Improving NFS Performance over Wireless Links*

Rohit Dube, Cynthia D. Rais and Satish K. Tripathi
Mobile Computing and Multimedia Laboratory
Department of Computer Science
University of Maryland
College Park, MD 20742 U.S.A.
{*rohit, cldavis, tripathi*}*@cs.umd.edu*

December 22, 1996

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 continued remote file access via NFS. However, low bandwidth and high error rates degrade performance on mobile hosts using wireless links, hindering the use of NFS. We conducted experiments to study the behavior of NFS in a wireless testbed. Based on these experiments, we incorporated modifications into the mobile NFS client. This paper presents two mechanisms which improve NFS performance over wireless links: an aggressive NFS client and link-level retransmissions. Our experiments show that these mechanisms improve response time by up to 62%, 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. Unfortunately, since most applications and reliable transport protocols have been optimized for wired networks and static hosts, they suffer from poor performance when used on wireless systems. The performance over wireless links is currently limited by low bandwidths, high error rates, temporary disconnections, and high latencies. Protocols and applications must be adapted to accommodate 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 is widely used on wired LANs to provide a mechanism for remote and distributed file access [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 errors and higher error rates prevalent over wireless media pose performance problems for mobile applications which use NFS-mounted files.

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

NFS should be usable in in-building wireless networks even in the presence of interference or loss. We limit our scope to these wireless LAN networks, (which have bandwidths on the order of 1 Mbps). NFS is designed for faster physical networks (on the order of 10 Mbps) which exhibit rare random errors. Therefore, packet losses are attributed to either network congestion or a server failure. NFS clients can back-off and retry a request after waiting for some predetermined time period. On wireless links, packet losses are usually due to burst errors rather than network congestion or server failures. These burst periods are on the order of a hundred milliseconds[BBKT96]. In response to such losses, NFS clients back-off to unnecessarily long wait periods, leading to severe performance degradation.

NFS improvements are needed, but they should not adversely affect the existing wired infrastructure: the static hosts and servers. After studying the behavior of NFS in our wireless testbed, we implement several mechanisms to improve NFS performance and measure the effectiveness of these mechanisms under a wide range of error behaviors. Based on these studies, we use smaller block sizes and a hybrid linear back-off, which result in a more aggressive NFS client. We also analyze link-level retransmissions as a mechanism to improve performance. As the experimental results show, these mechanisms dramatically improve NFS performance and reduce response times by up to 62% without requiring any modifications at the NFS server. These techniques achieve comparable performance gains with several mobile hosts.

The changes that we incorporate are simple and achieve a substantial performance improvement, making NFS over wireless links a feasible option. The changes to the wireless NFS clients do not require any changes at the server or static NFS clients. The simplicity of our solution in a highly complex protocol stack is crucial. The changes we suggest are general and can be applied to other data and file transfer protocols.

The rest of this paper is organized as follows. Section 2 discusses background and related work, and section 3 presents our approach for improving NFS performance. Section 4 describes the testbed and the error model used for our experiments, and section 5 presents the results of these experiments. Finally, section 6 summarizes this paper and presents some ideas for future investigation.

2 Background and Related Work

NFS provides transparent remote file access in heterogeneous networks. Since NFS is usually implemented over UDP, it must provide its own reliability mechanisms. Versions of NFS over TCP use TCP reliability mechanisms, but are non-standard and potentially incompatible with versions of NFS over UDP. For ease of recovery and robustness, the NFS servers are stateless and maintain no client information. Therefore, the NFS clients must initiate all communication with the NFS server by sending requests for service. The server responds with an acknowledgment or the requested data. If the client does not receive a server acknowledgment, the client is responsible for retrying the requests using its own timeout and retransmission polices.

Previous research in this area has concentrated on improving NFS performance over wired LANs, on TCP/IP performance over wireless media, and on developing new file systems for mobile computers. These studies do not discuss NFS performance over wireless links, but their results provide useful insights to the specific problem being considered in this paper. The relation of these studies to our research is discussed in the following paragraphs.

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 but would not significantly help the performance of NFS in a wireless environment where the bottleneck is the mobile NFS client and the wireless link transmission. Some of the traditional improvements, such as the use of larger block sizes, actually have a detrimental effect on lossy wireless links and will degrade NFS performance at mobile hosts.

Disk-caching is implemented in the Andrew File System (AFS) [Sat90] to improve remote file access performance. Coda [KS92], its later version Odyssey [SNKP94], and Little Work [HHRB92] are successors of AFS and all use disk-caching to support disconnected and intermittent operation. Although prefetching and disk-caching are useful on wireless clients to decrease the time the user spends waiting for file transfers, they do not solve the problem of poor throughput and low wireless link utilization in the cases when a file must be obtained from a server.

A study of AFS in a very low bandwidth network (9.6 Kbps) has shown the importance of using dynamically adjustable parameters to quickly adjust to changes in round-trip times and to losses [BHH94]. In the 1 Mbps LAN environment which we consider, this use of dynamic parameters is less critical, although it offers additional improvement at the cost of added complexity.

We considered using an NFS implementation over TCP instead of UDP. Unfortunately, NFS over TCP implementations are non-standard: the 4.4BSD TCP implementations do not work with non-BSD servers. To insure that mobile hosts can inter-operate with all NFS servers, we use NFS over UDP. Thus TCP improvements can not be directly used for improving NFS.

TCP Vegas [BOP94] improves throughput by using available bandwidth more efficiently. It uses the system clock to accurately calculate round-trip times and does not wait for course-grained timeouts. It also tries to avoid loss due to congestion by changing the congestion window based on the actual and expected throughput values. Our NFS approach also responds more quickly to losses, but uses a simpler method with non-dynamic parameters.

The **fast retransmission** approach [CI94] improves TCP performance by notifying the transport layer after mobile host motion as soon as the cell-switch hand-off is complete to avoid initiating congestion control policies. This approach addresses mobility but not the poor performance caused by burst errors on the wireless link.

In the TCP **split connection** approaches [YB94], [BB95a], 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 [BB95b] is a variation of the TCP split connection approaches which seeks to improve performance by separating the connection at the RPC level [Gro88]. These approaches have the disadvantage of high buffer requirements, a complex migration algorithm, consistency problems, and increased load at the base-station. File systems have high consistency requirements and would be vulnerable under some of the split connection approaches. Although NFS uses RPC, the M-RPC approach would require modifications to the RPC and NFS code on both the client and the server.

In the **snoop** approach [BSAK95], an agent at the base station monitors all TCP packets and caches unacknowledged segments. For the cached packets, the snoop agent suppresses

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

any duplicate acknowledgments, which indicate packet loss, and retransmits the lost packets. This approach preserves TCP semantics, while improving performance, but has a larger buffer requirement and requires complex changes.

Link-level retransmissions have been proposed to improve performance over wireless links [DCY93], [PAL+95], [BBKT96]. In addition, [BDSZ94] presents and simulates a media access protocol which provides reliability on the wireless links. These mechanisms aim at concealing losses on the wireless link from the higher layers and are discussed further in the following section.

3 Solution Approach

In the interests of inter-operability with static hosts, any attempt towards enhancing performance should be limited to modifications made to machines supporting the wireless link. NFS servers cannot be modified. In accordance with this requirement, our solution approach requires modifications only to the mobile NFS client and the wireless device drivers.

In studying NFS over wireless links, we noted several factors that cause poor performance. First, NFS uses large block sizes (usually 8192 bytes) to decrease the overhead involved with requesting and sending each segment of data. These blocks must be fragmented at the IP layer before being sent on to the physical link, which has a Maximum Transmission Unit (MTU) of 1500 bytes. A single lost fragment causes retransmission of the entire block. For the error prone wireless links, this leads to many retransmissions which can be avoided by using smaller block sizes.

The second NFS feature which causes poor performance is the exponential back-off algorithm used by NFS clients. This occurs when the block requested by the client is lost or delayed due to network congestion or server overload. Exponentially increasing timeout 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 burst errors on the wireless link cause most losses. Burst errors are on the order of hundred milliseconds, which is much shorter than the typical server failure or congestion period. Exponential back-off constitutes an over-reaction. The mobile NFS client would achieve better performance by using linear back-off before reverting to an exponential algorithm (in the case of repeated losses). The first few losses will be retransmitted after a short linearly increasing timeout period. Subsequent losses will have exponentially increasing timeout values. This is in some sense the inverse of the congestion control mechanisms proposed in [Jac88].

In addition to improvements at the NFS client, optimizations can be made at the device driver by using link-level retransmissions. Many wireless device drivers do not implement a good retransmission policy or a reservation protocol to guard against the frequent physical layer errors. This leads to bad link utilization and triggers detrimental behavior (such as unnecessary back-off) in the higher level protocols or applications involved. Using link-level retransmissions ([DCY93], [PAL+95], [BBKT96]) for a more robust wireless link would shield the higher layers, preventing them from over-reacting to errors. Since we are constrained to work within the current hardware and protocol stacks, we must develop link-level reliability over current hardware which includes part of the MAC protocol rather than implementing an entire MAC protocol such as the one proposed in [BDSZ94].

We considered buffering IP packets at the base-station or running a TCP connection from the mobile to the base-station where UDP packets would be constructed and relayed to (and from) the NFS server. Conceptually, both these approaches and the snoop and split connection approaches mentioned in section 2 seek to build a more robust link between the mobile hosts and the base-station. The solution approach described in the previous paragraphs achieves the same goal, but unlike the snoop and split connection approaches, it de-couples the building of a robust link from the end-to-end reliability mechanisms which are higher layer protocol dependent.

Tightly coupled approaches in which information is exchanged between higher and lower layers of the protocol stack can be useful for improving performance, as shown in [BPSK96]. This approach should be used only if necessary, and the degree of interaction should be minimized. The TCP/IP protocol stack has inherited increasing complexities as it has evolved and been modified over the years. This complexity makes it difficult to incorporate and test changes to this protocol stack. Tightly coupled approaches keep track of the semantics of several protocol layers and require large buffer space and extensive modifications. Our goal is to achieve improved performance using simpler mechanisms which do not require tight coordination between the layers of the stack.

The performance problem can therefore be tackled in a simpler manner at two layers: the wireless link and the higher layer protocol or application. No complex mechanism involving the higher layer protocols need be built. The most that may be required of the higher layer protocol is a smart retry mechanism. This philosophy is fairly general and applies to other protocols and applications as well as NFS.

4 System Setup

We consider a micro or pico-cellular building network environment with base-stations in designated locations to handle traffic from mobile hosts. In this environment, the wireless burst errors have a very complex impact on the higher layers. The errors on the wireless link can be modeled, but due to the vertical dependency of the higher layers on the physical layer, it is not possible 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 control of the errors at the wireless link.

4.1 Testbed

As shown in the testbed diagram, in Figure 1, the IBM RS6000 acts as the NFS server, and an IBM RT-PC is configured as the base-station. Two mobile NFS clients, an IBM PS/2 and an IBM RT-PC, access the NFS server through the base-station using a 1 Mbps wireless link. The DEC-ALPHA monitors traffic on the Ethernet by running TCPDUMP [MJ93]. TCPDUMP reported zero packet loss in all tests.

Using this testbed, we study the performance of NFS reads and writes by measuring the *response times* obtained for file transfers to and from the local disks of the mobile hosts. In order to control the error rate, we place the wireless devices in close proximity, which minimizes uncontrolled errors on the wireless link. Instead of using physical errors on the link, an error model is introduced into the device driver of the mobile hosts as explained in the next section. This allows control over the error patterns while keeping the benefits of a real testbed, such as realistic bandwidth limitations, processing delays, and buffer or device

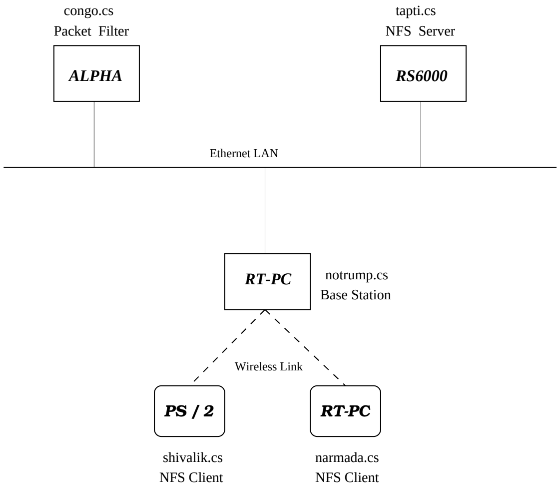

Figure 1: Testbed for NFS Performance Evaluation

limitations. Finally, to ensure consistent results, the Ethernet segment holding the static hosts is isolated from the rest of the building network during all experiments.

4.2 Error Model

Signal fading, multipath, and transmission interference cause burst errors on wireless links. Burst errors are sequences of corrupted or lost bits covering a period of up to a few hundred milliseconds, which degrade the quality of the low bandwidth wireless channels.

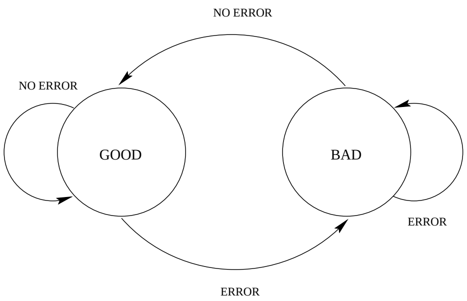

Figure 2: The 2-state Error Model

We use a two-state Markov error model, as shown in Figure 2, to represent the quality of a channel between the mobile host and the base-station. Instead of attempting to empirically model the channel, we capture the relative performance information using a simple, but non-trivial model that can easily be incorporated into the device driver for our experiments.

The two states in the Markov model represent error and error-free periods. During the *good* state bits are transmitted without errors. The *bad* state occurs when bits are lost or corrupted. Two-state models have been used by previous researchers to characterize the error on the wireless channel, [BBKT96], and as a starting point for an empirical model for wireless error behavior [NKNS96]. A two-state model is chosen over the three-state model discussed in other studies [SKKF93],[WM95], as the two-state model simplifies the model without losing information pertinent to this research. The third state would represent a *guard* state, which would occur when a single bit is corrupted. Then a move into the 'bad' state would occur when several consecutive bits have been corrupted. We combine this 'guard' state with the 'bad' state and allow time in the 'bad' state to be arbitrarily small.

A temporal model for the error behavior of the wireless channel is desired to capture the temporal nature of burst errors. The quality of the channel (good or bad) in such a model depends only on the current time instant. This differs from the model used in [BSAK95] which non-stochastically corrupts a small percentage of the bits in transit over the wireless channel.

We implement a temporal model by using a distribution for the periods spent in the good and bad states. We perform experiments with various distributions (including uniform and deterministic) and with a wide range of error rates. Only the results for *uniform distributions* with a good period mean of one second and bad period means between 0 and 160 milliseconds are presented. Discrete functions with ranges of 200 for the bad period and 2000 for the good period are used. We choose the error range based on research which has measured fading in bursts of 10 to 100 ms in an office building environment [HMVT94],[BBKT96]. The model is implemented using the kernel TIMEOUT mechanism, with a granularity of twenty milliseconds. We incorporate this error model into the input and output routines of the mobile hosts' device drivers and locate the entire model at the client to maintain control over synchronization of the transmit and receive channels.

5 Experimental Results

Experiments are conducted by reading and writing NFS-mounted files from the mobile hosts using mean bad periods in the range 0 to 160 milliseconds. Since various file sizes yield similar results, only the response times for experiments with 1 MB files are reported here. Each experiment is run 10 times[2], and the graphs in the following sections present the average response times (in seconds). We first present detailed results for the single mobile host case and then discuss how these results extend in the presence of several mobile hosts.

5.1 Effect of Block Size

The effect of NFS block size on the response time of reads and writes is shown in Figures 3 and 4. The curves in both the figures exhibit either a knee or a low at 4096 bytes. The best performance is expected between the two extremes of 1024 and 8192 bytes. Using large block sizes increases the number of fragments, which increases the probability of a fragment being lost during the bad state. A single fragment loss requires retransmission of the entire block and can have a detrimental impact on performance [KM87], [RF95]. Alternatively, small block sizes cause an increase in request and acknowledgment messages. This hand-shaking increases the transmission latency.

The trade-off between these two opposing effects can be seen in Figures 3 and 4. It is especially clear at higher error rates where the errors exaggerate the opposing effects. The combination of higher error rates and the 8192 block size results in many more losses and block retransmissions for each fragment loss, which increases the response time substantially. When using a 1024 block size at high error rates, data transfer is very slow due to the combined

[2] We also conducted experiments using more than 10 runs. The confidence intervals were not significantly improved for larger numbers of runs.

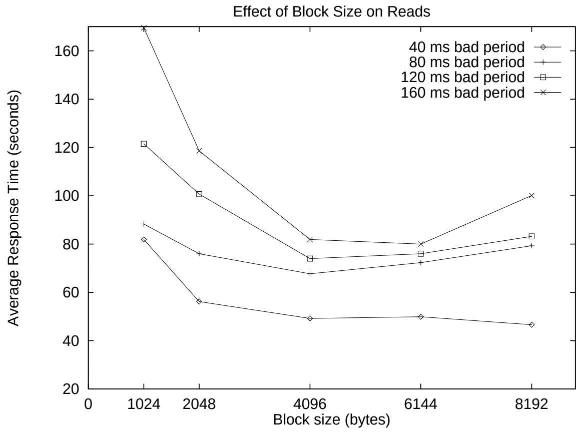

Figure 3: Reads perform best with 4096 byte blocks

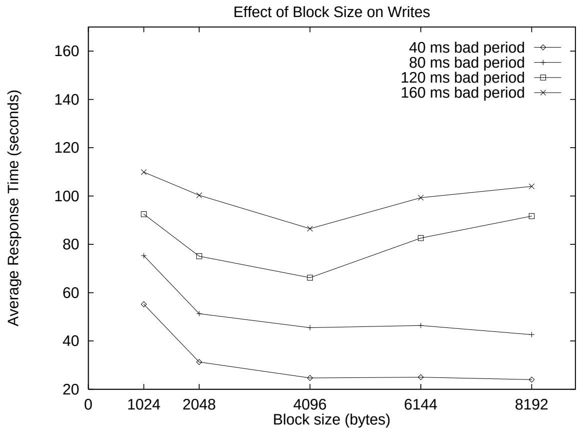

Figure 4: Writes perform best with 4096 byte blocks

cost of increased hand-shaking overhead and frequent losses. The unexpected result of faster write response times is due to a buffer overflow problem which we observe at the wireless device drivers and cards. This overflow is more pronounced for the reads which leads to the anomaly of faster response times for writes.

Since 4096 byte blocks provide the best NFS performance, the rest of the experiments use this block size for both reads and writes. Using the 4096 block size, experiments are conducted for various error rates. Figures 5 and 6 show these results in the *No Changes* curves. These curves display the poor performance of NFS prior to our Hybrid Linear and Retransmission modifications, which we present in the following subsections. Even with the improved block size, the performance is still limited by the slow response to losses. We also ran experiments to measure the performance with zero errors on the wireless links. This appears in the *Zero Errors* line to provide a base-line for evaluating the effectiveness of our modifications.

The block size of 4096 bytes is optimal in our testbed environment, and the same approach can be used to determine optimal block sizes in other environments. The optimal block size is influenced by the prevailing error rates as well as the relative power of communicating machines. Since there is a significant performance difference between using the optimal and non-optimal block size, it is advantageous to use an optimal block size.

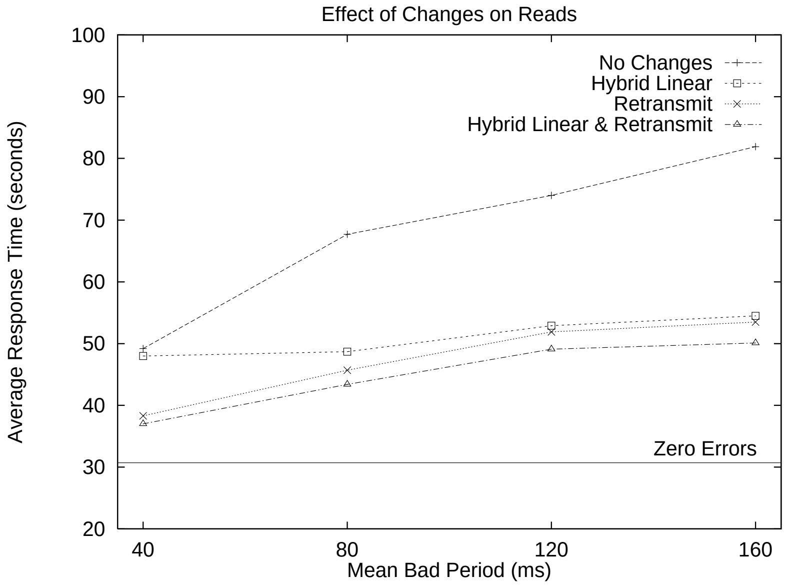

Figure 5: Reads show significant improvement with modifications

5.2 Effect of Hybrid Linear Back-off Algorithm

Our modified NFS client uses a hybrid of linear and exponential algorithms to calculate back-off intervals in the event of a timeout. For the first few timeouts, the client calculates the next timeout value by adding a small constant to the previous timeout value (linear back-off). The next few timeout values are calculated according to an exponential back-off algorithm, upper-bounded by a maximum timeout value, (*Max_Timeout*).

$$Timeoutperiod = max(Max\_Timeout, (Initial + min(a, R) * c) * 2^{max(0,R-a)})$$

Here c and *Initial* are small constants, R is the number of retransmissions of the current packet, and a equals the number of times linear back-off is used before going to exponential.

The results obtained by using such a hybrid back-off algorithm are shown in the *Hybrid Linear* curve in Figure 5 (reads) and in Figure 6 (writes). As compared to the *No Changes* curves, the use of hybrid linear back-off constitutes a significant improvement. This improvement is more prominent at higher error rates because more frequent packet losses increase the probability of a block losing its fragments repeatedly, leading to a substantial difference in the timeout values calculated by the hybrid linear and the pure exponential algorithms. For repeated losses, the quicker retransmissions of the hybrid linear back-off improve performance by eliminating the long idle periods which previously followed each loss.

In the reported results, transfers lasting more than 300 seconds are terminated and rerun. Such terminations are not required when using hybrid linear back-off since the aggressive NFS client can receive and send packets within a reasonable time. Consequently, using a hybrid linear back-off algorithm serves to bound the time required for a read or write operation to complete in addition to improving performance.

On a LAN consisting of both wireless and wired segments packet losses and delays are primarily due to noise on the wireless channel rather than congestion on the wire or heavily loaded servers. The method for dealing with losses and congestion depends on the location of the bottleneck. When errors and low bandwidth cause the wireless link to be the bottleneck,

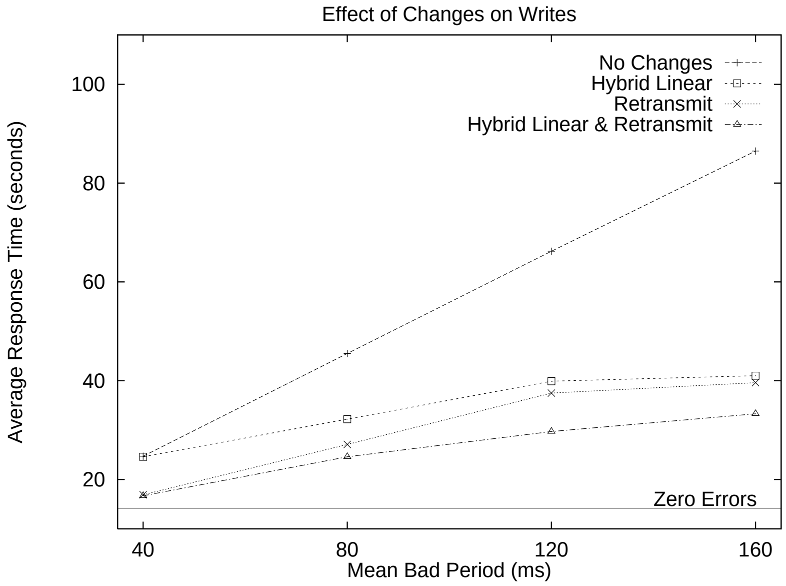

Figure 6: Writes show significant improvement with modifications

these losses should be assumed to be the norm rather than congestion losses. Since the minimum NFS timeout is around 700 milliseconds, well over the typical length of a noise burst, exponential back-off will not react appropriately to a packet loss. In fact, repeated fragment loss may force the client to back-off to a large wait period, severely degrading performance. Hybrid linear back-off is an optimistic mechanism for recovering quickly from losses on the noisy wireless link. In the occasional case where congestion on the wired or wireless network or at the server causes the timeout, pure linear back-off could cause congestion collapse. To prevent this, the hybrid linear algorithm switches from linear back-off to exponential after three re-trials. In an in-building environment where the number of mobile hosts per base-station is limited, this restricts the number of extra packets transmitted during congestion, preventing congestion due to traffic generated for or by the mobile NFS clients on both wired and wireless links.

5.3 Effect of Link-level Retransmissions

Link-level retransmissions are a mechanism by which the receiver and the sender try to reliably exchange data by using *control* messages before and after transmitting data according to a well-defined protocol. The sender and receiver retransmit data and other messages if a loss interrupts the protocol. This limits the needed retransmissions to the wireless link and improves throughput since the retransmissions occur much more quickly than at NFS. The link-layer retransmits lost packets for a certain number of times before giving up. This retry limit is necessary to minimize the round-trip time variability and the impact on other queued data transmissions and higher layer reliability mechanisms. Therefore, NFS must ultimately insure reliability for all attempted data transfers. A link-layer retransmission policy which is fair to multiple channels is discussed in [BBKT96].

We emulate link-level retransmissions by holding incoming and outgoing packets at the device-driver during the bad period and transmitting them when the switch from bad state to good state occurs.

The results obtained by using link-level retransmissions (disabling the hybrid linear back-off in the NFS client) are shown by the *Retransmit* curves in Figures 5 and 6. The use of retransmission achieves considerable performance improvement over *No Changes* since lost packets are successfully retransmitted after the burst period ends instead of waiting for a course grained timeout. Improvement over the *Hybrid Linear* technique is substantial only at lower error rates. This is because longer burst periods cause the link-level mechanisms to wait for intervals which approach the length of NFS back-off periods.

Even though the use of this technique presents only a best-case for the link-level retransmission, the gains obtained are promising enough to warrant a complete retransmission protocol implementation.

5.4 Effect of Hybrid Linear Back-off and Link-level Retransmissions

To ensure that hybrid linear back-off and link-level retransmissions do not interfere with each other when used together, we perform experiments with both these mechanisms enabled. Figures 5 and 6 show the results in the *Hybrid Linear & Retransmit* curves. No interference between the two mechanisms appears as these curves lie below the *Hybrid Linear* and *Retransmit* curves. The improvements from the two techniques do not add up completely. This is because in most cases, link-level retransmissions are able to deliver packets before the NFS timer expires. In the cases with longer error bursts, link-level retransmissions may not be able to react in time, causing the NFS client to re-send a request. Thus link-level retransmissions perform most of the work, with the NFS client taking over if the link-level mechanisms recover slowly. One might then suggest an increased minimum NFS client timeout to allow the link more time to get packets across by retransmissions. Although this works for wireless burst errors, it will cause larger latencies when problems occur on the wired side of the network.

The performance of the system with all the modifications installed approaches closely the *Zero Error* case at low error rates and provides substantial improvement over the *No Changes* curve at higher error rates.

5.5 Performance with Multiple Mobile Hosts

To study the interaction of aggressive NFS mobile clients, we analyze the performance with two mobile hosts as NFS clients (shivalik and narmada). Figures 7 and 8 present the response times for shivalik when two modified NFS mobile clients were accessing the NFS server. For both reads and writes, the modified system (the *All Changes* curves) performs much better than the base system (the *No Changes* curves) and comes close to the performance of the modified system with only a single host, which was shown in the *Hybrid Linear & Retransmit* curves in figures 5 and 6. With two mobile clients, congestion on the wireless link is not a problem, and each host can achieve substantial improvement with these retransmission techniques. As the number of mobile clients in one wireless cell increases, congestion will eventually become more of a problem than losses due to errors on the wireless link. Experiments in a large testbed are needed to determine how many mobile clients can be supported without causing congestion.

In these experiments, this NFS system is shown to be stable as well as effective. More experiments need to be performed in a larger testbed to confirm the scalability of this method. We expect our system to perform well in micro and pico-cellular environments where the number of mobile hosts is not expected to be large and where the NFS server is within

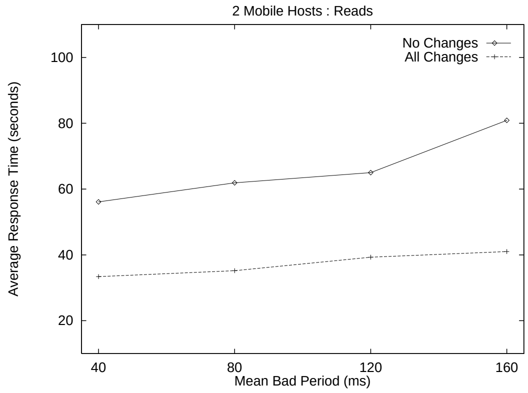

Figure 7: Reads scale well with multiple clients

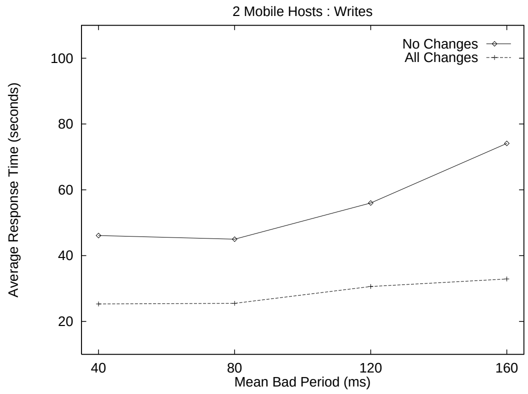

Figure 8: Writes scale well with multiple clients

several wired hops of the wireless subnet. The extent of scalability depends greatly on the implementation of the link-level retransmissions. With a lightweight mechanism and an efficient implementation, the wireless link would appear to be robust to applications and higher layer protocols. This would decrease higher layer retransmissions, thereby improving both performance and scalability. Section 6 discusses this further.

5.6 Summary of Results

The data collected suggests that 4096 bytes is a reasonable NFS block size for both reads and writes. The use of hybrid linear and exponential back-off in the NFS client improves the worst case performance and also the average performance at higher error rates. At the same time, the hybrid linear back-off bounds the maximum transfer times.

Link-level retransmissions reduce response times considerably and warrant a full implementation and more detailed study. The results show that there is no significant interference between link-level retransmissions and the NFS clients aggressive retry mechanisms. In addition, link-level retransmissions will solve the buffer overflow problem which we observe at the wireless device drivers and cards. This overflow is more pronounced for the reads which leads to the anomaly of faster response times for writes.

Link-level retransmissions alone are not sufficient to achieve the best worst case NFS performance since they do not guarantee complete reliability. The higher layer must initiate retransmission in cases of very long bad periods when link-level retransmissions have failed, and they should recognize cases where the link is completely disconnected and take appropriate action. The higher layer also must act in ways that minimize interference between application and protocol streams to achieve good performance since the link-level has no knowledge of this interaction. Thus the changes to the NFS client are needed to complement link-level retransmissions.

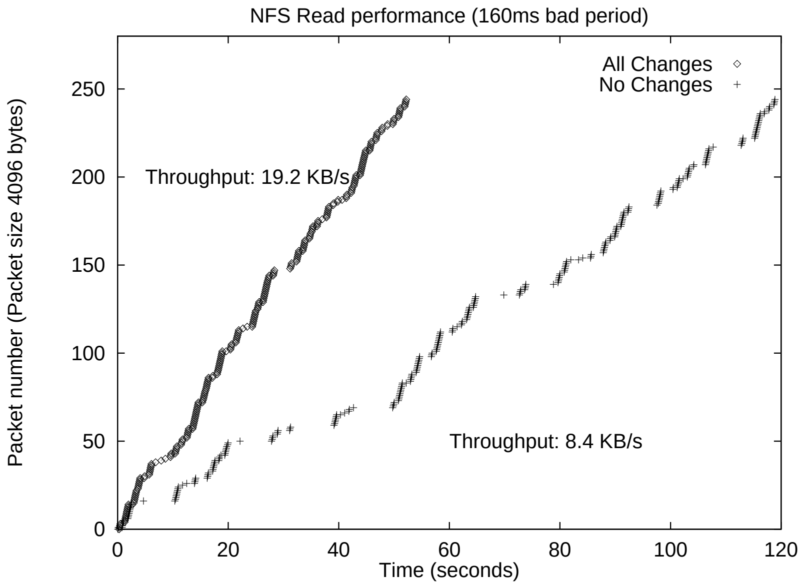

Figure 9: Fewer NFS retransmissions with the modifications

The output of TCPDUMP in Figure 9 illustrates the performance difference with and without our modifications. With no changes, idle periods follow most losses and occasional sequential losses cause even longer idle periods due to exponential back-off. These long idle periods degrade the performance to 8.4 KB/s. When we ran the mobile client without modifications, the same read call was performed with a throughput of 19.2 KB/s. These changes more than doubled the throughput by eliminating the idle periods after each loss.

Using the aggressive NFS client and link-level retransmissions improves response times by up to 62% and brings them within 5% of the zero error case in several experimental runs (with 40 millisecond bad period). An additional benefit of these techniques is consistency of response times. Calculation of the 95% confidence intervals for the data collected indicates that the intervals vary widely for high error rates and no modifications. The modifications eliminate this high variability, and the resulting confidence intervals are consistently small.

The performance improvements are maintained with two mobile hosts and work well through a range of error rates. The results are expected to scale well with hosts in a micro or pico-cellular environment. As our techniques do not require modifications to the NFS server, static hosts using NFS are not affected. The performance improvements and this compatibility strongly recommend the techniques we propose.

6 Conclusions and Future Work

NFS performance over wireless links is dramatically improved by using an aggressive NFS client combined with link-level retransmissions, as our experimental results demonstrate. Our approach improves response times by up to 62% (corresponding to a 160% increase in throughput) and brings performance to within 5% of the "zero error on the wireless link" case. This performance improvement makes the use of NFS over wireless LANs a feasible option.

NFS is widely used on traditional LANs and is important in heterogeneous networks to maintain remote file access as users migrate to mobile and wireless platforms. Our philosophy focuses on incorporating simple changes which do not require tight interaction between protocol layers. The mechanisms we use provide substantial performance improvement while maintaining compatibility with existing NFS servers. Although this paper discusses performance improvements only with respect to NFS, the techniques apply to application and protocol performance over wireless links in general. We plan to extend these solutions to improve the performance of other protocols over wireless links.

Our experiments show that an aggressive NFS client using a block size of 4096 bytes with a hybrid linear back-off performs well over the wireless link. In addition, the NFS client should be tuned 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 signaling mechanism needs to be devised to inform the NFS client (and other applications) of the impending cell-switch. The client can adjust its timeout values appropriately to prepare for the latencies which occur during hand-offs. We are working toward providing such a mechanism. This is a situation where more interaction is beneficial. If the link layer signals an impending cell switch NFS can better interpret losses and react appropriately. This improves the higher layers' performance at the cost of a small amount of interaction that does not require complex state information or large buffer space.

Together with link-level retransmissions, the aggressive client techniques deliver improved performance through a range of error rates. No significant interference between link-level retransmissions and the aggressive client is seen. This improved performance and the absence of interference present a strong case for both link layer techniques for enhancing average case performance and higher layer mechanisms to improve worst case performance over wireless links.

In order to implement a complete MAC protocol such as the one proposed in [BDSZ94], the hardware would have to be modified. As an alternative, we will add link-level retransmission functionality to the device driver. We plan to implement per destination MAC address queues at the base-station as in [BBKT96]. Each queue is scheduled in round-robin fashion. This improves fairness by eliminating head-of-the-line blocking which occurs in the output queue of the device driver if the channel for the current packet's destination is in a *bad* state. In addition, each queue would run a DATA-ACK protocol with the queue at the mobile host. Base-stations maintain one queue per destination MAC address, and mobiles maintain only one queue for the base-station. CSMA/CA is used to determine when data should be sent, but data transfer is considered complete only when an explicit acknowledgment (ACK) is received; (this ACK is different from the acknowledgment received by the controller on the wireless card). If a message loss causes this protocol to break, the lost message is retransmitted. The exact number of retransmissions for a given message will become clear only after further experimentation. The implementation of this protocol will achieve some of the gains of the MAC protocol presented in [BDSZ94] within the limitations of the current hardware. We believe that this mechanism has the potential of improving the performance of all higher layer protocols (not just NFS) without causing interference with end-to-end flow and congestion control mechanisms.

Improved scalability will be one result of the per destination scheduling. Our tests using several hosts show substantial performance improvements. Since only a few hosts per base-station are expected in a micro or pico-cellular environment, we expect our results to scale. Future experiments in a larger testbed with numerous mobile hosts will provide a better understanding of the scalability of our approach.

We also intend to combine hoarding, as used in Coda [KS92] and [TLAC95], with the predictive file caching techniques of [KL96] to fetch files which have a high probability of being accessed and transferred from the server onto the local disk. This can be implemented over the mechanisms discussed in this paper and would reduce remote file access latencies and user response time.

Acknowledgments

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, Marwan Krunz, Ibrahim Korpeoglu, Sambit Sahu, Sally Floyd and the anonymous reviewers for helping improve the manuscript at various stages of its conception.

References

[BB95a] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Fifteenth International Conference on Distributed Computing Systems*, May 1995.

[BB95b] 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.

[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*, March 1996. An extended version appears in ACM/Baltzer Journal on Wireless Networks, July 1996.

[BDSZ94] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A Media Access Protocol for Wireless LAN's. In *ACM SIGCOMM*, 1994.

[BHH94] D. Bachmann, P. Honeyman, and L.B. Huston. The Rx Hex. In *Proceedings of the 1st International Workshop on Services in Distributed and Networked Environments*, June 1994.

[BOP94] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *ACM SIGCOMM*, August 1994.

[BPSK96] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *ACM SIGCOMM*, August 1996.

[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. An extended version appears in ACM/Baltzer Journal on Wireless Networks, December 1995.

[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*, 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.

[HMVT94] H. Hashemi, M. McGuire, T. Vlasschaert, and D. Tholl. Measurements and Modeling of Temporal Variations of the Indoor Radio Propagation Channel. *IEEE Transactions on Vehicular Technology*, 43(3), August 1994.

[Jac88] V. Jacobson. Congestion Avoidance and Contol. In *ACM SIGCOMM*, 1988.

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

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

[KL96] T.M. Kroeger and D.D.E. Long. Predicting Future File-System Actions from Prior Events. In *USENIX Winter Technical Conference*, January 1996.

[KM87] C. Kent and J. Mogul. Fragmentation considered Harmful. In *ACM SIGCOMM*, August 1987.

[KS92] J.J. Kistler and M. Satyanaraynan. Disconnected Operation in the Coda File System. *ACM TOCS*, 10(1), 1992.

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

[NKNS96] G. Nguyen, R. Katz, B. Noble, and M. Satyanarayanan. A Trace-Based Approach for Modeling Wireless Channel Behavior. In *Proceedings of the Winter Simulation Conference*, December 1996.

[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.

[RF95] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13(4), May 1995.

[Sat90] M. Satyanaraynan. Scalable, Secure and Highly Available Distributed File Access. *IEEE Computer*, 23(5), 1990.

[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.

[SKKF93] T. Sato, M. Kawabe, T. Kato, and A. Fukasawa. Throughput Analysis Method for Hybrid ARG Schemes Over Burst Error Channels. *IEEE Transactions on Vehicular Technology*, 42(1), February 1993.

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

[TLAC95] C. Tait, H. Lei, S. Acharya, and H. Chang. Intelligent File Hoarding for Mobile Computers. In *First ACM International Conference on Mobile Computing and Networking*, November 1995.

[WM95] H.S. Wang and N. Moayeri. Finite State Markov Channel - A Useful Model for Radio Communication Channels. *IEEE Transactions on Vehicular Technology*, February 1995.

[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.