Extending Cluster File Systems beyond Last Miles

Hann-Huei Chiou, Pan-Lung Tsai, Jiunn-Jye Lee, Hsing-Fu Tung, Chun-Ying Huang, and Chin-Laung Lei
Department of Electrical Engineering, National Taiwan University
{*koala, charles, jye, molisado, huangant*}*@fractal.ee.ntu.edu.tw, lei@cc.ee.ntu.edu.tw*

***Abstract*— The growth of the bandwidth available in WAN links stimulates novel usage of traditional network systems. By extending the boundary of cluster file systems to the customers' premise, it is now possible to provide home users efficient, dependable, and responsive network storage. In this paper, we identify the primary issue of network latency when implementing cluster file systems across the last miles, and propose a solution by replacing the round-based data transmission protocol of Coda file system with a rate-based one. The performance evaluation of the prototype system shows significant improvements of both throughput and response time for file-transfer operations, especially under high network latency. The result is a latency-resistant cluster file system.**

Keywords: Cluster File System, Last Mile, Network Latency, Coda, UDT.

I. Introduction

The advancement of computer and communication networks has made data transmission reach the speed of multiple gigabits per second, and next-generation optical technologies together with the breakthrough of modern routing and switching techniques are going to further increase the transmission bandwidth to provide even faster links in the backbone. Similar elevation of link speed also takes place in the so-called last miles. Although there is still a gap between the data rate on the edge and in the backbone of the Internet, emerging services like FTTH (Fiber to the Home) and FTTB (Fiber to the Building) usually provide more than sufficient bandwidth and hence enable various new applications. A commonly referred example is the construction of computational grids [1], within which multiple nodes are interconnected via high-speed links and accomplish computation-intensive tasks in a collaborative manner.

This kind of revolutionary change also encourages novel usage of traditional network systems. One of the killer applications will come from the unleashed power of cluster file systems: since the last miles are no longer highly bandwidth constrained, it becomes a good idea to extend the boundary of a cluster file system to the customers' premise so that it can be used to provide home users efficient, dependable, and responsive network storage.

Migrating existing cluster file systems to the coming network paradigm places several challenges in the protocol design. In particular, as bandwidth constraints vanish, the inherent transmission latency of last-mile WAN (Wide Area Network) links becomes the primary issue. In this paper, we investigate the implications and consequences of network latency, and provide quantitative evaluation of the impact on a cluster file system. We also propose an adequate solution and present a prototype implementation based on Coda file system.

The rest of the paper is organized as follows. Section II gives a brief introduction to Coda file system and a detailed examination of the corresponding data transmission protocol, SFTP (Simple File Transmission Protocol). Section III considers the issue of network latency and evaluates the impact on the performance of Coda file system, followed by the explanation of the proposed solution in section IV. Section V then concludes the paper by summarizing our major achievements.

†This work was supported in part by the National Science Council of the Republic of China under grant NSC 94-2218-E-002-038.

II. Coda File System

Various cluster file systems have been proposed in the last decades, and Coda file system is one of the most popular. The well-known Linux operating system even includes Coda file system as part of its standard implementation [2]. Therefore, Coda file system becomes a reasonable choice as a representative candidate for the evaluation of cluster file systems.

The Coda file system, which is based on AFS2 (Andrew File System v2) [3], is developed since 1987 by the group led by Prof. M. Satyanarayanan at Carnegie Mellon University. To achieve the goals of constant data availability and scalability, Coda incorporates several features which are not found in other systems, such as client-side persistent cache, write-back caching, network bandwidth adaption, disconnected operation for mobile clients, failure resilience, Kerberos-like authentication, ACLs (Access Control Lists), well-defined sharing semantics, and so on [4]–[6].

Since our goal is to improve the performance of file transfer in Coda, here we will focus on its client-server architecture and the file transfer protocol which it uses. Further discussions on Coda are left in the literature.

A. Client and Server Architecture

The Coda server, Vice, is a user-level process using existing kernel services. Therefore, the Coda server can be run on unmodified kernels. However, to run the Coda client, like other file systems, a computer needs kernel support to access Coda files. Some Coda-specific changes are required to be made to the kernel, which add Coda-specific definitions to the vnode layer in the kernel. This is necessary for satisfying the requirements of the VFS (Virtual File System) layer in the kernel, that is, Coda provides a VFS interface which translates user-generated system calls into file system requests. As a

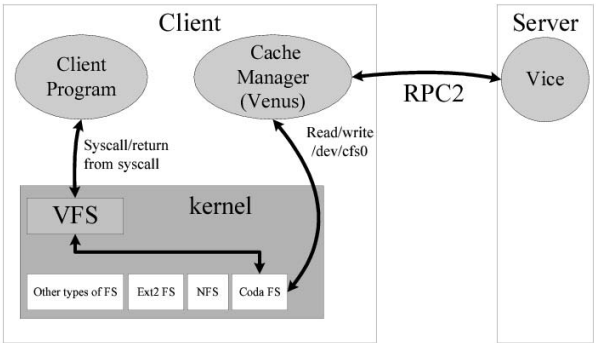
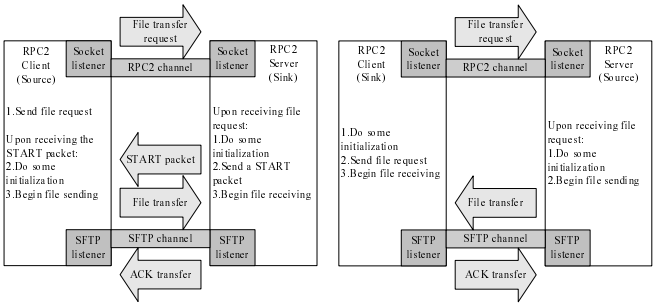


Fig. 1. The Coda client/server architecture.



(a) The source is an RPC2 client. (b) The source is an RPC2 server.

Fig. 2. The RPC2-SFTP architecture.

result, the Coda client consists of two components: the kernel module and the user space cache manager, Venus. The former functions as described above, while the latter is responsible for maintaining the local cache in the client side and contacting with the servers upon cache misses.

The interaction among Coda kernel module, Venus, and Vice is shown in Fig. 1. The communication between Venus and the Coda kernel module occurs by allowing Venus to read the request from the character device /dev/cfs0, which provides the kernel with access to Venus for handling the VFS vnode kernel interface. Venus and Vice communicate via RPC2 (Remote Procedure Call v2) [7], [8], which is in turn associated with a specialized protocol called SFTP [9] to perform efficient file transfers.

B. SFTP Protocol

SFTP is a specialized protocol used as a *side effect* associated with an RPC2 connection. The side effects can be viewed as function calls with special purposes in a RPC2 connection, where application-specific network optimizations can be performed. SFTP provides efficient file transfers either from an RPC2 server to an RPC2 client or vice versa.

Initially, an RPC2 channel is established via UDP sockets with a SocketListener at each end to monitor the channel. The RPC2 server and client use this channel to process the initialization steps of file transfer. When the initialization is complete, another pair of sockets comprises an SFTP channel, which is monitored by a pair of SFTPListeners and takes over the responsibility of actual file transfer.

As shown in Fig. 2(a), when the source is an RPC2 client, it first sends a file transfer request to the RPC2 server to inform it of the file transfer. The server then processes some initializations, preparing for the file transfer. When it is ready, the server return a START packet, notifying the client to start the file transfer.

On the contrary, if the source is an RPC2 server, after receiving the file transfer request from the client, it processes some initializations and then starts file transfer without a START packet as shown in 2(b).

It is noticeable that no matter whom the source is, it is the RPC2 client that initiates the file transfer. Although it may seem a little confusing at first glance, just recall that the RPC2 client and server correspond to the Coda client and server respectively. Therefore, Fig. 2(a) can be thought of the case where a Coda client wishes to copy a file from the local file system to the Coda file system, while Fig. 2(b) can be thought of the case where a Coda client wishes to copy a file from the Coda file system to the local file system.

SFTP is a round-based protocol, that is, an SFTP file transfer is basically a cyclic exchange of data and ACK (Acknowledgment) packets. In each round, the source (sender) sends a block of data, waits until the ACK is returned from the sink (receiver), and enters the next round. The ACK contains information about the sequence numbers of the packets that has been received by the sink. If there are any lost packet, the source first retransmit data packets in the lost list, followed by next packets in the sending queue.

A timeout interval is used to detect packet losses, and timeout events are always monitored at the RPC2 server side. In the case where the source is the RPC2 server, occurring of a timeout means that all data packets transmitted in this round are lost and it is necessary to put them in the lost list. Conversely, if the source is the RPC2 client, occurring of a timeout means that the ACK is lost, and the sink (server) has to retransmit ACK.

SFTP adopts a simple window-based congestion control mechanism that the total number of packets which have been transmitted and not acknowledged can not exceed the maximum size of the transmission window.

Finally, to tune the performance of file transfer, Coda adjust the retransmission timeout interval according to the RTT (Round-Trip Time) of the network link. Initially, the retransmission interval is set to 2 seconds, but varies depending on RTT observations collected during file transfers. When a timeout occurs, the retransmission interval is backed off. SFTP collects RTT observations by packet timestamps.

III. The Impact of Network Latency

As the available bandwidth increases, the primary issue of the last mile now goes to the high network latency. In a high BDP (Bandwidth-Delay Product) environment, traditional protocols, such as TCP and SFTP, performs poorly due to its

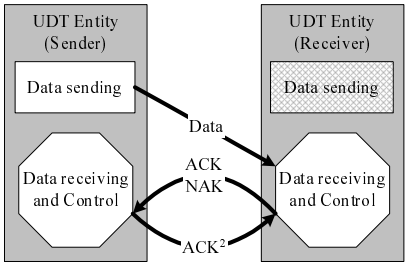


Fig. 3. The UDT sender/receiver architecture.

window-based congestion control algorithm. The link utilizations are low because of the slowly increased window size. And the network latency in each round will be "accumulated" thus further downgrade the overall performance.

Coda tries to overcome this issue by introducing the *Disconnection Operation*, which allows a client to sustain the access to files in the local cache even when it cannot contact any server (the effective bandwidth is below 50KB/s [6]). When the server becomes available again, Coda will automatically integrate the local modifications to the server. However, this can be quite inconvenient to users because he/she cannot access those absent from local caches.

There are many researches regarding this latency issue. Some of them requires changing to the existing protocols, such as HS-TCP (High-Speed TCP) [10]. Other approach, such as XCP (eXplicit Control Protocol) [11], defines a new network control protocol. Most of them cannot easily be adopted into the existing infrastructure.

And then third category, namely *Rate-based Control Protocols*, attracts our attention. They can be deployed at the application level without modifications to existing network infrastructure and can achieve high bandwidth utilization. As shown in [12], In an experiment with three SABUL connections running on a channel with 10Gbps link capacity and 110ms RTT, an amazing throughput of 2.8Gbps is reached, corresponding to 933Mbps per SABUL connection. And the rate-based control mechanism provides a smooth sending pattern and is more effective on high-BDP networks.

From the discussion above, UDT/SABUL adopts dynamic rate-based control mechanism and is proved to have superior performance. Therefore, it is chosen as the candidate of the file transfer protocol in our Coda derivative.

A. UDT/SABUL Protocol

UDT (UDP-based Data Transfer protocol) [13], [14] is an end-to-end unicast transport protocol built on top of UDP. SABUL (Simple Available Bandwidth Utilization Library) [12], [15] is an earlier implementation of UDT using two logical connections for each unidirectional transmission.

In UDT, each entity consists of two components: data sending component as well as data receiving and control component, as shown in Fig. 3. Not all of the components are active in each entity. For example, The data sending component is inactive in a UDT receiver entity.

There are five timers, maintained by the components of an UDT entity to schedule packet sending and facilitate rate-based and window-based control. The rate that the sender transmits data packets depends on the SND timer, and is independent of the ACK packets received. Similarly, the receiver transmits ACK packets only if the ACK timer expires. The other three timers are NAK, SYN, and EXP, which are used for loss report, rate control, and timeout detection, respectively.

During transmission, both the sender and the receiver maintain their own loss list. It is because that they may have different knowledge on the lost packets due to the asynchronous nature of the algorithm.

UDT defines two classes of packets for communication: data packets and control packets. Control packets can be further categorized as keep-alive, ACK (Acknowledgment), ACK^2 (Acknowledgment of acknowledgment), and NAK (negative acknowledgment). Their usages are shown in Fig. 3.

Each time the sender receives the ACK packet, it will return an ACK^2 packet immediately. The ACK^2 packet can be used to inform the receiver that the ACK packet it sent previously has been received by the sender, and each pair of ACK and ACK^2 packets can be used to calculate the RTT, which is useful in congestion control.

The NAK packet is used for loss report. An NAK packet is sent when loss is detected during data receiving or when an NAK timer expires (and the loss list is not empty). The content of payload depends on which case this NAK packet is sent for. In the former case, the payload carries the sequence number of the lost packet. In the latter case, the payload carries a subset of the loss list.

Since UDT is built on top of UDP, it needs to build its own congestion control mechanism. UDT employs both window-based and rate-based mechanisms. The rate control mainly decides the performance, while the window-based flow control is for efficiency and fairness.

The progresses of both mechanisms undergoes two phases. In both phases, sender updates the window size upon receiving an ACK packet. In the first phase, which is similar to the "slow start" of TCP, the size of window starts from 2 and is updated to the total number of acknowledged packets recorded in an ACK packet. In the second phase, the sender updates the window size by packet arrival speed which is also recorded in the ACK packet supplied by the receiver. The first phase ends when the sender receives an NAK.

The rate control applies to the inter-packet interval and is also associated with the two-phase progress in the flow control. In the first phase, the inter-packet interval is initialized to 0, which means the sender tries its best to send packets. The protocol enters the second phase because of packet losses being detected, the sending rate has to be slowed down and then sped up after the packet loss situation is relieved.

B. Round-Based vs. Rate-Based

In a round-based protocol, such as SFTP, a file transfer is a cyclic exchange of data and ACKs. The source will enter next round only when it receives the ACK from the sink or a

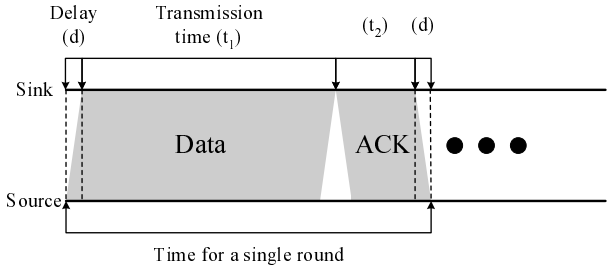


Fig. 4. The effect of network latency in SFTP.

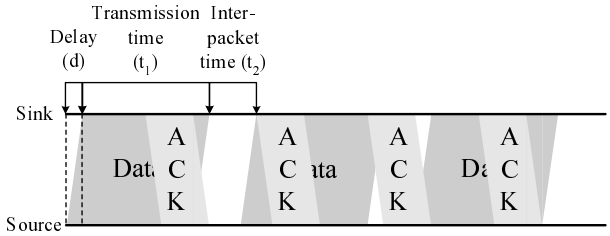


Fig. 5. The effect of network latency in UDT.

timeout occurs. This can be a performance hit. As in Fig. 4, the time for a single round is the summations of transmission time of the data packets, t_1, the ACK, t_2, plus twice of the network latency, $2d$. And the total transfer time for n rounds is $(t_1 + t_2 + 2d) \times n$. If the network delay is increased by d', the total transfer time will be increased by $n \times d'$.

On the contrary, the accumulation of network delay does not exist in UDT. When calculating the total time of file transfer, the network delay is counted only twice (the first and the last ACK). Consequently, when the network latency increases, the effective is "diluted" in the calculation of the total time.

Moreover, as the available bandwidth grows, the transmission time t_1 and t_2 will decrease. Therefore, the proportion of network latency in the time for a single round increases, which means that the performance degradation due to the network latency goes even worse. On the other hand, regardless of the delay increment, if we compare the total transfer time of SFTP to some other transport protocol which is not round-based, the accumulative delay will produce a higher transfer time, and therefore result in a lower effective bandwidth. As a consequence, we can anticipate that the performance of Coda will not be satisfactory in a high BDP environment and it is beneficial to incorporate UDT into Coda.

IV. A Latency-Resistant Cluster File System

To extend cluster file system, such as Coda, beyond the last mile, we have to overcome both bandwidth and network latency issues. In this section, a derivative of Coda that can still perform well even under high latency is described. As discussed in Section III-B, the performance of Coda will downgrade drastically due to the round-based nature of its SFTP protocol. The SFTP protocol will be replaced with the UDT protocol in the proposed system, and the performance of both systems, original and the derivative one, will be evaluated.

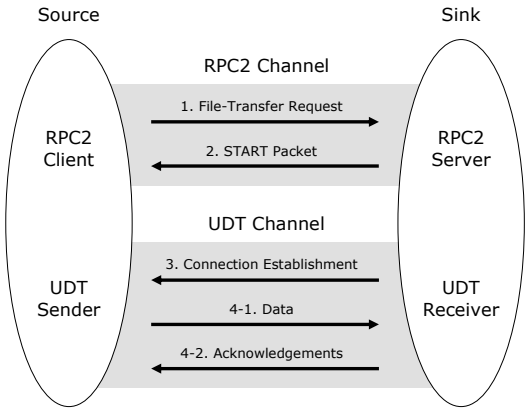


Fig. 6. Details of a file-transfer operation in the prototype implementation.

A. Design and Implementation

In the design of Coda file system, each file-transfer operation involves the concurrent use of two separate communication channels between the source and the sink of file data. The file transfer operations in the two directions between a client and a server generally follow the same procedure with only little differences in minor details. To facilitate our discussion, we take a typical scenario of uploading a file from a client (i.e., a user at home) to a server as an example. In this case, the client is the source of file data, and the server is the sink of file data.

Fig. 2(a) shows the details of the example file-transfer operation. To initiate the file-transfer operation, the source acts as an RPC2 client and issues a file-transfer request via the RPC2 channel. Upon receiving the reply from the RPC2 server, the source invokes the SFTP client as a side effect mentioned previously and sends the file data through the SFTP channel.

In the proposed design, the SFTP client and server are replaced with the UDT sender and receiver, respectively. Instead of transmitting the file data through the SFTP channel, the source now transfers the file data through the UDT channel. Fig. 6 illustrates the new procedure of the improved file-transfer operation.

Before initiating the file-transfer operation, the source first instructs the UDT sender to listen on a well-known UDP port, and then issues the file-transfer request. When the sink receives the file-transfer request, it replies to the source with the START packet and also sends out the connection-establishment request for the UDT channel. Note that it is the convention of UDT for the receiver of data to issue the connection-establishment request. Once the UDT channel is established, the source transmits the file data through the UDT channel.

B. Performance Evaluation

We have conducted a series of tests to understand the performance characteristics of Coda file system as well as the effectiveness of the proposed design when transferring file data across last-mile WAN links. The network configuration

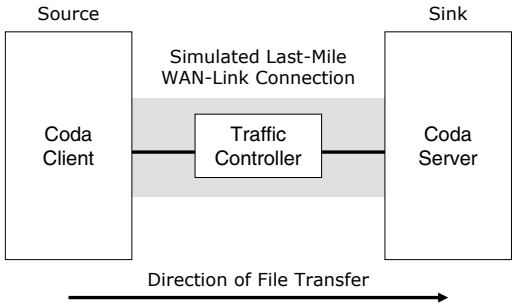


Fig. 7. Network configuration for performance evaluation.

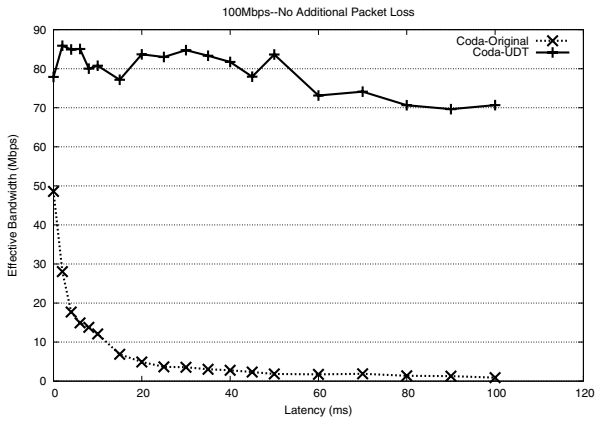


Fig. 8. Measured throughput of Coda file system under different levels of network latency.

used in these tests is depicted in Fig. 7. The Coda client, the traffic controller, and the Coda server are personal computers interconnected by Gigabit Ethernet, and all of them share exactly the same hardware configuration of Intel® Xeon 3.06 GHz, 1 GB DDR SDRAM, and 40 GB hard drives. They also run identical operating systems, Linux with kernel 2.4.20. The file size is 128 MB. In order to measure the throughput of file-transfer operations, additional source codes for profiling have been added to the clients so that statistical information is reported upon the completion of each file transfer.

In addition to the base operating system, we also run NIST Net [16] on the traffic-controller box to throttle available bandwidth of network connections and optionally inject additional delay or cause packet losses. For the purpose of simulating a typical last-mile FTTH/FTTB connection, we set the limit of available bandwidth to be 100 Mbps. The results of performance evaluation are depicted in Fig. 8 and 9.

Fig. 8 shows that when SFTP is used as the transmission protocol, the file-transfer throughput of Coda file system degrades dramatically as network latency increases. In contrast, the increment of network latency only causes slight impact on the measured throughput of the proposed design. In particular, the throughput of Coda file system drops below 1 Mbps when network latency climbs to 100 milliseconds, while the proposed design still sustains the throughput of more than 70

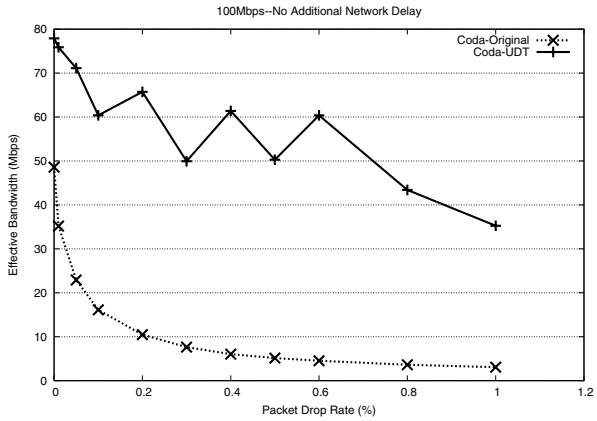


Fig. 9. Measured throughput of Coda file system under different packet-loss rates.

Mbps. Fig. 9 compares the resistance to packet losses between Coda file system and the proposed design. It is clear that the proposed design also exhibits better resistance to packet losses than Coda file system does.

We also discover that SFTP seems to be incapable of achieving high bandwidth utilization. Further tests confirm that the measured throughput of SFTP barely reaches 100 Mbps even with a direct Gigabit-Ethernet connection between the source and the sink.

V. Conclusions

In this paper, we first identify the primary issue of network latency when cluster file systems are deployed and used across the last miles, and provide quantitative evaluation of the impact on a cluster file system. We also propose an effective solution and implement it by replacing the round-based data transmission protocol of Coda file system with a rate-based one.

The prototype implementation not only serves as a proof of concept but also gives us an idea about the effectiveness of the proposed design. The results of performance evaluation show that when the round-based SFTP is adopted, the throughput of file-transfer operations degrades dramatically as network latency increases, while the proposed design, which incorporates the rate-based UDT, is able to sustain superior throughput even under high network latency.

In addition, the proposed design exhibits better resistance to packet losses and has much higher utilization of available bandwidth. All these results together lead us to the conclusion that the proposed design overcomes the difficulties resulting from the last-mile WAN links and successfully extends the boundary of cluster file systems beyond the last miles.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[2] "The Linux Kernel Archives," http://www.kernel.org/. [Online]. Available: http://www.kernel.org

[3] "AFS Reference Page," http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/afs.html. [Online]. Available: http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/afs.html

[4] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere, "Coda: a highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, pp. 447–459, 1990.

[5] P. J. Braam, "The coda distributed file system," *Linux Journal*, vol. 1998, no. 50es, p. 6, 1998. [Online]. Available: http://www.coda.cs.cmu.edu/ljpaper/lj.html

[6] M. Satyanarayanan, M. R. Ebling, P. J. Braam, and J. Harkes, *Coda File System User and System Administrators Manual*, 1997. [Online]. Available: http://www.coda.cs.cmu.edu/doc/html/manual/book1.html

[7] M. Satyanarayanan, R. Draves, J. Kistler, A. Klemets, Q. Lu, L. Mummert, D. Nichols, L. Raper, G. Rajendran, J. Rosenberg, and E. H. Siegel, *RPC2 User Guide and Reference Manual*. [Online]. Available: http://www.coda.cs.cmu.edu/doc/html/rpc2_manual.html

[8] Peter J. Braam and Robert Watson, "RPC2: A Snapshot of Understanding," http://www.coda.cs.cmu.edu/doc/ppt/rpc2.ppt. [Online]. Available: http://www.coda.cs.cmu.edu/doc/ppt/rpc2.ppt

[9] M. K. Lottor, "Simple file transfer protocol," RFC 913, 1984.

[10] S. Floyd, "HighSpeed TCP for large congestion windows," IETF Internet Draft, 2003, http://www.icir.org/floyd/papers/draft-floyd-tcp-highspeed-02.txt.

[11] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2002, pp. 89–102.

[12] Y. Gu, X. Hong, and M. Mazzucco, "Sabul: A high performance data transfer protocol," Laboratory for Advanced Computing, University of Illinois at Chicago, 2003. [Online]. Available: http://www.dataspaceweb.net/papers/sabul-hpdt-03.pdf

[13] Y. Gu and R. L. Grossman, "UDT: A transport protocol for data intensive applications," IETF Internet Draft, 2003.

[14] Y. Gu, R. Grossman, X. Hong, and M. Mazzucco, "Using UDP for reliable data transfer over high bandwidth-delay product networks," Laboratory for Advanced Computing, University of Illinois at Chicage, 2003. [Online]. Available: http://www.lac.uic.edu/~grossman/papers/sabul-protocol.pdf

[15] R. L. Grossman, M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang, "Simple available bandwidth utilization library for high-speed wide area networks," *Journal of Supercomputing*, vol. 34, no. 3, pp. 231–242, 2005.

[16] M. Carson and D. Santay, "NIST Net: a Linux-based network emulation tool," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, 2003.