

# High Throughput Data Transfers using the Tornado Transport Protocol

Vijayan Prabhakaran, Joseph Stanley and Paul Barford

*Abstract*— The need to transfer extremely large files over very high bandwidth network paths is becoming more and more common. Current window-based transport protocols often limit throughput to a level well below what might otherwise be achievable. In this paper we describe and evaluate the Tornado Transport Protocol (TTP) which is designed to be reliable, to achieve very high throughput and to maintain an adjustable level of TCP-friendliness. Reliability is achieved through the use of efficient forward error correction. High throughput is achieved by eliminating the need for acknowledgment-based pacing from the receiver. TCP-friendliness is achieved by setting a threshold to specify when to react to loss. We compare TTP with an out-of-the-box configuration of TCP NewReno and two TCP-based methods for high throughput file transfers under a variety of traffic conditions. Our simulations show that for large file transfers in long delay, large bandwidth environments, TTP achieves throughput nearly an order of magnitude higher than out-of-the-box TCP NewReno, and up to 300% higher than the next best high performance transport protocol. In each case we assess TCP-friendliness and find that TTP effectively allows competing TCP flows a fair share of bandwidth.

## I. INTRODUCTION

Internet technology has evolved to the point where network bandwidth can greatly exceed end hosts' capacity to transmit packets. This is coupled with a growing demand to transfer very large data sets – on the order of a terabyte or more in scientific applications – often over links with large propagation delays. It would be reasonable to assume that end hosts configured with gigabit Ethernet cards and connected to a network with even higher capacity (10 Gbps backbone links are not uncommon today) should be able to transfer a 1TB file in about 2.2

hours over an uncongested path. However, this is far from the case. In fact, it is common today to send hard disks through the mail for data sets of a terabyte or more since wide area systems do not provide the performance required for transfers of this magnitude – even in dedicated, highly engineered networks.

One of the most basic reasons for the ineffective utilization of available network bandwidth is that window-based transport protocols such as TCP place hard limits on the amount of data that can be in flight at any instant in time (this is the basic feature of flow control). The problem is that out-of-the-box configurations of TCP are optimized for low bandwidth links which limit potential throughput in large bandwidth environments. This observation is not new. For example, Jacobson and Braden proposed TCP extensions to improve performance of transfers over long delay paths in 1988 [1]. That work resulted in a series of RFC's all targeted at improving TCP performance over high bandwidth paths [2], [3]. Recently, Floyd has proposed further extensions to TCP which target the congestion control mechanism and should enable high per-connection throughput in low loss environments [4]. Results of initial tests of these capabilities appear to be promising [5].

There are three standard alternatives to using out-of-the-box TCP for large data transfers. The first is to open a number of parallel TCP connections as proposed in [6], [7]. In [8], Hacker *et al.* evaluate the performance of parallel TCP connections and show how to select the number of sockets that maximize overall throughput. The second approach is to scale the send/receive windows/buffers in TCP to match the delay bandwidth product between end hosts. A number of projects have adopted this approach including [9], [10]. The third approach is to use UDP-based transport methods which requires additional functionality to facilitate capabilities inherent in TCP such as reliability. These methods

V. Prabhakaran, J. Stanley and P. Barford are members of the Computer Science Department at the University of Wisconsin, Madison. E-mail: vijayan.jass.pb@cs.wisc.edu.

have been proposed principally in the context of reliable multicast whereas our application is unicast - examples include [11], [12], [13].

Our research is focused on the question of how to maximize throughput without sacrificing reliability or congestion control for the common case in large data transfers. Our approach differs from most of the aforementioned high throughput protocols in that we do not rely on TCP as the basis for our method. Instead, we use UDP as our foundation and achieve very high throughput at the expense of *guaranteed* reliability and the *most strict* interpretations of congestion control and TCP-friendliness [14].

Our contribution in this paper is the description and evaluation of the Tornado Transport Protocol (TTP). This protocol has been developed to provide high performance, reliable delivery of data in large bandwidth, long delay, low loss networks. TTP uses forward error correction (FEC) to recover lost data in a manner that is somewhat similar to Lundqvist and Karlsson’s incorporation of FEC in TCP [15]. While TTP differs significantly from that work in both approach and objectives, they also demonstrate the performance benefits of using FEC in large bandwidth, long delay environments. TTP also incorporates a variable initial send rate and a *limited* acknowledgment mechanism to modulate sending speed during data transfers. Furthermore, TTP’s design includes an *adjustable* level of TCP-Friendliness that allows TTP flows to comfortably coexist with TCP flows under a range of network conditions.

We evaluate the performance and impact of TTP in a series of *ns2* simulations [16]. Our simulations compare the throughput of TTP against a default TCP NewReno implementation, a parallel TCP implementation and a window right-sized TCP implementation. Our first set of experiments show that TTP achieves much higher throughput than each of these alternatives over a range of transfer sizes, network configurations and congestion levels. Our second set of experiments evaluates the TCP-Friendliness of TTP. We show that it adapts well to congestion conditions and has negligible impact on TCP flows sharing a bottleneck link. These results suggest that TTP may be very effective for transferring large data files in high performance networks.

The combination of TTP’s mechanisms emphasize high utilization of available bandwidth at the

potential cost of failed transfers and the potential to rapidly introduce congestion. Tuning TTP for optimal performance and minimal cost is based on knowledge of a small amount of end-to-end network state. While it can certainly be tuned to operate appropriately in today’s commodity networks, many of its performance benefits would likely be lost. Because of these issues, we propose TTP as a mechanism best suited to large data transfers in specialized, high performance networks such as Abilene, ESnet or the Teragrid.

The rest of this paper is organized as follows. TTP’s architecture is described in Section II. We evaluate the performance and impact of TTP in Section III. We summarize our results and conclude in Section IV.

## II. THE TORNADO TRANSPORT PROTOCOL

The Tornado Transport Protocol extends the User Datagram Protocol (UDP) to facilitate very high throughput data transfers with reliability and TCP-friendliness. In this section, we describe TTP’s architecture and implementation details.

### A. Reliability in TTP

Reliability in the transport layer can be provided by two different methods. The first and most common uses acknowledgments (ACKs) generated by the receiver, and the other uses forward error correction generated by the sender. In the first approach, after correctly receiving data packets, a receiver will return ACK packets to the sender. This is the approach used by TCP<sup>1</sup>. ACKs play an additional role in TCP, namely, as a means for indicating congestion in the network. Expiration of a timeout period before receiving an ACK or the receipt of an ACK-based signal (such as three duplicate ACKs) are implicit indications of network congestion and cause the send rate to be throttled. The advantages of using ACK-based reliability are that minimal redundant data is sent over the network and minimal overhead is incurred at the end hosts.

Forward error correction through the use of specialized codes is another standard means for achieving reliability. In this approach, redundant informa-

<sup>1</sup>See [17] for a good description of ACK-based reliability in TCP

tion is added to the data such that the receiver is able to recover the original data after receiving some percentage of the total number of encoded packets. Examples of using encoding for facilitating reliable data delivery include [11], [18], [19]. An important advantage of this approach is that reliability and flow control can be decoupled. This has significant implications in large bandwidth environments where adjusting send rates based on ACK streams can be a significant limitation to achieving a desired level of performance. However, there are a number of disadvantages to the use of FEC.

Since redundant data must be sent along with original data, the total number of packets transmitted from sender to receiver will increase. A closely related disadvantage is that in spite of aggressive encoding, the receiver might not be able to recover all of the data if loss rates are above the encoding threshold. Therefore, careful selection of the encoding threshold is required to balance efficiency in the number of packets versus the ability to fully recover data in lossy environments. There is also an encoding and decoding overhead at the sender and the receiver. For example, encoding and decoding 10MB of data using traditional Reed-Solomon codes can take thousands of seconds [20]. Fortunately, recent developments in the coding theory such as the development of Tornado codes [11] reduce the same encoding/decoding time to a few seconds.

The Tornado code algorithm calls for a source wishing to transmit  $k$  data packets to generate an additional  $l$  check packets. All  $k + l$  packets are transmitted by the sender. If the receiver successfully receives any  $m(> k)$  packets then the decoding algorithm enables complete recovery of the original  $k$  data packets. Parameterization in Tornado encoding/decoding requires the specification of the number of data packets  $k$ ,  $\epsilon$  which is the reception overhead and the error rate of the link which determines the number of check packets required. Tornado codes are built using sparse, *xor*-based equations, which enable the high encoding and decoding speeds. The encoding process partitions the file into equal sized fragments called *data nodes* and generates the set of *check nodes*, which are the same size in bytes as data nodes. Using a series of bipartite graphs, check nodes are assigned to two or more neighbors and are calculated as the bit-wise *xor* of

the value of its neighbors. Decoding is an iterative process that also consists of a series of *xor* operations between neighbor fragments to recover all of the original data.

To achieve reliability in TTP without imparting high encoding/decoding or packet transmission overhead we use Tornado codes in conjunction with a projected loss rate threshold. We use this threshold to set the encoding level such that we send a sufficient number of packets for a successful recovery with high probability. The loss rate projection is derived from knowledge of characteristics of the end-to-end channel. The initial value for this threshold can be derived through the use of measurement environments such as Surveyor [21] or IDMaps [22], or, in the case of specialized networks in which routes are relatively static (such as Abilene) through historical accounting of data transfers. As will be seen in the next section, TTP has the ability to adapt itself to loss characteristics of a specific path. This capability is facilitated through *chunked* encoding of large files (in our experiments we encode 1MB chunks). The amount of padding (check packets) added to the encoding of chunks after the first depends on the loss rate measured in the prior chunk<sup>2</sup>.

### B. Flow Control in TTP

TTP's flow control mechanism is designed to provide very high throughput in networks with large bandwidths and short-lived loss events<sup>3</sup>. The idea is to begin sending at a very high rate and to only reduce the sending rate when congestion exceeds certain thresholds. Flow control is enabled through a limited acknowledgment mechanism in the form of *control packets* that are also used to set encoding levels for chunks during a file transfer. The mechanism is designed to be tunable, allowing TTP to be optimized for performance.

Control packets, sent by the receiver to the sender, are simple bitmaps that indicate the status of the last  $n$  packets received. A "1" indicates that a packet was successfully received and a "0" indicates that a packet was lost. Sequence numbers are used in TTP data packets to provide an ordering in the control packet bitmap but do not relate to specific pieces

<sup>2</sup>Thus, the only chunk that we need to use the projected loss rate for is the first.

<sup>3</sup>The Abilene network has these characteristics [23]

of original data since no ordering is required by the Tornado codes. When there are no packet losses, one control packet is transmitted by the receiver for every  $n$  data packets. The receipt of a control packet showing zero loss causes the sender to *increase* its send rate. The receipt of a control packet showing that losses occurred can cause the sender to reduce its send rate and/or to increase the padding used in encoding the next chunk depending on whether losses exceeded specified thresholds. When losses exceed these thresholds, the receiver will react by increasing its feedback rate by sending control packets for every  $n/2$  packets in an attempt to increase sensitivity to loss conditions. Similarly, when no loss is measured for the last  $n$  packets, the receiver will decrease its feedback rate by a constant factor.

Since control packets are used only as a mechanism to infer network conditions, the loss of a control packet will result in diminished response by the sender and possibly an inaccurate calculation of link loss rate. We view the former as being of minor importance. The latter only becomes an issue if both the perceived loss rate and the overall link loss rate increased significantly and the control packet with this information was lost. Since our focus is on networks with very low loss rates, we view this as a very low probability event.

A challenge in designing TTP was to decide how to establish congestion thresholds. Unlike TCP which reduces its send rate whenever loss is sensed, TTP was designed to have the ability to ignore short-lived loss events. This enables the send rate to be maintained at a higher level at the risk of contributing to further congestion in the network. We implement TTP's congestion thresholds in two ways. The first method allows TTP to be sensitive to persistent short lived congestion. Specifically, the *perceived loss rate* (short term loss rate seen by the receiver which is the ratio between the number of packets missing and the number of packets received for the last  $n$  packets) is calculated by the sender using the information in the control packet bitmap. If the perceived loss rate is greater than the *link loss rate* (long term loss rate seen by the sender which is the ratio between the *total* number of packets lost and the *total* number of packets transmitted) then TTP reacts by reducing its sending rate. The second congestion threshold considers the length of individual

loss events. Namely, if  $q$  consecutive packets are lost at any time, the link is considered to be in an unstable state and TTP reacts by reducing its sending rate.

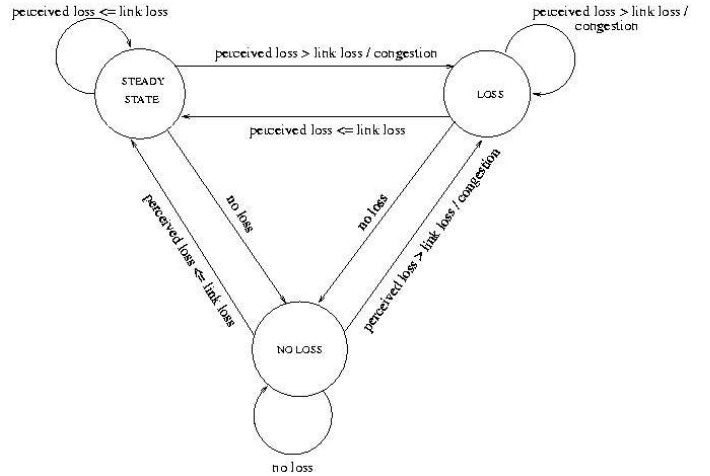


Fig. 1. TTP state transition diagram

The state transition diagram for TTP is shown in Figure 1. The sender starts in the “steady state” transmitting at a sufficiently high initial speed. This value, like the loss rate threshold, is meant to be aggressively set based on bottleneck bandwidth which can be determined by inspection in high performance networks with relatively static paths or through measurements. The sender continues to transmit at this rate until a control packet is received. If the congestion threshold is exceeded, a transition from “steady state” to “loss” state occurs. The packet send rate is halved (multiplicative decrease) and the data encoding rate is adjusted in the next chunk if necessary. If the bitmap indicates no loss, a transition from “steady state” to “no loss” state occurs, the packet send rate is increased by a constant factor (additive increase) and the data encoding rate is adjusted in the next chunk if necessary. If the bitmap indicates a loss and no congestion threshold is exceeded, no transition occurs. To prevent drastic changes in send rates maximum and minimum transmission rates are fixed and the transmission rate is never increased or decreased beyond these values. The multiplicative decrease on congestion imparts TCP-friendliness and setting the consecutive loss threshold  $q$  enables reaction to loss to be tunable.

### C. Implementation Details

The link loss rate and initial send rate play major roles in the behavior of TTP. Loss rate dictates the behavior of the encoding algorithm, the amount of data transmitted and the ability to recover from loss. Because of these issues, we implement calculation of loss rate dynamically as a weighted moving average of the current link loss rate and the perceived loss rate. This ensures that the loss rate adapts to the actual loss rate of the link and is not affected drastically due to transient high losses.

Initial send rate dictates how much of the typical bottleneck bandwidth will be used by a TTP flow. We found in the experiments reported in the next section that this has a significant impact on the overall performance of TTP, and it should be chosen as aggressively as possible. Send rates after the receipt of the first control packet are determined by congestion conditions and min/max transmission thresholds.

Since TTP is aimed at large data transfers, the task of encoding data chunks can be pipelined with data transmission. Data transfers are divided into two phases: the *encoding* phase and the *transmission* phase as is seen in Figure 2. Data is split into chunks, and each chunk is encoded using the calculated link loss rate. While this chunk of data is transmitted the next data chunk is encoded. In a similar manner the decoding can be overlapped with data reception. Through the parallel execution of encoding/decoding with transmission/reception, the overhead of encoding/decoding is effectively hidden.

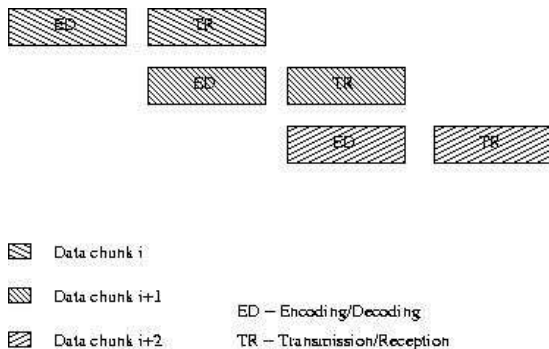


Fig. 2. Parallelized Encoding/Transmission/Decoding in TTP

## III. TTP EVALUATION

### A. Simulation Environment

Our analysis of TTP began by implementing the Tornado code algorithm to measure the encoding and decoding overheads. Our implementation was tested on a 1GHz PIII machine with 512MB RAM. The operating system used was Linux 2.4.18. The timing for encoding/decoding a range of file sizes which might reasonably be used for TTP chunks are shown in Figure 3. The figure shows that even for 10MB data chunks the time taken for encoding is less than 5 seconds. The decoding times are negligible in all instances. These results show that the encoding/decoding overhead can easily be amortized over the time to transmit very large data chunks even in high bandwidth environments.

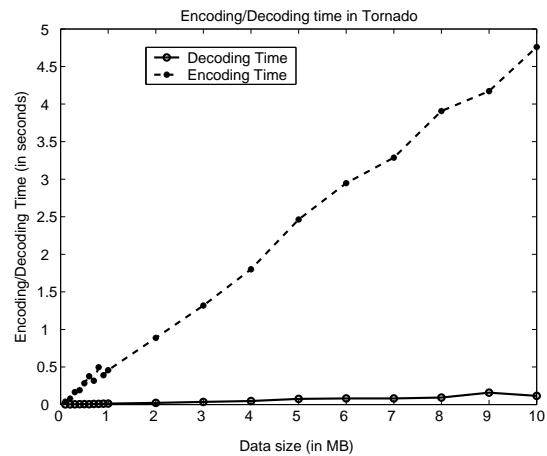


Fig. 3. Encoding/Decoding overhead in Tornado codes

We used Network Simulator v2.0 for our performance evaluation of TTP. We used a simple dumb-bell topology as is shown in Figure 4 with a high speed test source (TTP plus those listed below), a TCP source (which sends the same size file as our test protocol and is used to assess TCP-friendliness), an ON/OFF TCP Pareto source (using default *ns2* parameters for file sizes and OFF time parameter set to 200ms) for background traffic, and a constant bit rate (CBR) source (using the CBR default packet size transmitting at 1Mbps) for additional background traffic. We evaluate performance using two different network configurations. The “short-slim” configuration (shown in the figure) has a bottleneck bandwidth of 50 Mbps with 25ms propaga-

tion delay and end links at 10 Mbps with 2ms propagation delay. The “long-fat” configuration has a bottleneck bandwidth of 155 Mbps with 100ms propagation delay and end links at 100 Mbps with 2ms propagation delay. In both topologies the bottleneck queue size was set to 10 packets (TTP’s performance improves as the bottleneck queue size increases so the results reported in this section with this small buffer are pessimistic for TTP). We specify loss rate in each experiment using the link error model provided by *ns2*.

We compared TTP’s performance with the *ns2* implementation of TCP-NewReno and two other TCP based high speed transfer protocols. NewReno was configured with a receive window set to 20 packets. Our choices for other high speed implementations were Parallel Sockets [7] and Dynamic Right Sizing [10]. For simulating Parallel Sockets, we used 8 parallel TCP NewReno connections each transmitting 1/8th of the data based on the work in [7]. For Dynamic Right Sizing, TCP calculates an *effective window* (*ewnd*) as  $ewnd = \min(fwnd, cwnd)$ , and since the congestion window (*cwnd*) varies dynamically according to the network state, the flow window (*fwnd*) should vary with the bandwidth-delay product of the network [10]. For our simulation experiments we set the flow window to be equal to the bandwidth-delay product.

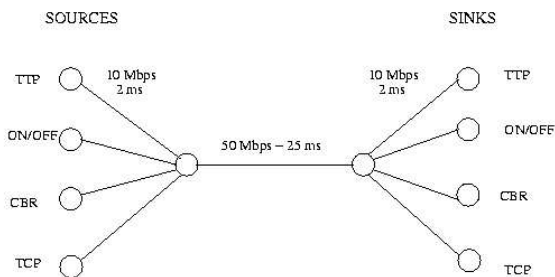


Fig. 4. Simulation network topology

TTP has a number of tunable features including initial send rate (which we set to 80% of the bottleneck bandwidth), congestion indicator  $q$  (which we set to 3), initial link loss rate (which we set to 50%)<sup>4</sup>, control packet size (which we set to 1K bits) coding chunk size (which we set to 1Mb) and min/max

<sup>4</sup>Our moving average method quickly reduces this value to track actual loss rate for subsequent data chunks

transmission rates (which we set to 8Mbps/10Mbps and 8Mbps/100Mbps for the short-slim, long-fat configurations respectively). In our simulations, we do not actually encode/decode the data chunks sent via TTP. Instead, we use encode/decode times from Figure 3 for the 1Mb chunks which are actually transferred and then include this latency in the overall transfer times that we measure. Setting the congestion indicator value  $q$  means that if a control packet is received with 3 consecutive packets lost, the sender will multiplicatively decrease its send rate. Obviously larger values will enable TTP to perform even better but at the expense of TCP-friendliness. Thorough investigation of the sensitivity of TTP performance to parameterization is beyond the scope of this paper and is left for future work.

### B. Throughput Evaluation

To evaluate the effectiveness of TTP in exploiting the available bandwidth we transferred different size files under different congestion levels in both the “short-slim” and “long fat” simulation environments. In similar network conditions we measured the transmission latencies of TCP NewReno (TCP), Parallel Sockets (PSOCK), and Dynamic Right Sizing (DRS). The time taken for each of file transfers with different loss rates for the short-slim configuration is show in Figures 5 to 9 (note: in all tests TTP transferred enough packets to successfully recover the data).

In all cases, TTP provides higher throughput than both TCP and DRS. Under very low loss rates (0%, 0.01% and 0.1%) for this low transmission latency environment, PSOCK outperforms TTP. However, as the link loss rate increases to 1% and 2%, TTP provides up to 20% higher throughput than the next best alternative (PSOCK). In fact, as loss rates increase beyond 2%, TTP’s performance improvements over the other protocols continues to increase. For smaller file sizes, TTP performance is approximately equal to the TCP variants because of the encoding overhead. However, as file sizes increase, this encoding overhead is easily amortized over the larger transmission times and TTP significantly outperforms TCP. It should also be noted that TTP does not have the overhead of opening multiple TCP connections (as in PSOCK) which requires larger buffering ca-

capacity at both the transmitter and receiver. This overhead is not considered in these experiments, thus the results for PSOCK are considered somewhat optimistic. These results show that in short-slim low loss environments, TTP is a viable alternative to other high performance transport protocols. TTP's benefits are much more clear in the next set of results.

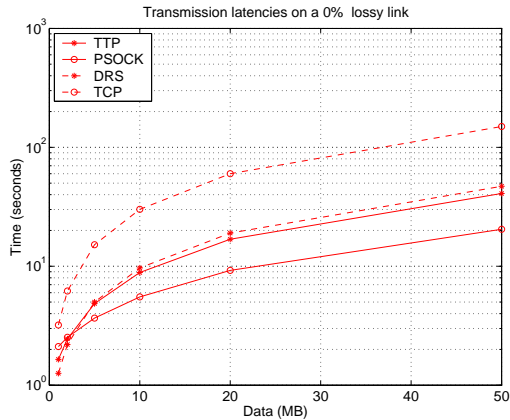


Fig. 5. Throughput comparisons under 0% loss rate for four different transport alternatives in the short-slim network topology.

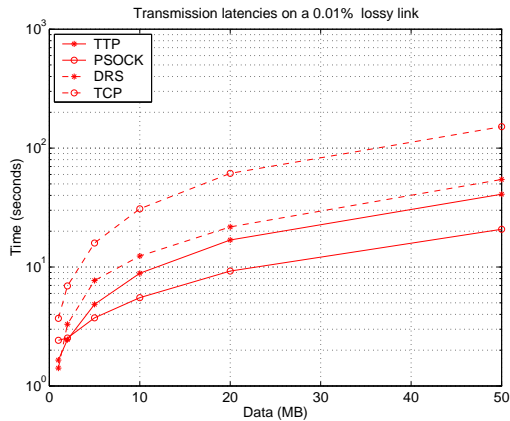


Fig. 6. Throughput comparisons under 0.01% loss rate for four different transport alternatives in the short-slim network topology.

Next, we evaluated throughput in the long-fat topology and the results are shown in Figures 10 to 14. TTP's performance benefits become very obvious in this environment. In all cases, TTP performs better than the other protocols. In each experiment, the ordering between TTP, PSOCK and DRS is the same. For the largest file transferred, TTP outperforms TCP by more than an order of magnitude. The

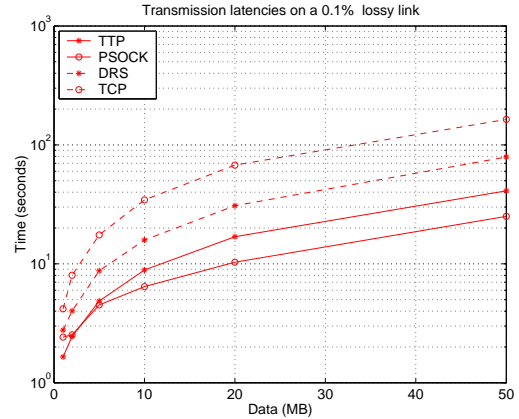


Fig. 7. Throughput comparisons under 0.1% loss rate for four different transport alternatives in the short-slim network topology.

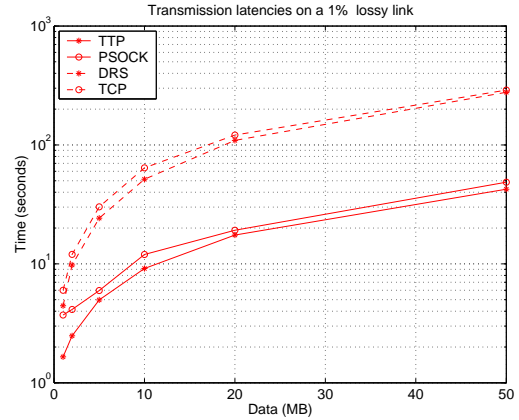


Fig. 8. Throughput comparisons under 1% loss rate for four different transport alternatives in the short-slim network topology.

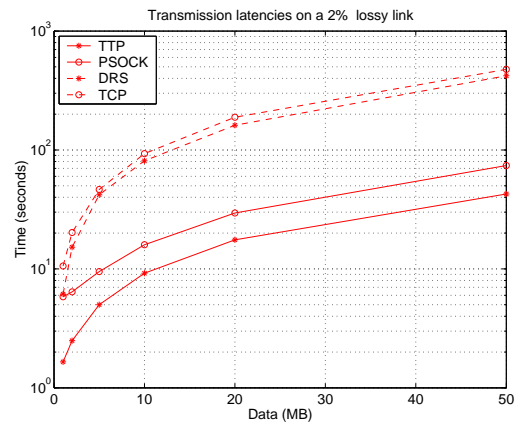


Fig. 9. Throughput comparisons under 2% loss rate for four different transport alternatives in the short-slim network topology.

graphs show that TTP typically out-performs DRS by more than an order of magnitude and PSOCK by more than 300%.

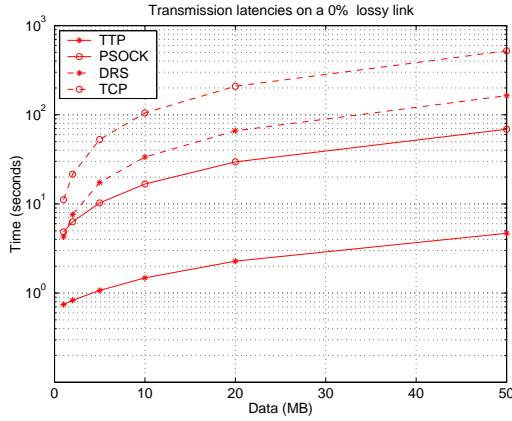


Fig. 10. Throughput comparisons under 0% loss rate for four different transport alternatives in the long-fat network topology.

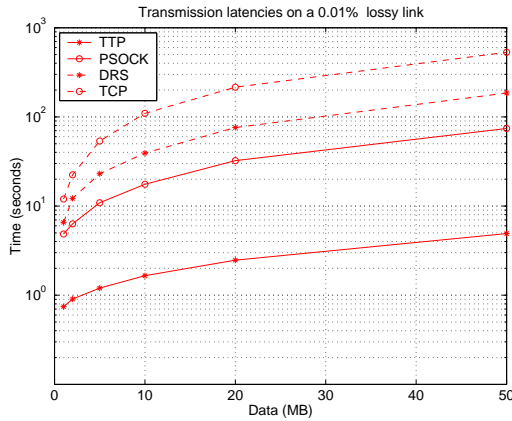


Fig. 11. Throughput comparisons under 0.01% loss rate for four different transport alternatives in the long-fat network topology.

### C. TCP Friendliness Evaluation

TCP-friendliness – the ability to adapt to congestion conditions in approximately the same way as TCP – is an important feature of transport protocols. Figure 15 shows the behavior of TTP under congestion conditions. The point marked by the arrow is an instance at which congestion is detected (3 consecutive packets lost) after which TTP reduced its transmission rate by half. While TTP is not specifically designed to be TCP-friendly in the most strict sense, it is designed to be sensitive to congestion.

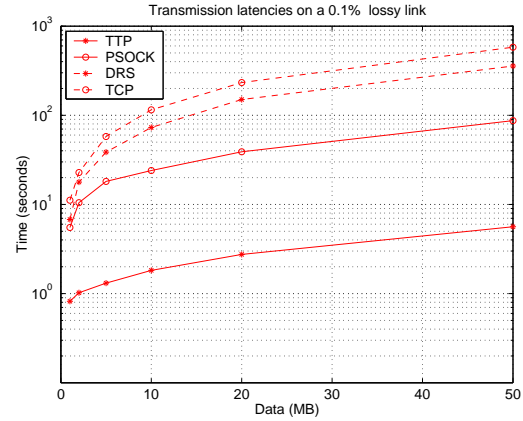


Fig. 12. Throughput comparisons under 0.1% loss rate for four different transport alternatives in the long-fat network topology.

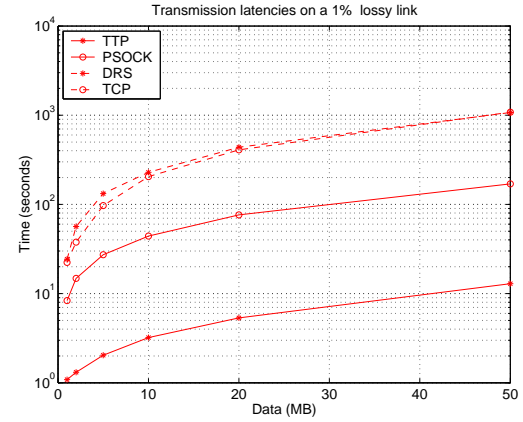


Fig. 13. Throughput comparisons under 1% loss rate for four different transport alternatives in the long-fat network topology.

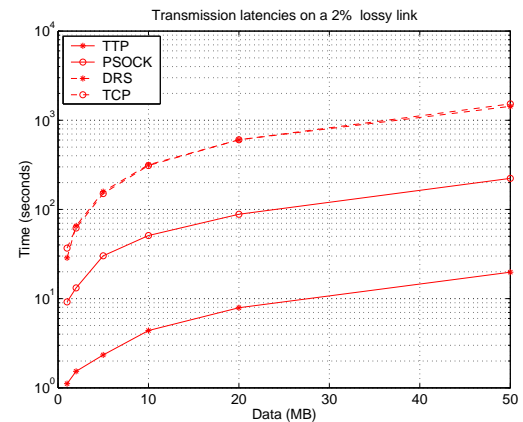


Fig. 14. Throughput comparisons under 2% loss rate for four different transport alternatives in the long-fat network topology.

There are a number of parameters which can be used to increase or decrease TTP's reaction to congestion. Our intention in designing TTP was to enable it to be tuned to a specific channel's typical congestion characteristics.

We conducted a number of experiments in which the consecutive loss threshold  $q$  was altered in the short-slim environment. We found that even in the 1% loss case, the number of instances of loss events consisting of more than 3 consecutive packets was very limited. Thus, our conjecture is that, in fact, TTP is not particularly sensitive to this parameter in very low loss environments. However, if loss rates were to increase, sensitivity to this parameter would likely increase.

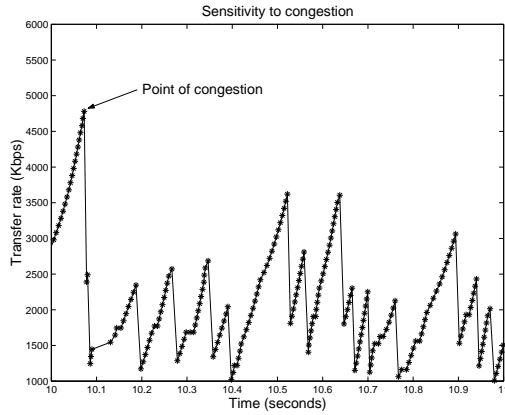


Fig. 15. TTP backoff behavior in a congested network

In order to evaluate the impact of using TTP on other TCP flows, we studied the transmission latencies of TCP flows against TTP flows in the long-fat environment. First, we measured the transmission latencies of a TCP flow when another TCP flow was competing against it for network bandwidth. Then, we measured the same TCP flow's transmission latencies when a TTP flow was competing against it for the network bandwidth. These measurements were taken for different file sizes and link loss rates. The comparison of the transmission time of the TCP flow in a TCP-TCP environment and TCP-TTP environment is shown in Figures 16 to 20. It can be seen that the transmission time of the TCP flow does not differ significantly between TCP-TCP and TCP-TTP flows in a non-lossy environment. Furthermore, there is very little difference for the 0.01%, 0.1%, 1.0% and 2% loss rates.

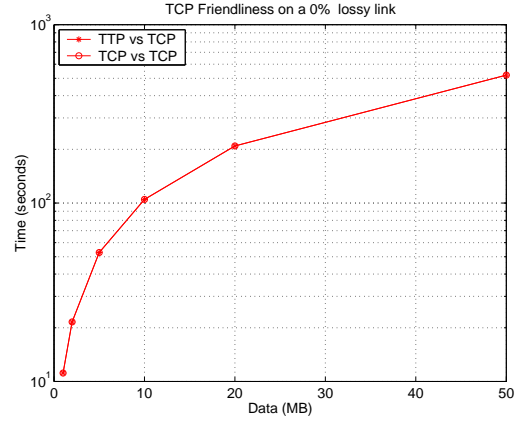


Fig. 16. TCP friendliness of TTP under 0 % loss level in the long-fat topology

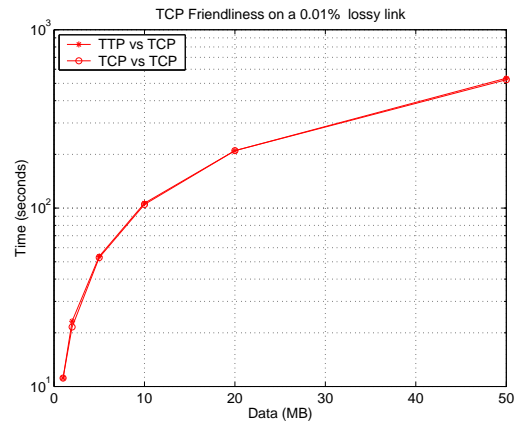


Fig. 17. TCP friendliness of TTP under 0.01 % loss level in the long-fat topology

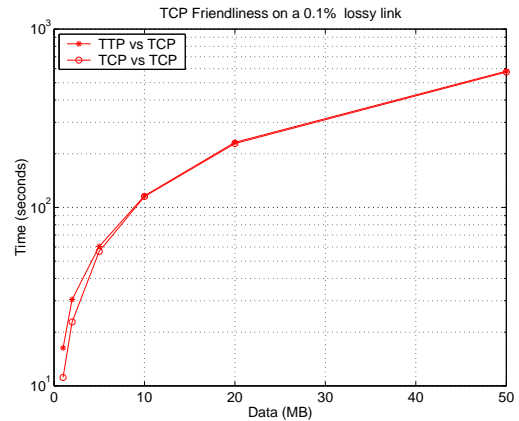


Fig. 18. TCP friendliness of TTP under 0.1 % loss level in the long-fat topology

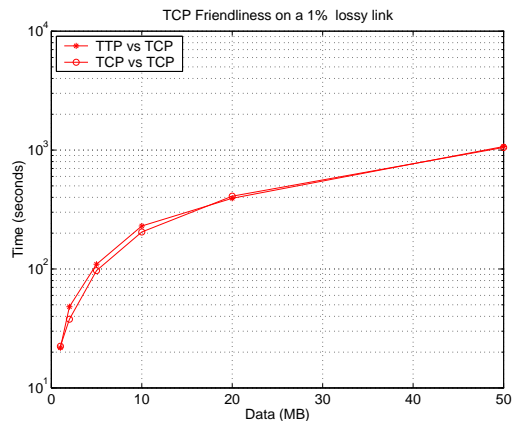


Fig. 19. TCP friendliness of TTP under 1 % loss level in the long-fat topology

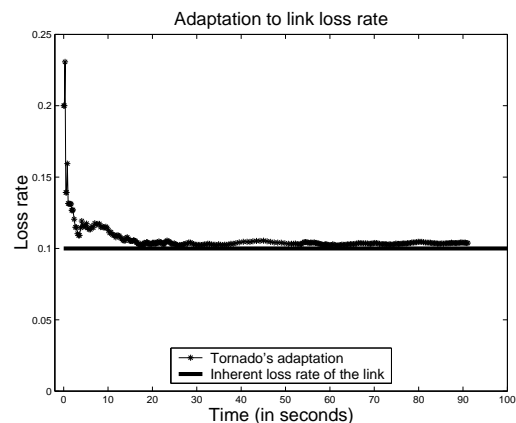


Fig. 21. Speed at which TTP stabilizes its encoding level based on link loss rate

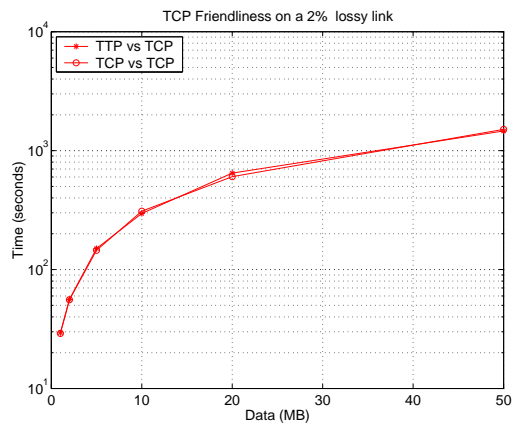


Fig. 20. TCP friendliness of TTP under 2 % loss level in the long-fat topology

#### D. Adaptation to Loss Rate

TTP's design includes the capability to adapt to loss conditions based on the link loss rates calculated at the sender. This measure determines the amount of encoding that will be used for each chunk, thus it is important that TTP be reasonably accurate in its estimation. To study this question, we adjusted the link loss rate over the duration of a large file transfer to be 10% in the short-slim topology. We selected this very high loss rate so that we could assess TTP in a scenario with higher variability. We set TTP to initially encode for 20% loss. Figure 21 shows that TTP adapts to the link loss rate after about the first 15% of the total transfer time. There are some minor transients beyond this time based on bursts of losses for individual chunks.

## IV. CONCLUSIONS AND FUTURE WORK

In this paper we describe the Tornado Transport Protocol. It is designed to facilitate very high speed transfers in large bandwidth environments with low loss characteristics. It achieves reliability through the use of forward error correction; in our evaluation we use Tornado codes although other encoding methods could certainly be used. TTP is also tunably TCP-friendly in the sense that one can specify when it will multiplicatively decrease in the face of loss.

We evaluate the performance of TTP in a simple simulation environment. Our evaluation compares TTP against out-of-the-box TCP NewReno as well as two high speed TCP implementations; Pockets and Dynamic Rightsizing using two different network topologies. We show that in long delay, large bandwidth environments, TTP outperforms TCP NewReno by nearly an order of magnitude and under low loss conditions outperforms the next best high performance TCP by nearly 300%. Our simulations indicate that TTP might well be a good candidate for the increasing number of applications that demand extremely high performance in wide area networks.

Next steps in this work include development of an actual implementation of TTP and controlled laboratory testing of the protocol. Further tests will include wide area evaluation in high performance networks such as Abilene.

## ACKNOWLEDGEMENTS

The authors would like to thank Andrea Arpaci-Dusseau for her input and support of this work. We would also like to thank Larry Landweber for fruitful discussions.

## REFERENCES

- [1] V. Jacobson and R. Braden, "TCP extensions for long delay paths," IETF RFC 1072, October 1988.
- [2] V. Jacobson, R. Braden, and L. Zhang, "TCP extensions for high speed paths," IETF RFC 1185, October 1990.
- [3] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," IETF RFC 1323, May 1992.
- [4] S. Floyd, "HighSpeed TCP for large congestion windows," IETF Internet Draft, June 2002.
- [5] T. Dunigan, F. Fowler, N. Rao, and B. Wing, "The Net100 project," <http://www.csm.ornl.gov/dunigan/net100/index.html>, 2002.
- [6] P. Gevros, F. Risso, and P. Kirstein, "Analysis of a method for differential TCP service," in *Proceedings of IEEE Globecom '99*, Rio de Janeiro, Brazil, December 1999.
- [7] H. Sivkumar, S. Bailey, and R. Grossman, "PSockets: the case for application level network striping for data intensive applications using high speed wide area networks," in *SC2000 High Performance Network and Computing Conference*, Dallas, Texas, November 2000.
- [8] T. Hacker, B. Athey, and B. Noble, "The end to end performance effects of parallel TCP sockets on a lossy wide-area network," in *Proceedings of the 16th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, CA, April 2001.
- [9] The Web100 Project, "[http : //www.web100.org](http://www.web100.org)," 2002.
- [10] E. Weigle and Wu chun Feng, "Dynamic right sizing: A simulation study," in *Proceedings of the 10th IEEE conference on Computer Communications and Networks*, October 2001.
- [11] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, September 1998.
- [12] L. Rizzo, "pgmcc: A TCP-friendly single-rate multicast congestion control scheme," in *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, September 2000.
- [13] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, August 2002.
- [14] J. Mahdavi and S. Floyd, "TCP-friendly unicast rate-based fbw control," Technical note to the end2end mailing list, January 1997.
- [15] H. Lundqvist and G. Karlsson, "TCP with forward error correction," Unpublished Manuscript, 2002.
- [16] UCB/LBNL/VINT Network Simulator - ns (version 2), "<http://www.isi.edu/nsnam/ns/>," 2000.
- [17] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.
- [18] A. Bestavros and K. Kim, "TCP Boston: A fragmentation-tolerant TCP protocol for ATM networks," in *Proceedings of IEEE INFOCOM '97*, Kobe, Japan, April 1997.
- [19] D. Bakin and M. Joa-Ng and A. McAuley, "Quantifying tcp performance improvements in noisy environments using protocol boosters," in *Proceedings of the Fifth IEEE Symposium on Computer Communications*, 2000.
- [20] J. Plank, "A tutorial on reed-solomon coding for fault tolerance in RAID-like systems," [http : //www.cs.utk.edu/plank](http://www.cs.utk.edu/plank), February 1999.
- [21] The Surveyor Project, "<http://www.advanced.org/csg-ippm/>," 1998.
- [22] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: a global internet host distance estimation service," *IEEE/ACM Transactions on Networking*, vol. 9(5), October 2001.
- [23] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of Internet path properties," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop '01*, San Francisco, November 2001.