Network Working Group
Request for Comments: 5532
Category: Informational

T. Talpey
C. Juszczak

May 2009

Network File System (NFS) Remote Direct Memory Access (RDMA) Problem Statement

Status of This Memo

Copyright Notice

Abstract

This document addresses enabling the use of Remote Direct Memory Access (RDMA) by the Network File System (NFS) protocols. NFS implementations historically incur significant overhead due to data copies on end-host systems, as well as other processing overhead. This document explores the potential benefits of RDMA to these implementations and evaluates the reasons why RDMA is especially well-suited to NFS and network file protocols in general.

Table of Contents

1. Introduction

The Network File System (NFS) protocol (as described in [RFC1094], [RFC1813], and [RFC3530]) is one of several remote file access protocols used in the class of processing architecture sometimes called Network-Attached Storage (NAS).

Historically, remote file access has proven to be a convenient, cost-effective way to share information over a network, a concept proven over time by the popularity of the NFS protocol. However, there are issues in such a deployment.

As compared to a local (direct-attached) file access architecture, NFS removes the overhead of managing the local on-disk file system state and its metadata, but interposes at least a transport network and two network endpoints between an application process and the files it is accessing. To date, this trade-off has usually resulted in a net performance loss as a result of reduced bandwidth, increased application server CPU utilization, and other overheads.

Several classes of applications, including those directly supporting enterprise activities in high-performance domains such as database applications and shared clusters, have therefore encountered issues with moving to NFS architectures. While this has been due principally to the performance costs of NFS versus direct-attached files, other reasons are relevant, such as the lack of strong consistency guarantees being provided by NFS implementations.

Replication of local file access performance on NAS using traditional network protocol stacks has proven difficult, not because of protocol processing overheads, but because of data copy costs in the network

endpoints. This is especially true since host buses are now often the main bottleneck in NAS architectures [MOG03] [CHA+01].

The External Data Representation [RFC4506] employed beneath NFS and the Remote Procedure Call (RPC) [RFC5531] can add more data copies, exacerbating the problem.

Data copy-avoidance designs have not been widely adopted for a variety of reasons. [BRU99] points out that "many copy avoidance techniques for network I/O are not applicable or may even backfire if applied to file I/O". Other designs that eliminate unnecessary copies, such as [PAI+00], are incompatible with existing APIs and therefore force application changes.

In recent years, an effort to standardize a set of protocols for Remote Direct Memory Access (RDMA) over the standard Internet Protocol Suite has been chartered [RDDP]. A complete IP-based RDMA protocol suite is available in the published Standards Track specifications.

RDMA is a general solution to the problem of CPU overhead incurred due to data copies, primarily at the receiver. Substantial research has addressed this and has borne out the efficacy of the approach. An overview of this is the "Remote Direct Memory Access (RDMA) over IP Problem Statement" [RFC4297].

In addition to the per-byte savings of offloading data copies, RDMA-enabled NICs (RNICS) offload the underlying protocol layers as well (e.g., TCP), further reducing CPU overhead due to NAS processing.

1.1. Background

The RDDP Problem Statement [RFC4297] asserts:

> High costs associated with copying are an issue primarily for large scale systems ... with high bandwidth feeds, usually multiprocessors and clusters, that are adversely affected by copying overhead. Examples of such machines include all varieties of servers: database servers, storage servers, application servers for transaction processing, for e-commerce, and web serving, content distribution, video distribution, backups, data mining and decision support, and scientific computing.
>
> Note that such servers almost exclusively service many concurrent sessions (transport connections), which, in aggregate, are responsible for > 1 Gbits/s of communication. Nonetheless, the cost of copying overhead for a particular load is the same whether from few or many sessions.

Note that each of the servers listed above could be accessing their file data as an NFS client, or as NFS serving the data to such clients, or acting as both.

The CPU overhead of the NFS and TCP/IP protocol stacks (including data copies or reduced copy workarounds) becomes a significant matter in these clients and servers. File access using locally attached disks imposes relatively low overhead due to the highly optimized I/O path and direct memory access afforded to the storage controller. This is not the case with NFS, which must pass data to, and especially from, the network and network processing stack to the NFS stack. Frequently, data copies are imposed on this transfer; in some cases, several such copies are imposed in each direction.

Copies are potentially encountered in an NFS implementation exchanging data to and from user address spaces, within kernel buffer caches, in eXternal Data Representation (XDR) marshalling and unmarshalling, and within network stacks and network drivers. Other overheads such as serialization among multiple threads of execution sharing a single NFS mount point and transport connection are additionally encountered.

Numerous upper-layer protocols achieve extremely high bandwidth and low overhead through the use of RDMA. [MAF+02] shows that the RDMA-based Direct Access File System (with a user-level implementation of the file system client) can outperform even a zero-copy implementation of NFS [CHA+01] [CHA+99] [GAL+99] [KM02]. Also, file data access implies the use of large Unequal Loss Protection (ULP) messages. These large messages tend to amortize any increase in per-message costs due to the offload of protocol processing incurred when using RNICs while gaining the benefits of reduced per-byte costs. Finally, the direct memory addressing afforded by RDMA avoids many sources of contention on network resources.

2. Problem Statement

The principal performance problem encountered by NFS implementations is the CPU overhead required to implement the protocol. Primary among the sources of this overhead is the movement of data from NFS protocol messages to its eventual destination in user buffers or aligned kernel buffers. Due to the nature of the RPC and XDR protocols, the NFS data payload arrives at arbitrary alignment, necessitating a copy at the receiver, and the NFS requests are completed in an arbitrary sequence.

The data copies consume system bus bandwidth and CPU time, reducing the available system capacity for applications [RFC4297]. To date, achieving zero-copy with NFS has required sophisticated, version-

specific "header cracking" hardware and/or extensive platform-specific virtual memory mapping tricks. Such approaches become even more difficult for NFS version 4 due to the existence of the COMPOUND operation and presence of Kerberos and other security information, which further reduce alignment and greatly complicate ULP offload.

Furthermore, NFS is challenged by high-speed network fabrics such as 10 Gbits/s Ethernet. Performing even raw network I/O such as TCP is an issue at such speeds with today's hardware. The problem is fundamental in nature and has led the IETF to explore RDMA [RFC4297].

Zero-copy techniques benefit file protocols extensively, as they enable direct user I/O, reduce the overhead of protocol stacks, provide perfect alignment into caches, etc. Many studies have already shown the performance benefits of such techniques [SKE+01] [DCK+03] [FJNFS] [FJDAFS] [KM02] [MAF+02].

RDMA is compelling here for another reason; hardware-offloaded networking support in itself does not avoid data copies, without resorting to implementing part of the NFS protocol in the Network Interface Card (NIC). Support of RDMA by NFS enables the highest performance at the architecture level rather than by implementation; this enables ubiquitous and interoperable solutions.

By providing file access performance equivalent to that of local file systems, NFS over RDMA will enable applications running on a set of client machines to interact through an NFS file system, just as applications running on a single machine might interact through a local file system.

3. File Protocol Architecture

NFS runs as an Open Network Computing (ONC) RPC [RFC5531] application. Being a file access protocol, NFS is very "rich" in data content (versus control information).

NFS messages can range from very small (under 100 bytes) to very large (from many kilobytes to a megabyte or more). They are all contained within an RPC message and follow a variable-length RPC header. This layout provides an alignment challenge for the data items contained in an NFS call (request) or reply (response) message.

In addition to the control information in each NFS call or reply message, sometimes there are large "chunks" of application file data, for example, read and write requests. With NFS version 4 (due to the existence of the COMPOUND operation), there can be several of these data chunks interspersed with control information.

ONC RPC is a remote procedure call protocol that has been run over a variety of transports. Most implementations today use UDP or TCP. RPC messages are defined in terms of an eXternal Data Representation (XDR) [RFC4506], which provides a canonical data representation across a variety of host architectures. An XDR data stream is conveyed differently on each type of transport. On UDP, RPC messages are encapsulated inside datagrams, while on a TCP byte stream, RPC messages are delineated by a record-marking protocol. An RDMA transport also conveys RPC messages in a unique fashion that must be fully described if client and server implementations are to interoperate.

The RPC transport is responsible for conveying an RPC message from a sender to a receiver. An RPC message is either an RPC call from a client to a server, or an RPC reply from the server back to the client. An RPC message contains an RPC call header followed by arguments if the message is an RPC call, or an RPC reply header followed by results if the message is an RPC reply. The call header contains a transaction ID (XID) followed by the program and procedure number as well as a security credential. An RPC reply header begins with an XID that matches that of the RPC call message, followed by a security verifier and results. All data in an RPC message is XDR encoded.

The encoding of XDR data into transport buffers is referred to as "marshalling", and the decoding of XDR data contained within transport buffers and into destination RPC procedure result buffers, is referred to as "unmarshalling". Therefore, the process of marshalling takes place at the sender of any particular message, be it an RPC request or an RPC response. Unmarshalling, of course, takes place at the receiver.

Normally, any bulk data is moved (copied) as a result of the unmarshalling process, because the destination address is not known until the RPC code receives control and subsequently invokes the XDR unmarshalling routine. In other words, XDR-encoded data is not self-describing, and it carries no placement information. This results in a data copy in most NFS implementations.

One mechanism by which the RPC layer may overcome this is for each request to include placement information, to be used for direct placement during XDR encode. This "write chunk" can avoid sending bulk data inline in an RPC message and generally results in one or more RDMA Write operations.

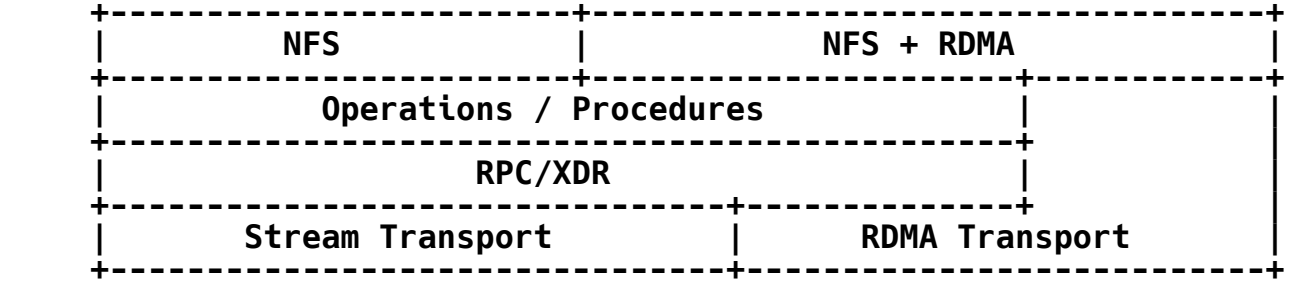
Similarly, a "read chunk", where placement information referring to bulk data that may be directly fetched via one or more RDMA Read operations during XDR decode, may be conveyed. The "read chunk" will

therefore be useful in both RPC calls and replies, while the "write chunk" is used solely in replies.

These "chunks" are the key concept in an existing proposal [RPCRDMA]. They convey what are effectively pointers to remote memory across the network. They allow cooperating peers to exchange data outside of XDR encodings but still use XDR for describing the data to be transferred. And, finally, through use of XDR they maintain a large degree of on-the-wire compatibility.

The central concept of the RDMA transport is to provide the additional encoding conventions to convey this placement information in transport-specific encoding, and to modify the XDR handling of bulk data.

Block Diagram



4. Sources of Overhead

Network and file protocol costs can be categorized as follows:

- per-byte costs - data touching costs such as checksum or data copy. Today's network interface hardware commonly offloads the checksum, which leaves the other major source of per-byte overhead, data copy.
- per-packet costs - interrupts and lower-layer processing (LLP). Today's network interface hardware also commonly coalesce interrupts to reduce per-packet costs.
- per-message (request or response) costs - LLP and ULP processing.

Improvement from optimization becomes more important if the overhead it targets is a larger share of the total cost. As other sources of overhead, such as the checksumming and interrupt handling above are eliminated, the remaining overheads (primarily data copy) loom larger.

With copies crossing the bus twice per copy, network processing overhead is high whenever network bandwidth is large in comparison to CPU and memory bandwidths. Generally, with today's end-systems, the effects are observable at network speeds at or above 1 Gbit/s.

A common question is whether an increase in CPU processing power alleviates the problem of high processing costs of network I/O. The answer is no, it is the memory bandwidth that is the issue. Faster CPUs do not help if the CPU spends most of its time waiting for memory [RFC4297].

TCP offload engine (TOE) technology aims to offload the CPU by moving TCP/IP protocol processing to the NIC. However, TOE technology by itself does nothing to avoid necessary data copies within upper-layer protocols. [MOG03] provides a description of the role TOE can play in reducing per-packet and per-message costs. Beyond the offloads commonly provided by today's network interface hardware, TOE alone (without RDMA) helps in protocol header processing, but this has been shown to be a minority component of the total protocol processing overhead. [CHA+01]

Numerous software approaches to the optimization of network throughput have been made. Experience has shown that network I/O interacts with other aspects of system processing such as file I/O and disk I/O [BRU99] [CHU96]. Zero-copy optimizations based on page remapping [CHU96] can be dependent upon machine architecture, and are not scalable to multi-processor architectures. Correct buffer alignment and sizing together are needed to optimize the performance of zero-copy movement mechanisms [SKE+01]. The NFS message layout described above does not facilitate the splitting of headers from data nor does it facilitate providing correct data buffer alignment.

4.1. Savings from TOE

The expected improvement of TOE specifically for NFS protocol processing can be quantified and shown to be fundamentally limited. [SHI+03] presents a set of "LAWS" parameters that serve to illustrate the issues. In the TOE case, the copy cost can be viewed as part of the application processing "a". Application processing increases the LAWS "gamma", which is shown by the paper to result in a diminished benefit for TOE.

For example, if the overhead is 20% TCP/IP, 30% copy, and 50% real application work, then gamma is 80/20 or 4, which means the maximum benefit of TOE is 1/gamma, or only 25%.

For RDMA (with embedded TOE) and the same example, the "overhead" (o) offloaded or eliminated is 50% (20% + 30%). Therefore, in the RDMA

case, gamma is 50/50 or 1, and the inverse gives the potential benefit of 1 (100%), a factor of two.

CPU Overhead Reduction Factor

No Offload	TCP Offload	RDMA Offload
1.00x	1.25x	2.00x

The analysis in the paper shows that RDMA could improve throughput by the same factor of two, even when the host is (just) powerful enough to drive the full network bandwidth without RDMA. It can also be shown that the speedup may be higher if network bandwidth grows faster than Moore's Law, although the higher benefits will apply to a narrow range of applications.

4.2. Savings from RDMA

Performance measurements directly comparing an NFS-over-RDMA prototype with conventional network-based NFS processing are described in [CAL+03]. Comparisons of Read throughput and CPU overhead were performed on two types of Gigabit Ethernet adapters, one type being a conventional adapter, and another type with RDMA capability. The prototype RDMA protocol performed all transfers via RDMA Read. The NFS layer in the study was measured while performing read transfers, varying the transfer size and readahead depth across ranges used by typical NFS deployments.

In these results, conventional network-based throughput was severely limited by the client's CPU being saturated at 100% for all transfers. Read throughput reached no more than 60 MBytes/s.

I/O Type	Size	Read Throughput	CPU Utilization
Conventional	2 KB	20 MB/s	100%
Conventional	16 KB	40 MB/s	100%
Conventional	256 KB	60 MB/s	100%

However, over RDMA, throughput rose to the theoretical maximum throughput of the platform, while saturating the single-CPU system only at maximum throughput.

I/O Type	Size	Read Throughput	CPU Utilization
RDMA	2 KB	10 MB/s	45%
RDMA	16 KB	40 MB/s	70%
RDMA	256 KB	100 MB/s	100%

The lower relative throughput of the RDMA prototype at the small blocksize may be attributable to the RDMA Read imposed by the

prototype protocol, which reduced the operation rate since it introduces additional latency. As well, it may reflect the relative increase of per-packet setup costs within the DMA portion of the transfer.

5. Application of RDMA to NFS

Efficient file protocols require efficient data positioning and movement. The client system knows the client memory address where the application has data to be written or wants read data deposited. The server system knows the server memory address where the local file system will accept write data or has data to be read. Neither peer however is aware of the others' data destination in the current NFS, RPC, or XDR protocols. Existing NFS implementations have struggled with the performance costs of data copies when using traditional Ethernet transports.

With the onset of faster networks, the network I/O bottleneck will worsen. Fortunately, new transports that support RDMA have emerged. RDMA excels at bulk transfer efficiency; it is an efficient way to deliver direct data placement and remove a major part of the problem: data copies. RDMA also addresses other overheads, e.g., underlying protocol offload, and offers separation of control information from data.

The current NFS message layout provides the performance-enhancing opportunity for an NFS-over-RDMA protocol that separates the control information from data chunks while meeting the alignment needs of both. The data chunks can be copied "directly" between the client and server memory addresses above (with a single occurrence on each memory bus) while the control information can be passed "inline". [RPCRDMA] describes such a protocol.

6. Conclusions

NFS version 4 [RFC3530] has been granted "Proposed Standard" status. The NFSv4 protocol was developed along several design points, important among them: effective operation over wide-area networks, including the Internet itself; strong security integrated into the protocol; extensive cross-platform interoperability including integrated locking semantics compatible with multiple operating systems; and (this is key), protocol extension.

NFS version 4 is an excellent base on which to add the needed performance enhancements and improved semantics described above. The minor versioning support defined in NFS version 4 was designed to support protocol improvements without disruption to the installed base [NFSv4.1]. Evolutionary improvement of the protocol via minor

versioning is a conservative and cautious approach to current and future problems and shortcomings.

Many arguments can be made as to the efficacy of the file abstraction in meeting the future needs of enterprise data service and the Internet. Fine grained Quality of Service (QoS) policies (e.g., data delivery, retention, availability, security, etc.) are high among them.

It is vital that the NFS protocol continue to provide these benefits to a wide range of applications, without its usefulness being compromised by concerns about performance and semantic inadequacies. This can reasonably be addressed in the existing NFS protocol framework. A cautious evolutionary improvement of performance and semantics allows building on the value already present in the NFS protocol, while addressing new requirements that have arisen from the application of networking technology.

7. Security Considerations

The NFS protocol, in conjunction with its layering on RPC, provides a rich and widely interoperable security model to applications and systems. Any layering of NFS-over-RDMA transports must address the NFS security requirements, and additionally must ensure that no new vulnerabilities are introduced. For RDMA, the integrity, and any privacy, of the data stream are of particular importance.

The core goals of an NFS-to-RDMA binding are to reduce overhead and to enable high performance. To support these goals while maintaining required NFS security protection presents a special challenge. Historically, the provision of integrity and privacy have been implemented within the RPC layer, and their operation requires local processing of messages exchanged with the RPC peer. This processing imposes memory and processing overhead on a per-message basis, exactly the overhead that RDMA is designed to avoid.

Therefore, it is a requirement that the RDMA transport binding provide a means to delegate the integrity and privacy processing to the RDMA hardware, in order to maintain the high level of performance desired from the approach, while simultaneously providing the existing highest levels of security required by the NFS protocol. This in turn requires a means by which the RPC layer may invoke these services from the RDMA provider, and for the NFS layer to negotiate their use end-to-end.

The "Channel Binding" concept [RFC5056] together with "IPsec Channel Connection Latching" [BTNSLATCH] provide a means by which the RPC and NFS layers may delegate their session protection to the lower RDMA

layers. An extension to the RPCSEC_GSS protocol [RFC5403] may be employed to negotiate the use of these bindings, and to establish the shared secrets necessary to protect the sessions.

The protocol described in [RPCRDMA] specifies the use of these mechanisms, and they are required to implement the protocol.

An additional consideration is protection of the integrity and privacy of local memory by the RDMA transport itself. The use of RDMA by NFS must not introduce any vulnerabilities to system memory contents, or to memory owned by user processes. These protections are provided by the RDMA layer specifications, and specifically their security models. It is required that any RDMA provider used for NFS transport be conformant to the requirements of [RFC5042] in order to satisfy these protections.

8. Acknowledgments

The authors wish to thank Jeff Chase who provided many useful suggestions.

9. References

9.1. Normative References

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009.

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.

[RFC5403] Eisler, M., "RPCSEC_GSS Version 2", RFC 5403, February 2009.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5042] Pinkerton, J. and E. Deleganes, "Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security", RFC 5042, October 2007.

9.2. Informative References

[BRU99] J. Brustoloni, "Interoperation of copy avoidance in network and file I/O", in Proc. INFOCOM '99, pages 534-542, New York, NY, Mar. 1999., IEEE. Also available from http://www.cs.pitt.edu/~jcb/publs.html.

[BTNSLATCH] Williams, N., "IPsec Channels: Connection Latching", Work in Progress, November 2008.

[CAL+03] B. Callaghan, T. Lingutla-Raj, A. Chiu, P. Staubach, O. Asad, "NFS over RDMA", in Proceedings of ACM SIGCOMM Summer 2003 NICELI Workshop.

[CHA+01] J. S. Chase, A. J. Gallatin, K. G. Yocum, "Endsystem optimizations for high-speed TCP", IEEE Communications, 39(4):68-74, April 2001.

[CHA+99] J. S. Chase, D. C. Anderson, A. J. Gallatin, A. R. Lebeck, K. G. Yocum, "Network I/O with Trapeze", in 1999 Hot Interconnects Symposium, August 1999.

[CHU96] H.K. Chu, "Zero-copy TCP in Solaris", Proc. of the USENIX 1996 Annual Technical Conference, San Diego, CA, January 1996.

[DCK+03] M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, M. Wittle, "The Direct Access File System", in Proceedings of 2nd USENIX Conference on File and Storage Technologies (FAST '03), San Francisco, CA, March 31 - April 2, 2003.

[FJDAFS] Fujitsu Prime Software Technologies, "Meet the DAFS Performance with DAFS/VI Kernel Implementation using cLAN", available from http://www.pst.fujitsu.com/english/dafsdemo/index.html, 2001.

[FJNFS] Fujitsu Prime Software Technologies, "An Adaptation of VIA to NFS on Linux", available from http://www.pst.fujitsu.com/english/nfs/index.html, 2000.

[GAL+99] A. Gallatin, J. Chase, K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds", 1999 USENIX Technical Conference (Freenix Track), June 1999.

[KM02] K. Magoutis, "Design and Implementation of a Direct Access File System (DAFS) Kernel Server for FreeBSD", in Proceedings of USENIX BSDCon 2002 Conference, San Francisco, CA, February 11-14, 2002.

[MAF+02] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, D. Gallatin, R. Kisley, R. Wickremesinghe, E. Gabber, "Structure and Performance of the Direct Access File System (DAFS)", in Proceedings of 2002 USENIX Annual Technical Conference, Monterey, CA, June 9-14, 2002.

[MOG03] J. Mogul, "TCP offload is a dumb idea whose time has come", 9th Workshop on Hot Topics in Operating Systems (HotOS IX), Lihue, HI, May 2003. USENIX.

[NFSv4.1] Shepler, S., Eisler, M., and D. Noveck, "NFSv4 Minor Version 1", Work in Progress, September 2008.

[PAI+00] V. S. Pai, P. Druschel, W. Zwaenepoel, "IO-Lite: a unified I/O buffering and caching system", ACM Trans. Computer Systems, 18(1):37-66, Feb. 2000.

[RDDP] RDDP Working Group charter, http://www.ietf.org/html.charters/rddpcharter.html.

[RFC4297] Romanow, A., Mogul, J., Talpey, T., and S. Bailey, "Remote Direct Memory Access (RDMA) over IP Problem Statement", RFC 4297, December 2005.

[RFC1094] Sun Microsystems, "NFS: Network File System Protocol specification", RFC 1094, March 1989.

[RPCRDMA] Talpey, T. and B. Callaghan, "Remote Direct Memory Access Transport for Remote Procedure Call", Work in Progress, April 2008.

[SHI+03] P. Shivam, J. Chase, "On the Elusive Benefits of Protocol Offload", Proceedings of ACM SIGCOMM Summer 2003 NICELI Workshop, also available from http://issg.cs.duke.edu/publications/niceli03.pdf.

[SKE+01] K.-A. Skevik, T. Plagemann, V. Goebel, P. Halvorsen, "Evaluation of a Zero-Copy Protocol Implementation", in Proceedings of the 27th Euromicro Conference - Multimedia and Telecommunications Track (MTT'2001), Warsaw, Poland, September 2001.

Authors' Addresses

Tom Talpey
170 Whitman St.
Stow, MA 01775 USA

Phone: +1 978 821-8577
EMail: tmtalpey@gmail.com

Chet Juszczak
P.O. Box 1467
Merrimack, NH 03054

Phone: +1 603 253-6602
EMail: chetnh@earthlink.net