```

4. File Transfer Operations

The 1.5 and 20 GiB test files are NOT included in the submission package, owing to their extremely large sizes. Since the content of the files is irrelevant to the transfer tests, suitable data files can be created with the commands:

```
dd if=/dev/random of=small_file bs=1M count=1536
dd if=/dev/random of=large_file bs=1G count=20
```

Depending on the system, /dev/urandom might support faster reads than /dev/random.

The transfer tests themselves, using different filenames than the above examples, were scripted using the copy-in-seq, copy-in-par, copy-out-seq, and copy-out-par scripts in the stoker-0.30/ directory. If the file is already present on the destination system prior to the copy, it must first be deleted to ensure that the OS does not take shortcuts to avoid copying the entire file.

5. pathChirp Tests

Execution of pathChirp was facilitated via the parallel version of Stoker.

Senders were started using:

stoker-0.31/stoker.py flamejets exec pathchirp-2.4.1/Bin/x86_64/pathchirp_snd

Receivers were started using:

stoker-0.31/stoker.py oiltanks exec pathchirp-2.4.1/Bin/x86_64/pathchirp_rcv

pathChirp tests were executed by the scripts pc_single_new and pc_mult_new in the pathchirp-2.4.1/Bin/x86_64 directory (where output also is placed)

6. Miscellaneous Tools

The standard ping and tcpdump tools are already installed on the cluster nodes. The tcptrace tool was installed on a workstation for use in post-experiment analysis, using the Synaptic front-end to the Ubuntu package manager.

7. Post-Processing, Plotting, and Reporting

GNUPlot was already installed on the analysis workstation. The 'plot-all' script in the plot/ directory, along with its dependent scripts, were utilized to automate post-processing and plotting of the Iperf and pathChirp data (which were copied into the data/ directory first). A similar script was used in the ipdiscrim/ directory to post-process the ping data into a form suitable for GNUPlot. Necessary GNUPlot scripts are included to make the process fully automatic.

In the ttimes/ directory is a post-processing script for the raw transfer time data (most of which was collected manually). This 'all-times' script will produce text files suitable for importing into the Gnumeric spreadsheet.

This import operation has been done, and the included transfer_times.gnumeric file contains the imported data and plots. Each plot was also manually exported to PNG format for inclusion in the final report.

For the simultaneous transfer data in simul/, the post-processing and plotting were performed manually, using the proc_iperf.py post-processing script and included GNUPlot scripts.

All resulting .png files from all operations were copied into the report/ directory, where LaTeX was used for the report. It should be noted that these PNG graphics files are suitable for pdflatex, but EPS conversions may be needed to run the regular latex program.

D References

[1] Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, Jos Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids: the in-vigo system. *Future Generation Computer Systems*, 21(6): 896–909, June 2005.

[2] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 273–286, Boston, MA, May 2005.

[3] Dell Inc. *Dell Powerconnect 6200 Series Switches*, 2007.

[4] Wesley Emeneker and Dan Stanzione. Dynamic virtual clustering. In *IEEE Cluster 2007*, Austin, TX, September 2007.

[5] Renato J. Figueiredo, Peter A. Dinda, and Jos A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[6] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 15(3):200–222, 2001.

[7] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the internet. In *Second International Workshop on Virtualization Technology in Distributed Computing*, Reno, NV, November 2007.

[8] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, Stanford, CA, August 1988.

[9] Piotr Luszczek, David Bailey, Jack Dongarra, Jeremy Kepner, Robert Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The hpc challenge (hpcc) benchmark suite. In *SC06 Conference Tutorial*, Tampa, FL, November 2006.

[10] Hideo Nishimura, Naoya Maruyama, and Satoshi Matsuoka. Virtual clusters on the fly - fast, scalable, and flexible installation. In *CCGRID 2007: Seventh IEEE International Symposium on Cluster Computing and the Grid*, May 2007.

[11] Shawn Ostermann. *tcptrace - Official Homepage*, 2003. URL `http://jarok.cs.ohiou.edu/software/tcptrace/`.

[12] Kihong Park, Gitae Kim, and Mark Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Fourth International Conference on Network Protocols*, 1996.

[13] Vern Paxson. Automated packet trace analysis of tcp implementations. In *ACM SIGCOMM '97*, Cannes, France, 1997.

[14] Vern Paxson. End-to-end internet packet dynamics. In *ACM SIGCOMM '97*, 1997.

[15] Lili Qiu, Yin Zhang, and Srinivasan Keshav. On individual and aggregate tcp performance. In *Seventh Annual International Conference on Network Protocols*, 1999.

[16] Vinay Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop*, 2003.

[17] Valerie Rybinski. De-mystifying category 5, 5e, 6, and 7 performance specifications. Technical report, Siemon Singapore, 1999. URL `http://www.siemon.com/sg/white_papers/99-12-17-demystifying.asp`.

[18] Seagate Technology LLC. *Data Sheet: Barracuda ES*, 2007. URL `http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es.pdf`.

[19] Stephen C. Simms, Gregory G. Pike, and Doug Balog. Wide area filesystem performance using lustre on the teragrid. In *TeraGrid 2007 Conference*, Madison, WI, June 2007.

[20] Ajay Tirumala, Les Cottrell, and Tom Dunigan. Measuring end-to-end bandwidth with iperf using web100. In *Passive and Active Monitoring Workshop*, 2003.

[21] Various Contributors. *Gnumeric - The Gnome Office Spreadsheet*, 2007. URL `http://www.gnome.org/projects/gnumeric/`.

[22] Various Contributors. *gnuplot homepage*, 2007. URL `http://www.gnuplot.info/`.

[23] Various Contributors. *LaTeX A document preparation system*, 2007. URL `http://www.latex-project.org/`.

[24] Various Contributors. *TCPDUMP/LIBPCAP public repository*, 2007. URL `http://www.tcpdump.org/`.

[25] Weikuan Yu, R. Noronha, and Shuang Liang D. K. Panda. Benefits of high speed interconnects to cluster file systems: a case study with lustre. In *20th International Symposium on Parallel and Distributed Processing*, 2006.