

Tsunami: A High-Speed Rate-Controlled Protocol for File Transfer

Mark R. Meiss
Indiana University

Abstract

We describe a reliable transfer protocol, Tsunami, designed for faster transfer of large files over high-speed networks than appears possible with standard implementations of TCP. Tsunami is an application-level protocol that features rate control via adjustment of inter-packet delay rather than a sliding-window mechanism. Data blocks are transferred via UDP and control data are transferred via TCP. We also discuss future steps in development of the protocol.

1 Introduction

In the early years of TCP, its assumption that it can attribute all packet loss in the network to congestion was largely correct. Network capacities were limited, and the flow control offered by TCP allowed fair and effective use of a limited resource. In recent years, however, we have seen the growth of high-bandwidth dedicated research networks that have never experienced significant congestion. These networks still experience some level of packet loss due to cabling, equipment, and other glitches. This low-level packet loss is generally unavoidable. Because TCP interprets all packet loss as congestion, its exponential backoff algorithm causes transfer throughput to collapse even though bandwidth is still plentiful. The result is that actual performance of TCP transfers on high-speed networks lags far behind available capacity.

Network researchers have proposed a number of solutions to the throughput problems of TCP. They have included changes to TCP itself, the use of multiple concurrent TCP streams for a single transfer, and the use of very large packets (“jumbograms”). Of these, application developers can support only multiple TCP streams without modifying the operating system or other network infrastructure. While multiple TCP streams do increase performance, they address only the symptoms of the TCP throughput problem and not its source. In addition, an application that uses multiple TCP streams may compete unfairly for network resources with applications that use only a single stream.

This situation led the Advanced Network Management Lab to develop the Tsunami protocol for high-speed file transfer over these uncongested research networks. This protocol uses UDP for data transmission and a TCP control channel for managing retransmission requests and error rate information. This protocol offers transfer rates of

around 400 to 450 Mbps disk-to-disk on commodity hardware with no special tuning. Better tuned end systems have offered disk-to-disk transfer rates of as high as 1 Gbps.

In the remainder of this paper, we describe the basic design and architecture of Tsunami and present some preliminary performance results. We discuss other projects that concentrate on improving on the throughput of standard TCP using some of the techniques mentioned above, as well as another new protocol that bears some similarity to Tsunami. We also discuss the future direction of the Tsunami project.

2 Design

We can treat file transfer as a special problem in protocol design. While file transfer does require reliable transmission of data, we lose the ability to take advantage of some important properties of file transfers if we take the stream-based approach of TCP. We generally know the size of a file before transmission. We also usually have random access to the file on both the sending and receiving systems, which means we do not need to concern ourselves with the order in which we transmit or receive blocks of data.

Relying strictly on TCP for file transfer thus seems inappropriate. On the other hand, our primary design goal was to develop a protocol that can achieve acceptable performance running as a user-space application and not requiring any modification to the network infrastructure. Because the use of raw IP sockets is a restricted operation on most operating systems, we were left with no choice but to build Tsunami on top of UDP, or TCP, or both. As mentioned before, we chose the last option.

We also wanted Tsunami to be relatively insensitive to network latency, as this is another weakness of TCP that often requires manual tuning at the application level. Here we can exploit the fact that file transfers are insensitive to block order and abandon the sliding window algorithm of TCP in favor of flow control through adjustment of the delay time between packets. Instead of requiring the acknowledgment of received data, we can simply have the client send negative acknowledgments for data that did not arrive as expected.

Even though our target networks for using Tsunami are typically not congested, the protocol did need some form of flow control in order to avoid exceeding the capability of the client to handle the incoming traffic. However, we still wanted to avoid collapsing the transmission rate in the presence of low-level packet loss. For this reason, we settled upon an adjustable error threshold. Packet loss above the threshold causes an exponential rise in the inter-packet delay; loss below the threshold causes an exponential decrease in the inter-packet delay until a target rate has been met.

3 Architecture

Our initial implementation of the Tsunami protocol consists of two user-space applications, a client and a server. The structure of these applications are illustrated in Figure 1.

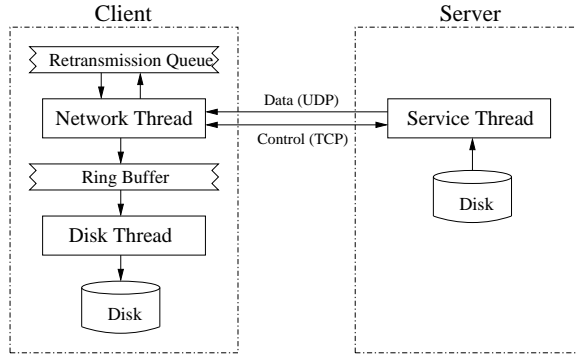


Figure 1: Tsunami architecture

During a file transfer, the client has two running threads. The network thread handles all network communication, maintains the retransmission queue, and places blocks that are ready for disk storage into a ring buffer. The disk thread simply moves blocks from the ring buffer to the destination file on disk. The server creates a single thread in response to each client connection that handles all disk and network activity.

The client initiates a Tsunami session by connecting to the TCP port of the server. Upon connection, the server sends a small block of random data to the client. The client then *xor*'s this random data with a shared secret, calculates an MD5 checksum, and transmits the result to the server. The server performs the same operation on the random data and verifies that the results are identical. We thus establish client authentication.

After exchanging protocol revision codes, the client sends the name of the requested file to the server. If the server indicates that the file is available, the client sends its desired block size, target transfer rate, error threshold, and inter-packet delay scaling factors. The server responds with the length of the file, the agreed-upon block size, the number of block, and a timestamp. The client then creates a UDP port and transmit the port number.

At this point, we are ready to transmit the file.

3.1 Server algorithm

On each pass through the transmission loop of the server thread, the following actions are performed:

1. If there is a message from the client waiting on the TCP control stream, we read it. This message can have one of three types.
 - (a) If it is a *retransmission request*, the server will send the requested block on this iteration instead of the block it would normally transmit.
 - (b) If it is a *restart request*, the server will flush the TCP control channel and restart transmission of the file at the given block number. The server assumes that the client has received every block previous to the restart block.

- (c) If it is an *error report*, then the server examines the error rate reported by the client. If it is over the threshold, the inter-packet delay is scaled upward. If it is under the threshold, then the inter-packet delay is scaled downward unless we have already reached the target rate.
 - (d) If it is a *completion report*, then the client has received every block in the file successfully and the control loop will exit. The server then awaits another filename from the client.
2. If the server did not have any waiting message from the client, then the server will send the next block of the file. If the last block of the file has already been sent, it is sent again.
 3. The server waits for the remaining portion of the inter-packet delay time.

3.2 Client algorithm

The disk thread in the client simply pulls blocks out of the ring buffer. If a block has not been saved already, it is saved to disk, and a bit array is updated to reflect the reception of the block. The thread also decrements the number of blocks remaining.

The network thread has a more involved control loop that performs the following operations on each iteration:

1. The thread reserves a slot in the ring buffer for an incoming data block.
2. The thread blocks until it receives a new data block. This block is marked as being one of three types.
 - (a) If the block is *original*, the client puts the indices of any blocks between the last original block received and this block in the retransmission queue. For example, if the client last received block 5100 and then receives block 5102, it will place block 5101 in the retransmission queue.
 - (b) If the block has been *retransmitted*, the client takes no special action.
 - (c) If the block is the *last block*, the client will no longer receive any original blocks. When the retransmission queue becomes empty because all requests have been satisfied, the client will indicate a complete transmission to the server.

On every 50th iteration, if the update period has elapsed, the client calculates the current error rate and sends it to the server. The error rate is based on the number of blocks placed in the retransmission queue in the last period, the current size of the ring buffer, and the previous value for the error rate.

If the retransmission queue has sufficiently few entries that are still valid, the client sends the contents of the queue to the server. Otherwise, the client sends a restart request to the server.

3.3 Tuning parameters

Because Tsunami is an experimental protocol, the user can adjust many of the parameters that affect the behavior of the algorithm. These parameters include:

- which network layer to use (IPv4 or IPv6);
- the size of each data block;
- the target transfer rate;
- the scaling factors for the inter-packet delay;
- the proportion of historical data used in calculating the error rate;
- the threshold error rate;
- the maximum size of the retransmission queue;
- the maximum number of entries in the ring buffer;
- the size of the UDP send and receive buffers; and
- the interval between update periods.

Neither the client nor the server attempt to modify global system properties that affect the performance of the protocol, such as filesystem parameters and network interface configuration. This is assumed to be privileged operations outside the scope of the Tsunami application.

4 Experimental Results

Our experience thus far with Tsunami has shown great promise.

An early version of the protocol that omitted disk activity was able to sustain an mean transmission rate of about 850 Mbps for over 17 hours over the Global Terabit Research Network between Seattle, Washington, and Brussels, Belgium. The “virtual file” consisted of a short message repeated indefinitely many times. We forced the length of this repeated message to be relatively prime to the data block size so as to eliminate any cache benefits we might otherwise enjoy.

Network engineers tested a recent version of the protocol on a temporary dedicated connection between TRIUMF in Vancouver, British Columbia, and CERN in Geneva, Switzerland, with about 150 ms latency. These tests involved actual disk activity at both ends and normally ran at around 600 Mbps once the end systems were tuned, though speeds of 1 Gbps were achieved on occasion. Representative performance of these runs can be seen in Figure 2.

Our current test apparatus consists of two low to medium-end Dell server systems. One (*singly*) has a single 1 GHz Pentium III processor and 1 GB of memory; the other (*dually*) has two 1.1 GHz Pentium III processors and 1 GB of memory. Both

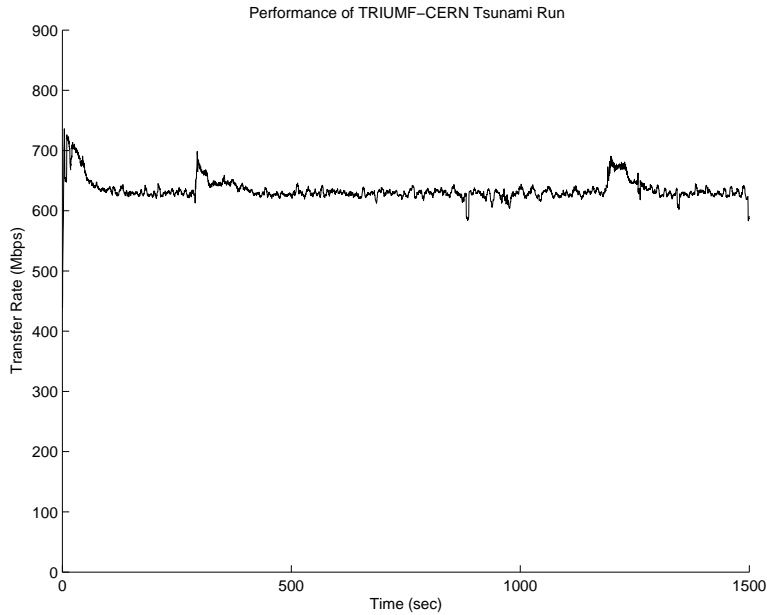


Figure 2: Tsunami Performance (TRIUMF-CERN)

have Intel PRO/1000 Ethernet adapters seated in 64-bit 66-MHz PCI slots. Their network connections are via Cat-5 cable to an HP ProCurve 4000 switch. The disk subsystem of each system contains a 3ware EIDE RAID controller that stripes the filesystem across five EIDE hard disks. Both are running RedHat Linux with version 2.4.18-10 of the kernel and use the standard *ext3* filesystem.

We measured the application-level disk performance of each system with simple test applications that write 5 GB of data to disk and read it back again. The results are shown below.

System	Read Speed	Write Speed
singly	1130 Mbps	850 Mbps
dually	1500 Mbps	900 Mbps

We have not adjusted any of the disk or network parameters from their default values on either system, except for enabling large transmit and receive buffers for sockets. The network stacks have both IPv4 and IPv6 available. The latency between the two systems is on the order of 200 microseconds.

In this test environment, a typical transfer of a 5 GB file achieves a mean throughput of between 400 and 450 Mbps. Representative performance can be seen in Figure 3. IPv6 performance is typically around four percent lower than IPv4 performance; a fair amount of this difference can be attributed to packet headers.

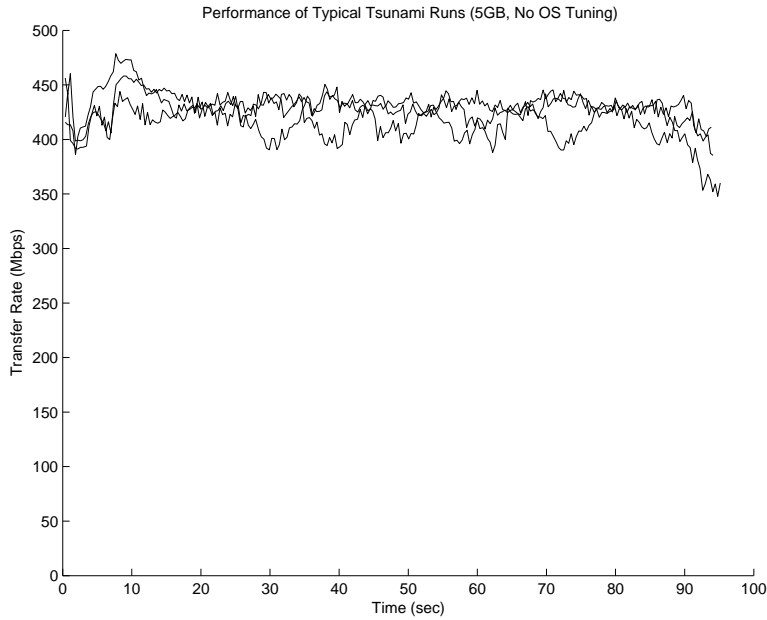


Figure 3: Tsunami Performance (Test Environment)

5 Further Development

The Tsunami parameter space has many dimensions, especially if we consider the additional complexity of trying to tune operating system parameters as well. An exhaustive search of this parameter space for optimal settings is not feasible, so we will conduct future research into tuning the protocol using a genetic algorithm. Preliminary results indicate that Tsunami throughput may fairly insensitive to some of our parameters, such as the amount of history included in the error rate.

We will also perform a detailed study of how Tsunami transfers interact with other flows on the same path through the network. Even though Tsunami does back off exponentially in response to high error rates, we cannot assert that the current version is not an overly aggressive protocol until we have done more analysis. We also have not yet studied how multiple concurrent Tsunami transfers interact with one another.

Though we believe that Tsunami should be insensitive to network latency, we have yet to perform tests involving controlled amounts of network delay. Previous performance implies that we have avoided the latency sensitivity of TCP, but more research is necessary.

We are also making a library version of the protocol and exploring the possibility of integrating Tsunami with the Globus Toolkit.

6 Related Work

Many network researchers have tried to address performance problems with TCP using the methods identified in the introduction: modifications to TCP, multiple streams, and increased packet size.

RFC 1323 describes modifications to TCP that include the large windows option for transmission over circuits with large bandwidth-delay products [11]. It also includes a mechanism for including timestamps in a TCP stream, which can be used for statistical measurement and avoiding accidental collisions from wrapped sequence numbers. These extensions can be used on modified hosts without requiring changes to intermediate network devices.

Others have proposed increasing the starting window size of TCP [2]. This would alleviate some of the performance problem associated with the “slow start” part of the TCP congestion algorithm. This proposal is particularly useful in providing better performance in problem domains that involve large numbers of short flows, such as web browsing [14].

Another proposal uses selective acknowledgments to avoid unnecessary retransmissions of TCP segments that were received correctly [13]. These redundant retransmissions are a side-effect of the cumulative acknowledgment used by TCP. Tsunami tries to avoid the same situation by using negative acknowledgments instead. Results suggest that this modification does not make the altered TCP stack unreasonably aggressive on busy networks [7]. It may be particularly useful over wireless connections and other links with loss unrelated to congestion [3].

Several existing systems use multiple TCP streams in order to improve performance. One of these is *bbFTP*, which focuses on the transfer of large files and includes support for other TCP enhancements such as large windows [6]. The *pTCP* protocol takes the different approach of striping a single transfer across multiple physical pathways, which seems to yield better results than simply using multiple sockets without reference to features of the individual network paths [10]. The *GridFTP* system offers support for multiple TCP streams through FTP command extensions, which helps to avoid the need for multiple file transfer services for varying types of client systems [1].

IPv6 includes support for jumbograms, which the IPv6 standards define as individual packets larger than 64 kilobytes [4]. The utility of these jumbograms is somewhat limited by features of the TCP standard, such as the initial maximum segment size negotiation and the size of the urgent pointer.

Of course, because of fragmentation, large packets at the network layer do not necessarily imply large packets at the physical layer. Empirical studies imply that raising the Ethernet MTU to around 8 KB can improve throughput and reduce CPU utilization by reducing per-packet overhead for network hosts [5, 9]. It is less clear how large MTUs affect the performance of TCP along the sort of long-distance networks with background packet loss described in this paper.

Finally, another recent file transfer protocol, SABUL, shares many traits in common with Tsunami [8]. Common features of the two protocols include the use of a TCP stream for control and UDP for data transmission, and the use of inter-packet delay instead of the sliding window algorithm to manage the basic transmission rate. The favorable performance results reported by the SABUL project imply that the basic

architecture of both Tsunami and SABUL is a fruitful area for further research.

7 Conclusion

Although the Tsunami protocol is still in development, our results to date indicate that it can successfully circumvent many of the performance problems with TCP on high-speed research networks with low-level packet loss. The key features that shape the performance of Tsunami are its specialization to the file transfer domain and its use of inter-packet delay as a means of flow control.

An implementation of Tsunami is available for download from the web site of the Advanced Network Management Laboratory at Indiana University [12]. This software is under an open source license.

8 Acknowledgments

Thanks are due to my colleagues at the Advanced Network Management Laboratory, David Ripley in particular, and the network researchers at TRIUMF and CERN. Steven Wallace and Dennis Gannon of Indiana University have provided great motivation and assistance in Tsunami research. This work is partially supported by a grant from the NSF.

References

- [1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments, 2001.
- [2] M. Allman, S. Floyd, and C. Partridge. *RFC 2414: Increasing TCP's Initial Window*. IETF Network Working Group, 1998.
- [3] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.
- [4] D. Borman, S. Deering, and R. Hinden. *RFC 2675: IPv6 Jumbograms*. IETF Network Working Group, 1999.
- [5] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.
- [6] Gilles Farrache. *bbFTP Documentation*. 2002.
- [7] Sally Floyd. Issues of TCP with SACK. Technical report, 1996.
- [8] Yunhong Gu, Xinwei Hong, Marco Mazzucco, and Robert Grossman. SABUL: A high performance data transfer protocol. *Submitted for Publication*, 2002.

- [9] D. Halstead, B. Bode, D. Turner, and V. Lewis. Giga-plant scalable cluster, 1999.
- [10] H.-Y. Hsieh and R. Sivakumar. ptcp: An end-to-end transport layer protocol for striped connections. In *IEEE International Conference on Network Protocols*, 2002.
- [11] V. Jacobson, R. Braden, and D. Borman. *RFC 1323: TCP Extensions for High Performance*. IETF Network Working Group, 1992.
- [12] Advanced Network Management Lab. Research at the Advanced Network Management Lab, <http://www.anml.iu.edu/anmlresearch.html>, 2003.
- [13] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *RFC 2018: TCP Selective Acknowledgement Options*. IETF Network Working Group, 1996.
- [14] K. Poduri and K. Nichols. *RFC 2415: Simulation Studies of Increased Initial TCP Window Size*. IETF Network Working Group, 1998.