NFS Dynamics Over Flow-Controlled Wide Area Networks

Koling Chang, Robert Morris and H.T. Kung
Division of Engineering and Applied Sciences, Harvard University
29 Oxford Street, Cambridge, MA 02138, USA
{chang, rtm, htk}@eecs.harvard.edu

Abstract

The Network File System protocol (NFS) has been the leading distributed file system for workstations since it was first introduced by Sun Microsystems in 1986. The geographical scale of NFS has been limited to the local area due to its relatively low performance on the wide area Internet. However, with the advent of high bandwidth wide area networks such as ATM, NFS over WANs may become more promising. In this paper, the performance of NFS over various sizes of WAN is studied. The effects of ATM flow-control and queuing strategies on NFS are discussed, as are the performance of TCP and UDP as NFS transport protocols. The primary conclusion is that standard NFS over UDP works well over ATM WANs as long as ATM-level flow control keeps the cell loss rate under one percent. In some cases, NFS over TCP works badly with small packets due to unfortunate interactions with TCP's congestion window.

1. Introduction

In this paper we study the performance of NFS [1, 3] over flow-controlled ATM networks with various geographical scales and drop rates. NFS has poor performance over wide area networks such as the Internet mainly because of three factors: high drop rate, long propagation delay, and low available bandwidth. The latter is most significant for NFS operations such as read and write that involve large amounts of data. However, large transfers are substantially less common than small ones in most NFS workloads [4]. In such cases, NFS performance is affected more by packet drop rate and propagation delay.

Our research explores four issues relating to NFS and WANs. ATM WANs may use flow control to reduce losses; NFS is affected by both the flow control and the remaining losses. NFS can be used over TCP on WANs, and we wish to compare this with NFS over UDP on a flow-controlled ATM. Even the best WAN cannot reduce propagation delay, and we identify the situations in which propagation is the limiting factor in performance. Finally, sophisticated ATM switches may provide per-VC queuing and round-robin scheduling, and we show the degree to which this improves NFS performance.

1.1 NFS Overview

Sun Microsystems' Network File System (NFS) [1, 3] allows programs to access remote files in the same way as local files. It has seen wide use on a variety of computers since its introduction in 1986. This is mainly due to its simplicity and independence of machine and operating system. NFS is a client-server system. The protocol is defined in terms of a set of procedures, and their arguments, results, and effects. These procedures are implemented using Sun's RPC (Remote Procedure Call) protocol [2] to simplify programs.

All NFS procedures involve a client sending a request to the server and waiting for the reply. RPC automates this by making the client block until the server has completed the call and returned the result. If the server does not respond because of a crash or packet loss, the client periodically resends the request.

The most common operations are client requests to look up file names or to fetch file attributes such as read/write permissions or file length; these involve small transfers of a few hundred bytes over the network, and their performance is dominated by propagation latency rather than bandwidth. In contrast, NFS read and write operations usually transfer file blocks of 8192 bytes and more than one outstanding request is allowed. In this case, network bandwidth can potentially limit their performance.

NFS can be implemented on top of different transport protocols because Sun's RPC is designed to be transport independent. However, the implementation on top of UDP is mostly used since UDP has historically incurred less CPU overhead than TCP [9], and because NFS is almost always used on local networks where TCP's sophisticated error recovery and congestion control are not required [5].

0-8186-7780-5/97 $10.00

However, implementations of NFS on TCP have recently demonstrated performance better than UDP over congested networks [4]. The reason is that TCP has better congestion control and faster error recovery than UDP-based NFS. Whenever a packet is lost, NFS/UDP waits a timeout period on the order of one second and retransmits the entire request, which might involve multiple packets. TCP, in contrast, can often respond to a single dropped packet in one round trip time, and under many circumstances it re-sends only the lost packet rather than a whole request. Thus NFS over TCP should tolerate higher loss rates than NFS over UDP.

1.2 Flow-Controlled ATM

The high drop rate and low bandwidth on current wide area networks may eventually be improved when Broadband ISDN using ATM becomes available. ATM [7], which aims at high bandwidth and low data loss rate, has been adopted by telecommunication providers as the standard for future wide area digital networks. Several flow-control protocols have been proposed for use at the ATM level to provide "available bit-rate" (ABR) services and reduce data loss; these include rate-based flow-control [8,10] and credit-based flow-control protocols [15]. Both of these approaches will allow heavily loaded networks to achieve high utilization and low data loss.

The rate-based flow control method recommended by ATM Forum, Enhanced Proportional Rate Control Algorithm (EPRCA) [11], is an end-to-end protocol in which the source of a connection adjusts its sending rate depending on the network congestion status along its path. EPRCA does not eliminate cell loss due to congestion. The cell drop rate varies depending on the available buffer space in the network switches and protocol parameters. In general EPRCA allows a trade-off between drop rate and bandwidth utilization when the switch bandwidth and buffer resources are limited.

Credit protocols such as QFC [15] and FCVC [12,13,14] use hop-by-hop buffer based flow-control. FCVC eliminates cell loss by reserving buffer space, but requires per-VC queueing inside the network switches. While per-VC queuing may increase a switch's cost, it provides performance benefits beyond those provided by other flow controls. A number of switch vendors provide it independently of flow-control algorithm. For instance, the Fore Runner ASX-200BX ATM Switch from FORE SYSTEMS is a per-VC queueing switch even though it does not support credit-based flow-control.

1.3 Paper Outline

This paper is arranged as follows: In Section 2, we describe our trace-gathering setup, simulator test-bed, and benchmarks. In Section 3, we use trace-driven simulations to explore NFS performance on a lossless network with various propagation delays. The TCP overhead due to slow-start is also shown. In Section 4, we simulate NFS over a loaded network with various cell drop rates, and with the EPRCA rate-based flow-control protocol. In Section 5, we show the performance advantages of per-VC queueing and credit-based flow-control for NFS performance. We conclude this paper in Section 6.

2. Experimental Test-Bed

In order to evaluate the performance of NFS without a wide-area ATM net dedicated to our experiments, we used a network simulator. This simulator is an event-driven ATM network simulator featuring rate-based EPRCA flow-control, credit-based FCVC flow-control, TCP, and AAL5 tail dropping mechanism with partial packet discard. The EPRCA implementation follows the latest ATM Forum Traffic Management specification of March 1996 [8]. The TCP stack in this simulator is taken from the source code of NetBSD 1.1.

Our NFS simulations are trace-driven; we do not model the applications that use NFS. We collected traces of real NFS traffic on a network which directly connects an NFS client and server. The traces contain the following data:

- Type
- Time of Request
- Time of Response
- Request Packet Size
- Response Packet Size

The simulator calculates the amount of time the client computes between requests and the amount of time the server takes to serve each request by assuming a propagation time of zero. The simulator preserves the computation times when it uses the trace, and varies only the amount of time it takes the network to forward requests and responses. The details of this procedure are:

- If a request is not a READ or a WRITE, only one outstanding request is allowed. When the reply arrives, the interval until the next request is sent will be the same as in the trace.

- The time between when a client issues a request and when it receives the response is composed of three parts. The first part is the time it takes the network to forward the request; the simulator calculates this based on propagation and queuing delay. The second part is the amount of time the server spends serving; this is taken from the trace. The third is the simulated time taken by the network to forward the response.
- Some READ transactions are actually read-ahead, for which the client does not wait. Similarly, some WRITE requests are actually write-behind, for which the client also does not wait. The simulator decides whether the client must wait for READs and WRITEs by analyzing their original Request and Response Times in the trace. If the difference between Request Times of consecutive READs or WRITEs is smaller than a threshold, or the Request Time of the next transaction is smaller than the Response Time of current transaction, the interpreter treats these transactions as read-ahead or write-behind. In this case, more than one outstanding request is allowed.

The client machine in these traces was a 133 mHz Pentium PC running NetBSD 1.0. The server was a 133 mHz DEC Alpha 3000/400 running OSF/1 3.0. Each traced benchmark was run after a reboot to prevent data being cached between runs. They are connected by a 10 Mbps Ethernet with no other competing traffic.

The main performance metric in these experiments is the task completion time (TCT) of a particular benchmark listed in Table 1. These tasks involve the Berkeley MPEG encoder "mpeg_encode-1.3" source code distribution. As detailed in Table 1, these traces include "ls -l" on the source directories, "cp -r" of the source directory before compiling, "make" inside the source directory which compiles an mpeg_encode executable file, and finally running the executable with 200 frames of non-compressed 360x240 video as input to produce a 1.2 MByte compressed mpeg file. All of these commands are run on the NetBSD client, with all files accessed using NFS to the server. For comparison, we also execute the encoder with the input raw frame data stored locally and output mpeg file remotely.

3. Effects of Propagation and TCP Overhead

Most NFS transactions are small independent requests, each of which must complete before the next can start. Thus as propagation time increases it will eventually dominate the task completion time. Before this point, the TCT is mainly affected by the underlying flow-control protocols, network congestion status, and processing time on the client and server.

Table 1: Command Task List

Command	Description
	S:bytes sent by client, R:bytes sent by server, TC=transaction count, TCT=Task Completion Time of original trace in seconds
ls	ls -l * */* of 37 files in the mpeg_encode source directory
	S:54KB, R:62KB, TC=453, TCT=2.11 sec.
cp	cp -r of the mpeg encoder source tree, 0.95 MegaBytes
	S:1034K, R:1032K, TC=771, TCT=9.58
cp-1	copy single 1 megabyte file
	S:1122K R 1126K, TC=268, TCT=5.23sec
make	compile mpeg_encoder which reads 0.95 mb source and generate 0.3 mb output.
	S: 672K, R: 1196K, TC=2023, TCT=60sec.
encode-remote	read 23 mb frame data and generate 1.2mb mpeg file; input and output are remote.
	S:1680K, R:23M, TC=3519, TCT=66.59 sec
encode-local	same as encode remote except the 23 mb input is stored on client local disk
	S:1247K, R: 157K, TC=173, TCT=50.094sec

In this section, we show simulation results for a lossless network with varying client-server distances. The network has link rates of 155 Mbps and carries no competing traffic. Figure 1 shows the TCT of several benchmarks that use NFS/UDP. Both X and Y axes are in log scale. When the network delay is small, the TCT is dominated by disk latency and processing time. However, once the network delay exceeds roughly 10 milliseconds, it is the major factor in TCT, and the TCT increases linearly with delay. We can see that the propagation delay dominates the TCT of ls earlier than the other benchmarks. This is because ls has only short requests and is not CPU bound. An RTT of less than 100ms appears to be the most tolerable, since beyond that even simple tasks such as ls take a minute or more.

One interesting detail of these results is that the TCT of encode-local increases relatively modestly with prop-

agation delay. Encode-local only uses NFS to write output to a file, and produces data slowly enough that NFS write-behind can avoid blocking the process. NFS can keep up to five write requests outstanding at a time; since each completes in roughly one network round trip time, NFS write throughput is limited to five blocks per round trip time. encode-local starts to slow down when the round trip time is so high that this throughput drops below the rate at which encode-local produces output.

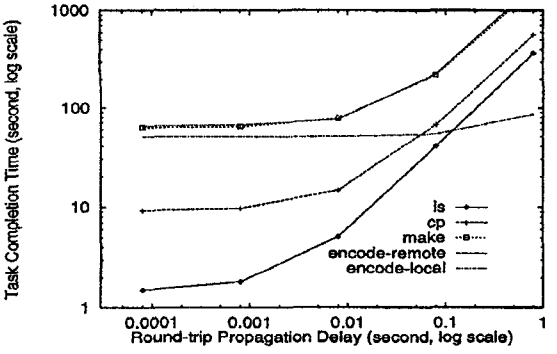

Figure 1: Task Complete Time vs. Round-trip Propagation Delay for NFS/ UDP over Perfect Net

NFS over TCP performs as well as NFS over UDP as long as the packet size (called MSS) is large, or 9192 bytes for these simulations. This packet size is typical for ATM networks. With packets of only 512 bytes, as are typically used over WANs, the TCTs of make and encode-local are around 15% and 30% larger with TCP than UDP respectively (see Figure 2). This degradation is due to TCP's slow-start ramp up time. If TCP is about to send something, but the connection has been idle for at least one round trip time, TCP first shrinks its congestion window. This decreases NFS write performance, since an 8192-byte NFS write will require many round trip times to send if the congestion window is only a few 512-byte packets. There is no similar effect for read requests because TCP does not consider the connection idle when the server sends the response: the client's request made the connection not idle. The congestion window does not affect requests other than read or write because all other requests can fit in the minimum size window. TCP with large packets always works well because even an 8192-byte write fits in a single packet.

A close reading of Figure 2 reveals that the TCT for encode-local over TCP actually decreases when the network delay increases. When the delay is small, the idle time between client requests is large relative to the round trip time; thus TCP often shrinks the client's congestion window. When the network delay is large, the client's relatively short inter-requests delays do not

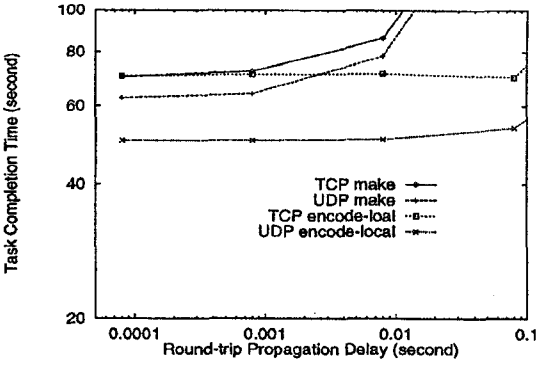

Figure 2: Comparing of TCP and UDP on "make" and "encode-local"

trigger TCP's idle behavior. Figure 3 compares the congestion windows of these two cases over time, for round-trip delays of 8 and 80 milliseconds. This phenomenon does not happen with encode-remote, which issues NFS requests much more often than encode-local because it reads input over NFS. These read requests prevent TCP from considering the connection idle.

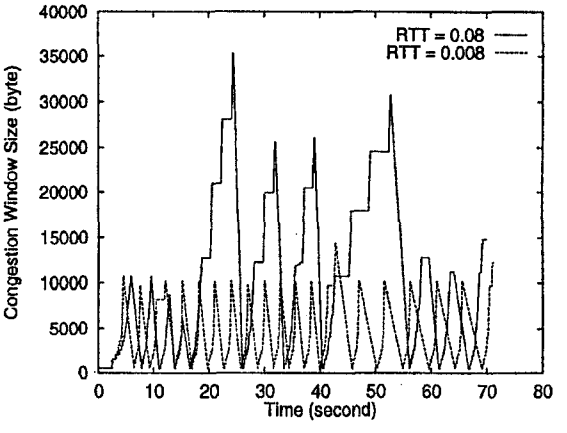

Figure 3: Congestion Window Size for Two "encode-local" Cases with Different RTT

4. Performance with Loaded Networks

Other work [4] has demonstrated that NFS/UDP performs badly over heavily loaded networks, and that NFS/TCP offers performance advantages. TCP recovers from packet loss faster than NFS/UDP, and can re-send just the lost packet rather than an entire request. TCP's adaptive congestion control window also allows it to use less switch or router buffer space than NFS/UDP, which decreases packet loss. These advantages hold primarily when all competing traffic uses TCP. If other traffic relies on ATM flow control, as is expected in future

networks, TCP's advantages for NFS are less significant. This section reports simulation results for NFS with both ATM flow control and either TCP or UDP, as detailed in Table 2.

Table 2: NFS Transport Protocols

Name	Description (MSS:Packet Size)
TCP-9192	TCP with MSS = 9192
TCP-512	TCP with MSS = 512
UDP	Raw AAL5 packet

The simulations involve a heavily loaded network depicted in Figure 4. Each VC is flow-controlled by the rate-based EPRCA protocol. The parameters for EPRCA are adjusted to vary the network's cell drop rate. The bottleneck switch has 2500 to 3500 cells of buffering depending on the drop rate. The 50 competing connections, 25 in each direction, send data as fast as EPRCA will allow. The NFS connection uses either TCP with no NFS retransmission timers or UDP with standard 0.875 second based retransmission timer.

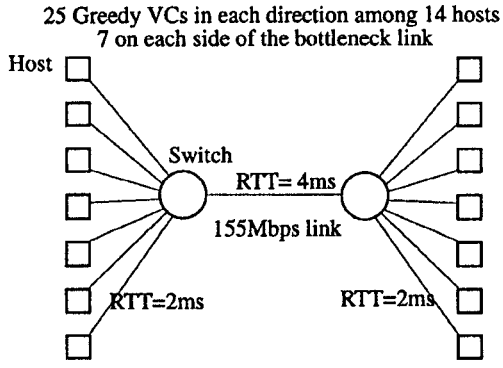

Figure 4: Loaded ATM Network Configuration

NFS over UDP and NFS over TCP with 9192-byte packets perform similarly in most cases. TCP-512 does not perform well when the drop rate is high and the amount of data to transfer is large. As seen in Figure 5 and Figure 6, TCP-512 performs slightly worse for "ls" and "make" than TCP-9192 and UDP. However, TCP-512 performs much worse than TCP-9192 and UDP in encode-remote and "cp-1" as seen in Figure 7 and Figure 8. This is because encode-remote transfers much more data than make. The cell loss causes the server's TCP congestion window to shrink often, which causes TCP-512 in particular to spend many round-trip times increasing the window again.

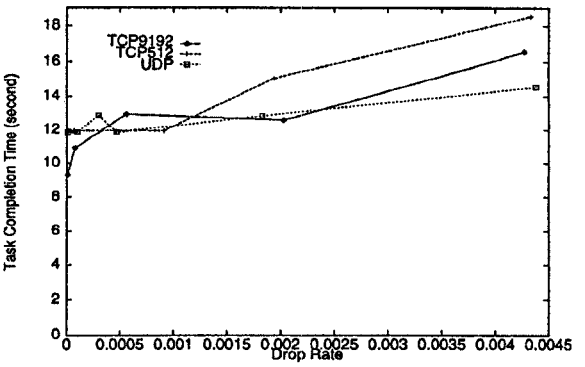

Figure 5: Task Completion Time of "ls"

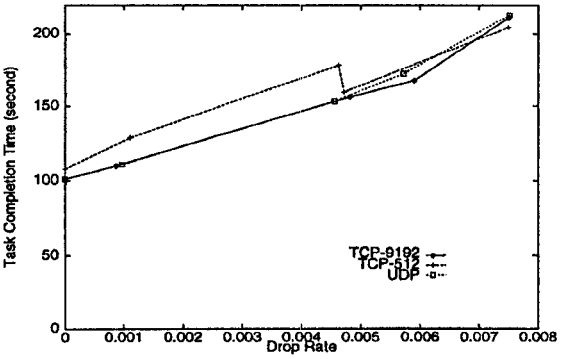

Figure 6: Task Completion Time of "make"

In these simulations, cell drop rate itself is not a configurable parameter. Instead, it is the measured result of a specific buffer size and EPRCA parameter set. That is why the drop rate index distance between each of the sample points in these graphs vary.

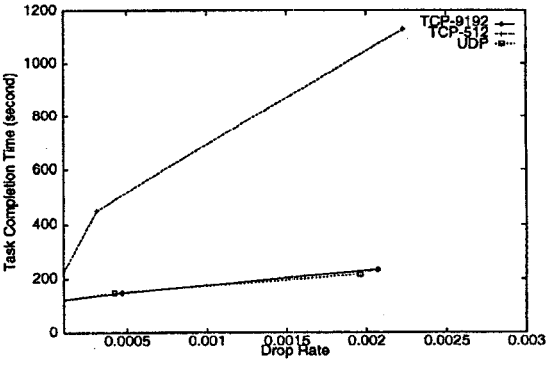

Figure 7: Task Completion Time of "encode-remote"

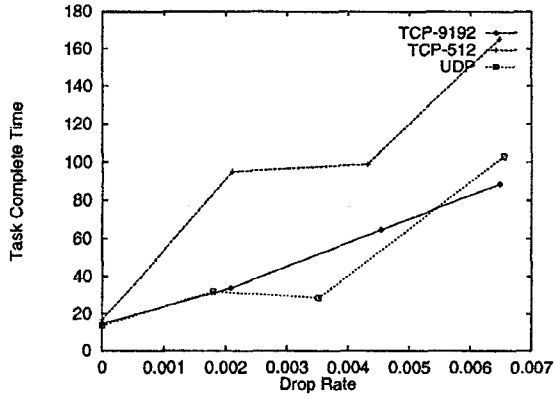

Figure 8: Task Completion Time of "cp-1"

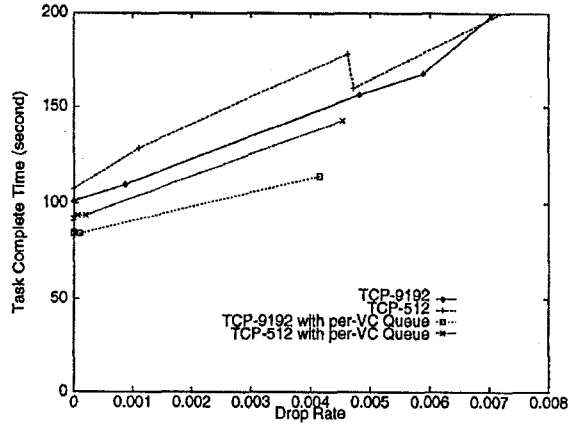

Figure 9: Per-VC Queueing vs. Shared Queueing in "make"

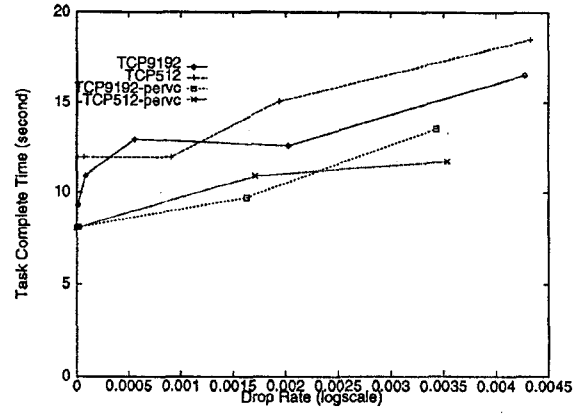

Figure 10: Per-VC Queueing vs. Shared Queueing in "ls"

5. Per-VC Queueing and Credit-Based Flow-Control

NFS, as a request-response protocol, is at a disadvantage when competing with "greedy" sources that always have data ready to send. Flow control mechanisms, while limiting the rate at which such sources send, typically allow them to keep fairly large amounts of data buffered in network switches. This leads to extra queuing delay. Senders that are limited only by flow control can maintain a high rate regardless of delay. Most NFS transactions, however, must slow down as queuing delay increases, since the client waits for each response before sending the next request. Even for READ and WRITE transactions, only 2 outstanding READ and 5 outstanding WRITE requests are allowed.

Switches that maintain separate queues for each connection (called per-VC queuing) and serve the queues fairly will decrease this effect. When an NFS request arrives at a switch, the NFS VC queue will be empty; thus the request will be sent relatively soon.

In the simulations discussed in the previous section, switch queue lengths varied from 0 to 3500 cells. At 155 megabits/second, this would impose a queuing delay of up to 10 milliseconds per NFS request. Figures 9 and 10 show the advantages of per-VC queuing for the "make" and "ls" benchmarks with EPRCA flow control.

Credit-based flow control performs better than rate-based for these traces for two reasons. First, credit flow control does not spend time ramping up its rate. Second, credit systems have round-robin scheduling among VCs. The simulation results of NFS over static FCVC with round trip propagation delay equal to 8 millisecond are listed in Table 3.

Table 3: Performance Improvement in FCVC with RTT = 8 millisecond

Task	ls	cp	cp1	make	encode -r
Perfect Net	5.1437	14.701	11.681	78.21	76.73
Static FCVC	5.3016	17.239	12.144	80.392	96.729
EPRCA Single FIFO with TCP-9192 and No Drop	9.1898	23.97	13.612	101.07	114.80

6. Conclusions

The performance of typical NFS tasks starts to suffer significantly when client and server are separated by

more than about 500 miles ($RTT \cong 10ms$). At that time a single "ls" command might take more than a minute to finish.

If ATM flow control is used to decrease packet loss, NFS over UDP performs as well as NFS over TCP when TCP uses large packets. With small packets, TCP spends an excessive amount of time with a congestion window smaller than an NFS request; this causes single requests to take many round trip times to transmit.

If ATM switches also implement per-VC queuing and round-robin scheduling, NFS can compete fairly with other traffic, further increasing performance. The combination of ATM flow control and per-VC queuing promise high performance for NFS over wide-area networks of reasonable size.

References

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem", Proceedings Summer 1985 USENIX Conference, Portland OR, June 1985.

[2] "RPC: Remote Procedure Call Protocol Specification", RFC 1057, Internic, http://www.internic.net/rfc/rfc1057.txt, June 1988.

[3] "NFS: Network File System Protocol Specification", RFC 1094, SRI Network Information Center, Menlo Park, CA March 1989.

[4] Rick Macklem, "Lessons Learned Tuning the 4.3 BSD Reno Implementation of the NFS Protocol", Winter USENIX Conference, Dallas, TX, 1991.

[5] Bill Nowicki, "Transport Issues in the Network File System", Computer Communication Review, pg 16-20, March 1989.

[6] Rick Macklem, "The 4.4BSD NFS Implementation", Sun Microsystems Inc.

[7] ATM Forum ATM User-Network Interface Specification, Version 3.0, 1993.

[8] "Traffic Management Specification Version 4.0", ATM Forum Technical Committee, af-95-0013R11, March 1996.

[9] Van Jacobson, "Congestion Avoidance and Control", in Proceedings of SIGCOMM 1988.

[10] Charny, A., Clark, D.D., Jain R., "Congestion Control With Explicit Rate Indication", Proc. ICC95, June 1995.

[11] L. Roberts, "Enhanced PRCA (Proportional Rate Control Algorithm)", AF-TM 94-0735R1, August 1994.[

[12] Kung, H.T., R. Morris, "Credit -Based Flow Control for ATM Networks", IEEE Network Magazine, Vol. 9 No.2 March/April 1995.

[13] Kung, H.T. and K. Chang, "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow control for ATM Networks", Proceedings of INFOCOM 1995, April 2-6, 1995.

[14] Kung, H.T., T. Blackwell, A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing", Proceedings of ACM SIGCOMM '94.

[15] "Quantum Flow Control", Version 2.0, FCC-SPEC-95-1.