Improving NFS Performance over Wireless Links*

Rohit Dube, Cynthia D. Rais and Satish K. Tripathi
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742
{*rohit, cldavis, tripathi*}*@cs.umd.edu*

December 19, 1995

Abstract

NFS is a widely used remote file access protocol that has been tuned to perform well on traditional LANs which exhibit low error rates. Users migrating to mobile-hosts would like to continue to use NFS for remote file accesses. However, low bandwidth and high error-rates degrade performance on mobile-hosts using wireless links thus hindering the use of NFS. In this paper, we present two mechanisms to improve NFS performance over wireless links : an aggressive NFS client and link-level retransmissions. Our experiments show that these mechanisms improve throughput by up to 200%, which brings the performance to within 5% of that obtained in zero error conditions.

1 Introduction

Mobile computing is increasingly in demand and will be an important part of the computing infrastructure in the near future. The use of wireless links gives the mobile user new freedom and flexibility. However, most applications and reliable transport protocols have been optimized for wired networks and static hosts. These suffer from unacceptable performance when used on wireless systems. The performance of network protocols and applications over wireless links is currently limited by low bandwidths, high error rates, temporary disconnections, and high latencies. Protocols and applications must accommodate for these characteristics in order to provide acceptable performance to mobile users.

In addition to transmission limitations, mobile users are constrained by limited disk-space. Unable to store all their data on their local disks, users often must fetch files from servers on the wired network via wireless links. The Network File System (NFS) protocol has been widely used on wired LANs to provide a mechanism for remote file access [WLS+85], [SGK+85], [PJS+94]. Users migrating to mobile-hosts from stationary workstations want to continue to use NFS to access their files. However, the bursty and higher error rates prevalent over wireless media pose performance problems for mobile applications which use NFS mounted files.

NFS was designed for faster physical networks which exhibit rare random errors. Therefore, packet losses were assumed to occur due to either congestion on the network or a server

*This work is supported in part by NSF grant CCR 9318933 and IBM equipment grants.

failure. NFS clients could safely back-off and retry a request after waiting for some predetermined time period. On wireless links, packet losses are usually due to burst errors [BBKT96], rather than network congestion or server failures. These burst periods are of the order of a hundred milli-seconds. In response to such losses, NFS clients back-off to unnecessarily long wait periods, leading to severe performance degradation. This degradation has been previously documented in [RT95].

Previous attempts at quantifying and improving performance over wireless links have concentrated mostly on TCP [DCY93], [YB94], [BSK95], [BB95b]. Since, most NFS implementations are UDP based [Nov89], the results from these TCP studies are not directly applicable to NFS. It is the goal of this paper to find and quantify mechanisms for improving NFS performance over wireless links.

On our testbed, we implemented an error model to allow us control over the errors on the wireless link. We then studied NFS performance over this wireless link. Based on our studies we chose to use smaller block sizes and linear back-off, which result in a more aggressive NFS client. We also analyzed link-level retransmissions as a mechanism to improve performance. As our results show, these mechanisms reduce response times by upto 53% for reads and 67% for writes (throughput increase of upto 100% for reads and 200% for writes). The modifications we made gave us response times within 5% of the zero errors case. The scalability study we performed showed that the more aggressive NFS client behavior and link-level retransmissions, did not have a detrimental effect when multiple mobile NFS clients were present. With multiple mobile clients running a modified version of NFS, response time decreased by about 45% for both reads and writes (throughput increased by 82%).

The rest of this paper is organized as follows. Section 2 discusses related work and our approach towards improving NFS performance. In section 3 we describe the testbed and the error model that we use for our experiments, and section 4 present the results of our experiments. We propose some ideas for future investigation in section 5. Finally, section 6 summarizes this paper and presents our conclusions.

2 Related Work and Solution Approach

Much research has been done to optimize NFS performance on wired LANs [Jus89], [Jus94], [PJS$^+$94]. New file systems have been developed to support disconnected operation via disk-caching [SK92], [SNKP94], [HHRB92]. For improving application performance over wireless links, an M-RPC system has been proposed [BB95a]. Enhancements have been suggested to improve TCP performance over wireless links : split connection approaches [BB95b], [YB94], [BSAK95], and the fast retransmission approach [CI94]. Link level retransmissions have also been proposed to improve performance over wireless links [DCY93], [BBKT96], [PAL$^+$95]. For reasons discussed shortly, most of these results are not directly applicable to improving NFS performance. Our research is the first to combine these issues and investigate performance improvements for NFS over wireless links. We use linear back-off and smaller block sizes on the NFS client on one hand, and link-layer retransmissions on the other to obtain performance improvements.

2.1 Related Work

The performance of **NFS over traditional wired LANs** has been improved by using a server reply cache [Jus89], by using write gathering to improve write throughput [Jus94], and by allowing larger than 8KB block sizes and allowing asynchronous writes [PJS$^+$94]. These modifications improve NFS performance at the NFS server and would not significantly help the performance of NFS in a wireless environment because the bottleneck are the mobile NFS client and the transfer over the wireless link. Some of the standard improvements, such as larger block sizes have a detrimental effect on lossy wireless links and will decrease NFS performance at mobile hosts. Overall, work on improving NFS performance has so far been limited to wired networks with a focus on NFS servers. Hence, these studies are orthogonal to ours.

Using **disk-caching**, Coda [SK92], its later version Odyssey [SNKP94], and Little Work [HHRB92] provide mechanisms for disconnected and intermittent operation. However these systems are not in wide use. It is possible to borrow ideas from them for use with the NFS client on a mobile-host. Prefetching and disk-caching decrease the time spent by the user waiting for file transfers, but that by itself does not solve the problem of poor throughput and wireless link utilization. We discuss the use of disk caching in Section 5.

In the **TCP split connection** approaches [BB95b], [YB94], [BSAK95], the base-station[1] buffers packets being sent to the mobile hosts in its vicinity. The base-station retransmits any lost packets to prevent end-to-end retransmission. The **M-RPC** approach [BB95a], is a variation of the TCP split connection approach. It seeks to improve performance by separating the connection at the RPC[2] level at the base-station. These approaches have the disadvantage of high buffer requirement, a complex migration algorithm and increased load at the base-station. Besides, file system actions which have high consistency requirements would be vulnerable under a split connection approach. In addition, the TCP split connection approach cannot be directly used as most NFS implementations are UDP based; the M-RPC approach cannot be used as it would require major modifications to the RPC and NFS code on the client and the server. Our use of link level retransmissions achieves reliability on the wireless link, but at a lower level in the protocol stack, without creating consistency problems. This has the added advantage of providing a common solution to all higher level applications and protocols.

The **fast retransmission** approach [CI94], improves TCP performance by notifying the transport layer prior to mobile host motion to avoid congestion control policies from being initiated. This approach does not address the poor performance caused by burst errors on the wireless link.

2.2 Solution Approach

In the interests of inter-operability with static-hosts, any attempt towards enhancing performance should be limited to modifications made on the base-stations and on the mobile-hosts. This implies that the NFS servers cannot be modified. Our solution approach does not require any such changes. We suggest modifications only at the mobile host and at the base station.

In studying NFS over wireless links, we noted several factors that caused poor performance. First, NFS uses large block sizes (usually 8KB). A large block size decreases the overhead involved with requesting and sending each block. However, the large blocks must be fragmented (at the MAC layer) before being sent onto the link. If any one of the fragments is lost, the entire block is discarded and must be retransmitted. For the error prone

[1]A Base-station is a bridge between the wired and the wireless segments of a network.

[2]NFS uses RPC [Gro88] for remote communications.

wireless links, this leads to many retransmissions. A smaller block size can decrease these retransmissions.

The second NFS feature which causes poor performance is the exponential back-off algorithm which is used by the NFS clients. When the block requested or an acknowledgement is lost or delayed, the client assumes that the network is congested or that the server is heavily loaded or down. Long back-off periods are appropriate in these cases, so that the client waits until the server is free or the congestion is decreased. In wireless networks, the main cause of losses and delays is the burst errors on the wireless link. Burst errors are much shorter than the typical server failure or congestion period (on the order of 10 to 100 milliseconds). Hence exponential back-off would constitute an over-reaction; the NFS client would achieve better performance using a linear back-off algorithm.

In addition to the NFS client, optimizations can be made at the device driver by using link-level retransmissions. Wireless device drivers don't usually implement a good retransmission policy or a reservation protocol. This leads to bad link utilization and triggers off unwanted behavior (back-off) in the higher level protocol or application involved. Our results show that link-level retransmissions improve NFS performance greatly.

We considered buffering at the IP layer in an attempt to improve performance. We also considered running a TCP connection from the mobile-host to the the base-station where UDP packets could be constructed and relayed to the NFS server (and vice-versa). Implementing either one of these strategies would mean handling NFS data separately from other data flowing between the mobile-host and the base-station and additional buffering, segmentation and re-assembly complications at the base-station. As in the other split connection approaches, guaranteeing consistency would be difficult if not impossible. Both these approaches would interfere with an aggressive NFS client and link-level retransmissions (which attempts to provide a reliable link at a lower level in the protocol stack).

3 System Setup

The environment we consider is that of a building network with base-stations in designated places to handle traffic from mobile-hosts.

3.1 Testbed

The impact of the wireless burst errors on the higher layers is very complex. The errors on the wireless link can be modelled, but due to the vertical dependency of the higher layers on the physical layer, it is impossible to accurately quantify the interactions between the layers. For this reason, it is important to use a real testbed to determine the effect of wireless burst errors on NFS. At the same time, accurate performance evaluation requires us to control the errors at the wireless link.

Figure 1 shows our testbed. The IBM RS6000 (tapti) acts as the NFS server. An IBM RT-PC (notrump) is configured as the base-station. There are two mobile NFS clients: an IBM PS/2 (shivalik), and an IBM RT-PC (narmada) which access the NFS server through the base-station using a 1 Mbps Infra-Red (IR) link. We use the DEC-ALPHA (congo) to monitor traffic on the ethernet by running TCPDUMP [MJ93].

We studied the performance of NFS reads and writes by measuring the *response times*

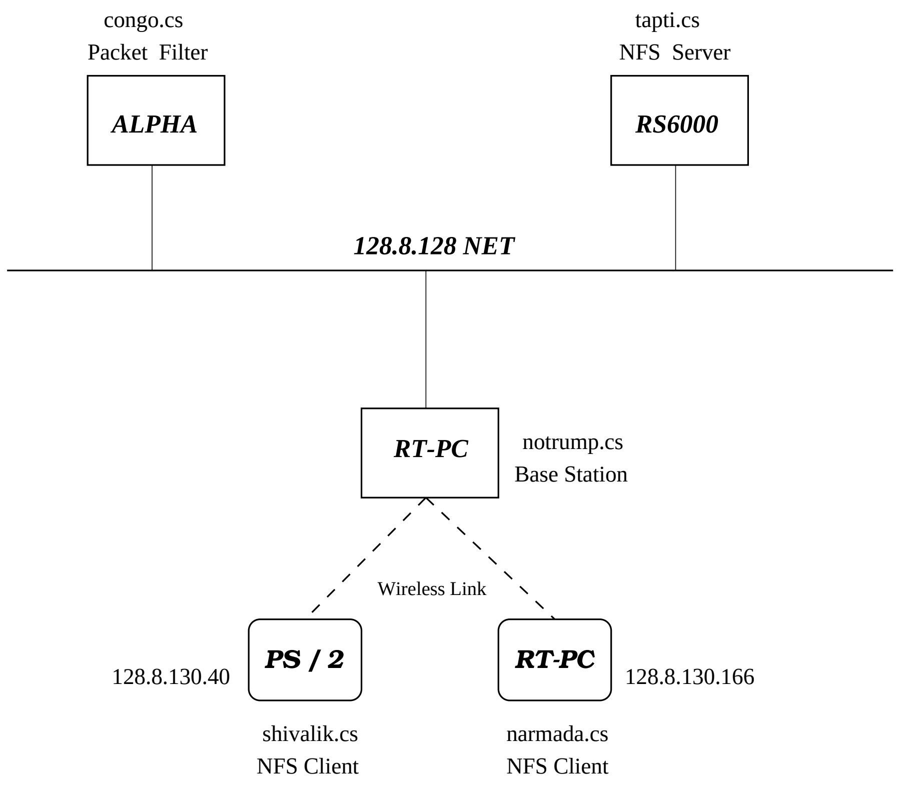

Figure 1: Testbed for NFS Performance Evaluation

obtained for file transfers to[3] and from[4] the local disks of the mobile-hosts. In order to gain control over the error rate on the wireless channel, we placed the IR devices on the base-station and the mobile-hosts in close proximity (to minimize un-controlled error during data transfer between the mobile-hosts and the base-station). An error model was then introduced into the device driver of the mobile-hosts as explained below in 3.2. In order to get consistent results, the ethernet segment holding the static-hosts was isolated from the rest of the building network while running experiments.

3.2 Error Model

We use a 2-state error model [BBKT96], [RT95] as shown in Figure 2 to represent the quality of a channel between the mobile-host and the base-station. The *good-state* is the normal mode of operation when bits flow intact between the target and source devices. Every now and then the communication channel goes into a *bad-state* when bits are corrupted or lost (perhaps due to *Null-spaces* or *Raleigh-fading*). The time the channel spends in the bad-state is usually long enough to force multiple bits into error. These erroneous bits may span more than one frame, leading to burst errors on the channel. It should be noted that a temporal model for the behavior of the communication channel is desired. This is because burst errors exhibited by the channel are a function of motion and electromagnetic interference which can in turn be modeled as functions of time. We model the time spent in both the good and bad states as

[3]Copying a file across the network from an NFS server to the local disk constitutes an NFS read.
[4]Copying a file across the network from the local disk to the NFS server constitutes an NFS write.

uniformly distributed periods ([RT95] showed that the degradation of NFS performance with uniformly distributed periods was worse than performance degradation with a deterministic error model. We chose uniform distributions since they are more realistic.).

This error model was incorporated into the output and input routines of the mobile-host's device drivers[5]. Based on our experience and results from [BBKT96] and [RT95], we chose a mean of 1000 milli-seconds for the *good-period* (time spent in the good-state) and experimented with means of 40 and 80[6] milli-seconds for the *bad-period* (time spent in the bad-state). The error model was implemented by using the kernel TIMEOUT mechanism on the mobile-hosts. The granularity of timeouts signalled by this mechanism was 20 milli-seconds. Hence, we used discrete functions in the range $0 \ldots 200$ for the bad periods and $0 \ldots 2000$ for the good period for the uniform distributions.

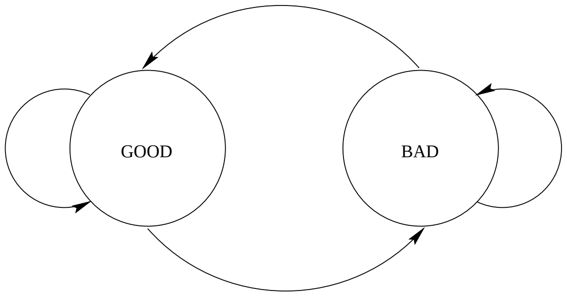

Figure 2: The 2-state Error Model

4 Experimental Results

We conducted experiments with various file sizes. For brevity, only the response times for experiments with 1 MB files are reported here. Other files sizes yielded similar results. Each experiment was run 10 times[7] and the minimum, maximum and average *response times* were recorded. All data (response times) is in seconds unless otherwise specified.

Our experiments indicate that reads where in general slower than writes. We traced this problem to the fact that the mismatch in the bandwidth of the wired and wireless media, causes buffer overflow at the receive buffers of the mobile-hosts. The output of TCPDUMP in Figure 3 shows that because of buffer overflows, there were frequent retransmissions for the read, leading to bad performance. There was some overflow for the writes too, but the problem was milder. The bottle-neck therefore was the mobile-host which was unable to cope with the data being dumped on it by the base-station.

[5]Alternatively, we could have put this error model completely into the base-station or distributed it between the input or output routines of the base-station's and the mobile-host's device drivers.The former would be difficult to scale as per destination channel status would have to be maintained at the base-station. The later would allow us lesser control over the errors introduced into a communication channel as synchronizing the two models would be impossible.

[6]Our experiments with other *means* yielded similar results. We chose to discuss the results for the 1000 milli-second good period and the 40 and 80 milli-second bad periods, because these were the most appropriate for wireless networks in a building environment.

[7]We have also experimented with more number of runs. The results obtained were similar to the ones we present.

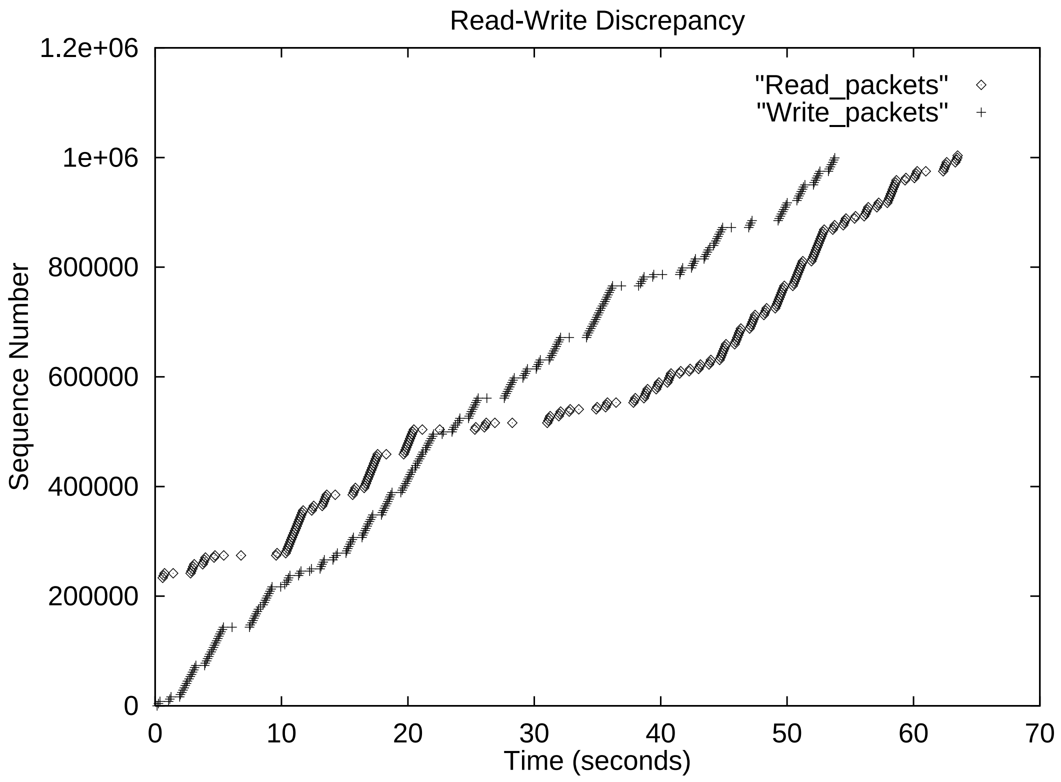

Figure 3: Effect of buffer overflow on reads and writes

Bad Period	**Read**			**Write**		
(milli-seconds)	Min	*Avg*	Max	Min	*Avg*	Max
40	43.3	*49.2*	51.9	22.5	*24.7*	27.7
80	54.8	*67.7*	78.4	33.4	*45.5*	54.4

Table 1: No Changes; 4096 byte Block

In the following sections, we look for general trends amongst the data presented. Interpreting the data literally or analyzing every departure from an expected result is futile because of the effects of variations on the wired and wireless media, non-uniform disk latencies and buffer overflows. We first present detailed results for the single mobile-host case (shivalik) and then discuss how these results scale in the presence of multiple mobile-hosts (shivalik + narmada).

4.1 Effect of Block Size

We first studied the effect of block size on the response time. As mentioned before, two different bad-periods, 40 and 80 milli-seconds, were used. The results are shown in Figure 4 and Table 1. As can be seen from Figure 4, there is a knee at 4096 bytes for the plots of the 40 milli-second read, 40 milli-second write, and the 80 milli-second write case. 80 milli-second reads show a bell-curve with the best performance at 4096 bytes. Since NFS performance

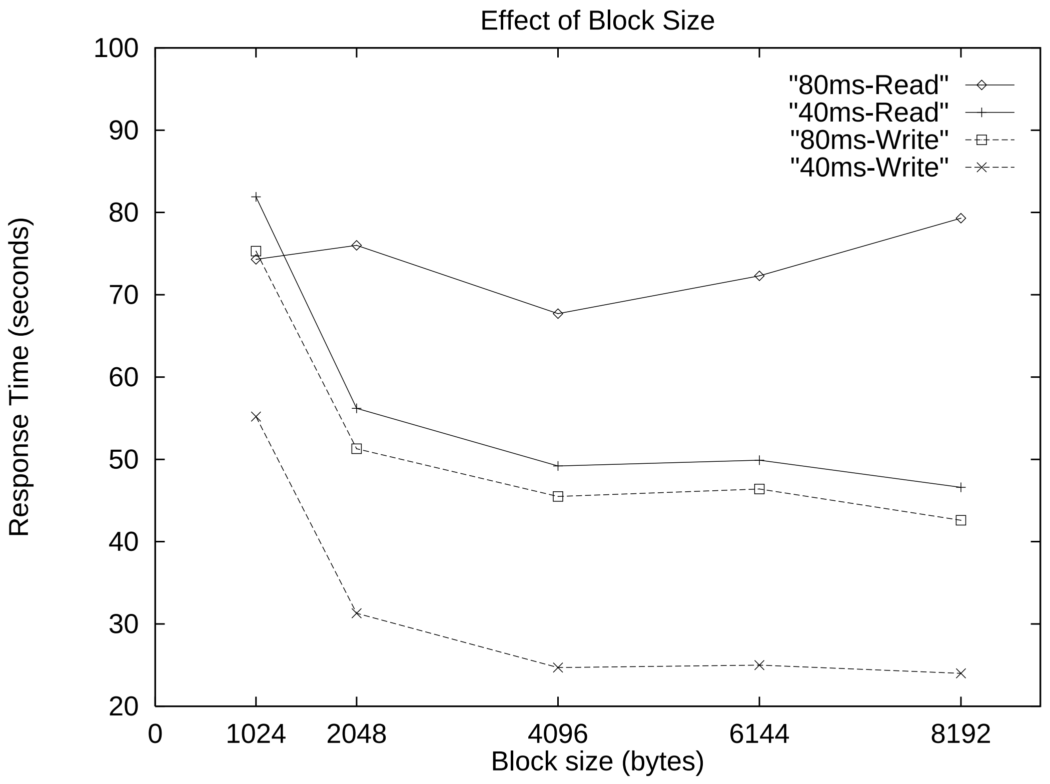

Figure 4: Effect of Block Size

Bad Period	**Read**			**Write**		
(milli-seconds)	Min	*Avg*	Max	Min	*Avg*	Max
40	42.1	*48.0*	54.4	21.9	*24.6*	26.7
80	43.8	*48.7*	52.9	22.6	*24.4*	27.5

Table 2: Effect of Linear Back-off Algorithm on Response time; 4096 byte Block

was good with 4096 byte blocks, we ran the rest of the experiments using this block size for both reads and writes.

We expected the best performance to be somewhere between the two extremes of 1024 and 8192 bytes. This is because at large block sizes, the number of fragments is more, which increases the chance of one of the fragments getting caught in the bad-state. With small block-sizes, more packets need be sent causing overhead due to an increase in hand-shake (request and acknowledgment messages). The compromise between these two opposing effects shows up clearly only in the case of the higher error-rate.

4.2 Effect of Linear Back-off Algorithm

Our back-off algorithm is a hybrid of linear and exponential algorithms : for the first few timeouts, the next timeout value is calculated by adding a small constant to the previous timeout value (linear back-off). The next few timeout values are calculated according to an

Bad Period	**Read**			**Write**		
(milli-seconds)	Min	*Avg*	Max	Min	*Avg*	Max
40	35.1	*38.3*	41.8	14.2	*16.9*	20.0
80	31.1	*31.7*	33.5	15.0	*15.3*	15.6

Table 3: Effect of Link-level Retransmissions on Response time; 4096 byte Block

exponential back-off algorithm, upper-bounded by a maximum timeout value. This is done so that in the case where the wired link is congested, or the server is down, frequent client retrials due to pure linear back-off, do not contribute additional un-necessary traffic. Thus the over-all effect is that of frequent retrials initially (linear timeout values) which get more spaced out (exponential timeout values) in the event that no replies are received.

The results obtained by using a linear back-off instead of the exponential back-off are presented in Table 2. In the 40 milli-second case, there is not much improvement in response time. However, for the 80 milli-second case, there is a 28% reduction in the read response time and 48% reduction in the write response time. In all the results we report, transfers which took more than 300 seconds were terminated and rerun. An observation which does not show up in the tables presented is that such terminations were not required when using linear back-off as the NFS client was able to get (and put) packets across within a reasonable time. Hence, using a linear back-off algorithm serves to bound the time required for a read or write operation to complete besides improving performance at the higher error rate.

4.3 Effect of Link-level Retransmissions

We simulated link-level retransmissions by holding the packets which arrive at the device-driver (both from IP and the device) during the bad-period and pushing them across when the switch from bad-state to good-state occurred. Even though the use of this technique presents only a best-case for the use of link-level retransmissions, the gains obtained seem promising enough to warrant a complete retransmission protocol implementation.

The results obtained by using link-level retransmissions (without enabling the linear back-off in the NFS client) are presented in Table 3. An improvement of upto 53% is observed for reads and upto 66% for writes with the use of link-level retransmissions.

It should be mentioned that since we simulated the use link-level retransmissions over an otherwise real system, buffer overflows as mentioned earlier in this section (Figure 3) still take place. An implementation of link-level retransmissions would solve this problem, leading to substantial performance improvement at least for the reads.

4.4 Linear Back-off + Link-level Retransmissions

We were apprehensive that the use of linear back-off and link-level retransmissions simultaneously would interfere with each other. The results obtained with both these mechanisms enabled are presented in Table 4. The average values gleaned from the Tables 1 to 4 are summarized in Table 5 for reads and Table 6 for writes. Neither the reads, nor the writes, show any significant interference between client linear back-off and link-level retransmissions

Bad Period (milli-seconds)	**Read**			**Write**		
	Min	*Avg*	Max	Min	*Avg*	Max
40	34.0	*37.0*	42.2	15.3	*16.7*	18.0
80	31.3	*31.9*	33.2	14.3	*14.8*	15.1

Table 4: Effect of Linear Back-off + Link-level Retransmissions; 4096 byte Block

Bad Period (milli-seconds)	**No Changes**	**Linear**	**Retransmit**	**Linear + Retransmit**
40	49.2	48.0	38.3	37.0
80	67.7	48.7	31.7	31.9

Table 5: Average Response times for Reads; 4096 byte Block

Bad Period (milli-seconds)	**No Changes**	**Linear**	**Retransmit**	**Linear + Retransmit**
40	24.7	24.6	16.9	16.7
80	45.5	24.4	15.3	14.8

Table 6: Average Response times for Writes; 4096 byte Block

Operation	**Min**	**Max**	**Avg**
Read	30.1	32.2	*30.7*
Write	13.9	14.9	*14.2*

Table 7: Response times with Zero errors; 4096 byte Block

Bad Period	**No Changes**			**Linear + Retransmit**		
(milli-seconds)	Min	*Avg*	Max	Min	*Avg*	Max
40	50.0	*56.1*	64.5	30.6	*33.4*	36.9
80	52.3	*61.9*	70.4	32.9	*35.2*	38.5

Table 8: Scalability : Reads with 4096 byte block

[8]. The link-level retransmissions are primarily responsible for the improved performance in the average case, whereas linear back-off serves to bound the worst case performance.

We ran some additional experiments to obtain the performance assuming zero errors. These results are shown in Table 7. From Tables 5, 6 and 7 it is clear that the modifications we suggest are effective and performance approaches that of the zero error case (upto within 5%). Read response time reduces by upto 53% with a corresponding increase in throughput of upto 112%[9]. Similarly, write response time reduces by upto 67% with a throughput increase of upto 208%[10].

4.5 Scalability

Since it was unclear how a more aggressive NFS client with link-level retransmissions would compete with other, similar clients, it was necessary to study the scalability of our system. We analyzed the performance of our system with two mobile-hosts (shivalik and narmada) as NFS clients. The response times for shivalik are presented in Tables 8 and 9.

[8]In Tables 5 and 6, *No Changes* refers to only the error model installed into the device driver; *Linear* refers to linear back-off in the NFS client and *Retransmit* refers to link-level retransmissions.

[9]Bad-period 40 milli-seconds - from 20,325 bytes/sec to 27,027 bytes/sec; Bad-period 80 milli-seconds - from 14,471 bytes/sec to 31,347 bytes/sec

[10]Bad-period 40 milli-seconds - from 40,486 bytes/sec to 59,880 bytes/sec; Bad-period 80 milli-seconds - from 21,978 bytes/sec to 67,568 bytes/sec

Bad Period	**No Changes**			**Linear + Retransmit**		
(milli-seconds)	Min	*Avg*	Max	Min	*Avg*	Max
40	35.9	*46.1*	51.6	20.0	*25.3*	34.4
80	31.7	*45.0*	68.1	20.5	*25.5*	30.2

Table 9: Scalability : Writes with 4096 byte block

Results for narmada were similar and are hence not presented. Our results show that even with multiple mobile-hosts, performance improved substantially. With the changes that we suggest, read response times reduced by upto 43% (throughput increase of upto 76%) and write response times by upto 45% (throughput increase of upto 82%), as compared to the two mobile-host system without any of our enhancements.

4.6 Summary of Results

Our experiments show that 4096 byte read and write blocks give good over-all performance. With higher error rates, the optimal block size is likely to be even smaller, but since higher error rates are less likely to occur, 4096 byte blocks are a reasonable size for obtaining good performance. We tried some additional experiments with all our modifications and 8192 byte blocks. The performance with 4096 byte blocks was in general, better.

Linear back-off was useful as it bounded the maximum transfer times. With exponential back-off, we observed several cases where response time exceeded 5 minutes. Replacing exponential back-off by linear back-off almost completely eliminated such behavior. At higher error rates, linear back-off improves performance considerably. This is because at higher error rates, packet losses are more frequent which increases the probability of a block losing one of its fragments, leading to substantial difference in the timeout values calculated by the two algorithms.

Link-level retransmissions reduced response times considerably. Although our simulation of link-level retransmissions presents only a best case for link-level retransmissions, the performance improvement was significant enough to warrant a full-scale implementation with perhaps a reservation protocol and channel state dependent packet scheduling built in [BBKT96]. A real implementation of link-level retransmissions would in addition solve the buffer overflow problem, leading to further improvement in response time.

The response times we obtained were within 5% of the zero error case. The results we present show that there was no significant interference between link-level retransmissions and the NFS clients retry mechanisms. Further the performance improvements scaled well with multiple mobile-hosts. Hence, our study suggests that link-level retransmissions along with an aggressive NFS client would be a desirable combination on mobile-hosts.

5 Future Work

Based on our experiments, we plan to build a reservation protocol (RTS-CTS-DATA-ACK) with link level retransmissions and a channel state dependent scheduling policy [BBKT96]. This would solve the buffer overflow problem (which is important because there is always going to be a mismatch in the capabilities of and load on machines used as servers, base-stations and mobile-hosts). Link-level solutions seem to be promising because they offer a common solution for higher level protocols and applications. However [DCY93] suggests that link-level retransmissions have an averse effect on the performance of TCP. We intend to investigate this matter further, as we believe it that is possible to come up with an elegant link-level solution which improves performance without significantly interfering with the higher layers.

Using disk-caching (which has been proven as an effective technique in the Coda [SK92], Odyssey [SNKP94] and Little Work [HHRB92] projects) for read-only files [ZD93], is also being considered.

Besides, the linear (followed by exponential) back-off described in Section 4.3, we need to tune the NFS client to behave intelligently during cell-switches (hand-offs). The information indicating an imminent cell-switch is usually available to the wireless device driver. An appropriate, signal (or some other mechanism) needs to be devised to inform the NFS client (and other applications) of the impending cell-switch. The client can appropriately adjust its timeout values preparing for the packet losses which are bound to occur. We are working towards providing such a mechanism.

6 Conclusions

Our experiments validate our approach of tuning the NFS client and using link-level retransmissions to improve performance. The data that we gathered suggests that 4096 bytes (4KB) is a reasonable block size for both reads and writes. Linear back-off (as opposed to exponential back-off) serves to bound the amount of time taken by a file transfer, even though by itself, this technique may not improve performance for low error rates. Link-level retransmissions on the other hand seem to greatly enhance the average case performance, leading to improvement in throughput of upto 112% for reads and 208% for writes. The performance obtained is within 5% of the "zero error on the wireless link" case.

Acknowledgements

We thank Pravin Bhagwat for many related discussions and Alexander Sarris for help with setting up the testbed. We also thank Pete Keleher's advanced systems class for being a critical audience to this work and Shamik Sharma, Ibrahim Korpeoglu and Sambit Sahu for helping improve the manuscript at various stages of its conception.

References

[BB95a] A. Bakre and B. R. Badrinath. M-RPC: A Remote Procedure Call Service for Mobile Clients. In *First ACM International Conference on Mobile Computing and Networking*, November 1995.

[BB95b] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Fifteenth International Conference on Distributed Computing Systems*, May 1995.

[BBKT96] P. Bhagwat, P. Bhattacharya, A. Krishna, and S.K. Tripathi. Using Channel State Dependent Packet Scheduling to improve throughput over Wireless LANs. In *IEEE INFOCOM*, 1996. Pre-print.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP Performance over Wireless Networks. In *First ACM International Conference on Mobile Computing and Networking*, November 1995.

[BSK95] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, December 1995. Pre-print.

[CI94] R. Caceres and L. Iftode. The Effects of Mobility on Reliable Transport Protocols. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, June 1994.

[DCY93] A. DeSimone, M.C. Chuah, and O.C. Yue. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *IEEE GLOBECOM*, pages 542 – 549, 1993.

[Gro88] Network Working Group. RPC : Remote Procedure Call Protocol Specification, 1988. RFC 1057.

[HHRB92] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. The Little Work Project. In *Third IEEE Workshop on Workstation Operating Systems*, 1992.

[Jus89] C. Juszczak. Improving the Performance and Correctness of an NFS Server. In *USENIX Conference Proceedings*, pages 53 – 63, January 1989.

[Jus94] C. Juszczak. Improving the Write Performance of an NFS Server. In *USENIX Conference Proceedings*, January 1994.

[MJ93] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture . In *USENIX Winter Conference*, January 1993.

[Nov89] B. Novicki. Transport Issues in the Network File System. *ACM Computer Communications Review*, 19(2):16 – 20, 1989.

[PAL$^+$95] S. Paul, E. Ayanoglu, T.F. LaPorta, K.H. Chen, K.K. Sabnani, and R.D. Gitlin. An Asymmetric Link-layer Protocol for Digital Cellular Communications. In *IEEE INFOCOM*, 1995.

[PJS$^+$94] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS Version 3 Design and Implementation. In *USENIX Summer Conference*, 1994.

[RT95] C.D. Rais and S.K. Tripathi. Measuring NFS Performance over Wireless Networks. Technical Report CS-TR-3582, University of Maryland, College Park, 1995.

[SGK$^+$85] R. Sandberg, D. Goldberg, S. Kleinman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network File System. In *USENIX Summer Conference*, 1985.

[SK92] M. Satyanaraynan and J.J. Kistler. Disconnected Operation in the Coda File System. *ACM TOCS*, 10(1):3 – 25, 1992.

[SNKP94] M. Satyanraynan, B. Noble, P. Kumar, and M. Price. Application-Aware Adaptation for Mobile Computing. In *6th ACM SIGOPS European Workshop*, September 1994.

[WLS$^+$85] D. Walsh, B. Lyon, G. Sager, J.M. Chang, D. Goldberg, T. Lyon, R. Sandberg, and P. Weis. Overview of the Sun Network File System. In *USENIX Winter Conference*, 1985.

[YB94] R. Yavatkar and N. Bhagwat. Improving End-to-End Performance of TCP over Mobile Internetworks. In *Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.

[ZD93] E. Zadok and D. Duchamp. Discovery and Hot Replacement of Read-Only File Systems, with Application to Mobile Computing. In *USENIX Summer Conference*, pages 69 – 85, 1993.