On the Performance and Robustness of Managing Reliable Transport Connections

Gonca Gursun Ibrahim Matta Karim Mattar
Computer Science, Boston University Department
Boston University, Boston, MA 02215
{goncag, matta, kmattar}@bu.edu

ABSTRACT

We revisit the problem of connection management for reliable transport as part of our clean-slate Recursive INternet Architecture (RINA) [5]. At one extreme, a pure soft-state (SS) approach (as in Delta-t [14]) safely removes the state of a connection at the sender and receiver once the state timers expire without the need for explicit removal messages. And new connections are established without an explicit handshaking phase. On the other hand, a hybrid hard-state/soft-state (HS+SS) approach (as in TCP) uses both explicit handshaking as well as more limited timer-based management of the connection's state. In this paper, we consider the worst-case scenario of reliable single-message communication. Using simulation, we evaluate various approaches in terms of correctness (with respect to data loss and duplication) and robustness to bad network conditions (high message loss rate and variable channel delays). Our results show that the SS approach is more robust, and has lower message overhead and higher goodput. Thus, SS presents the best choice for reliable applications, especially those operating over bandwidth-constrained, error-prone networks. This result also suggests that within a clean-slate transport architecture, explicit connection messages for data reliability are not needed, and so a simple common packet interface based on Delta-t—rather than TCP vs. T/TCP vs. UDP, *etc.*— can be provided to support both transactional and bulk, reliable and unreliable (unacknowledged) applications.

1. INTRODUCTION

Reliable end-to-end transport communication has been studied since the 70's and various mechanisms have made their way into TCP [11], the reliable transport protocol widely used on the Internet today. Many of these mechanisms provided incremental patches to solve the fundamental problems of data loss and duplication. Richard Watson in the 80's [14] provided a fundamental theory of reliable transport, whereby connection management requires only timers bounded by a small factor of the Maximum Packet Lifetime (MPL). Based on this theory, Watson *et al.* developed the Delta-t protocol [6], which we classify as a pure soft-state (SS) protocol – *i.e.*, the state of a connection at the sender and receiver can be safely removed once the connection-state timers expire without the need for explicit removal messages. And new connections are established without an explicit handshaking phase. On the other hand, TCP uses both explicit handshaking as well as more limited timer-based management of the connection's state. Thus, TCP's approach, including variants such as HULA [9] and T/TCP [10], can be viewed as a hybrid hard-state/soft-state (HS+SS) protocol.

Motivated by the design of a clean-slate network architecture [5], we had to question the design of every aspect of the current Internet architecture. In this paper, we question a specific design aspect of TCP, that of connection management for data reliability: *Despite Watson's theory, why does a popular transport protocol, like TCP, manage its connections using both a state timer at the sender as well as explicit connection-management messages for opening and closing connections?*

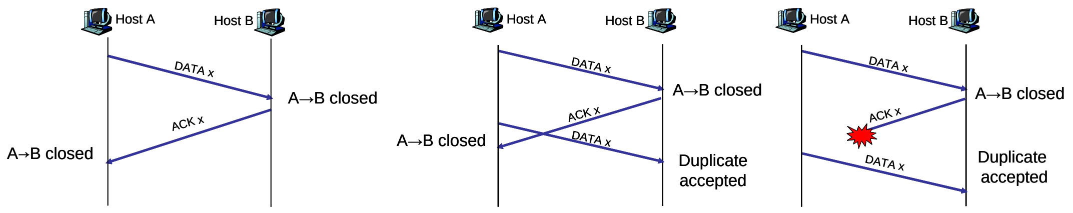
Note that connection management is concerned with maintaining consistent view of connection-states at the sender and receiver to distinguish new from old connections. Though connection management may leverage data and acknowledgements to piggyback signaling information, and so data may be falsely acknowledged (data loss) or duplicated, it is a *separate* function from data-transfer functions such as congestion control, error control, flow control, *etc.* Besides data reliability, later in 1996, TCP's three-way handshake got overloaded with protection mechanisms against transport-level attacks [3]. This results in the coupling of two different mechanisms, state synchronization for reliability and security. In our RINA architecture, we decouple these two mechanisms from each other. How the security issues are handled in RINA is out of the scope of this paper and discussed in detail in [4]. In this paper, we focus only on connection management for reliability, assuming single-message communication.

Though over a decade ago, we have seen many pioneering work in the area of reliable transport—see [13, 2, 6, 14, 12] for examples—this body of work has focused on the correctness aspects of reliable delivery but not performance. From the correctness point of view, Watson's theory states that one can achieve reliability using an SS approach, as long as one can bound exactly three timers for: (1) the maximum time that a sender expends retransmitting a data packet (G), (2) the maximum time that an acknowledgment is delayed by the receiver (UAT), and (3) the maximum time that a packet is allowed to live inside the network (MPL). Watson argues that all these times are naturally bounded in actual implementations. And since G and UAT are typically much smaller than MPL, connection-state timers (at both sender and receiver) can be bounded by a small factor of MPL. Note that TCP itself, despite its use of explicit connection-management messages, uses a connection-state timer (at the sender). And TCP *has to* use such a state timer in order to operate correctly[1]. Thus, from a correctness point of view, there is no way around the need for state timers, only that TCP relies on less of them.

Our Contribution:

From a performance point of view, to the best of our knowledge, there is no work that compares the hybrid HS+SS approach of TCP against the arguably simpler SS approach of Delta-t. In this paper, we provide a first performance comparison study. We consider the worst-case scenario of reliable *single-message* communication and study issues related to data loss / abort / duplication due to inconsistent connection-states at the sender and receiver or failure to establish a connection. Using simulation, we evaluate four approaches to reliable transport in terms of correctness (with respect to data loss and duplication) and robustness to bad network conditions (high message loss rate and variable channel delays). Our results show that the SS approach is more robust, and has lower message overhead and higher goodput. By *robustness*, we mean that performance does not precipitously degrade under worse loss/delay conditions [8]. Thus, SS presents the best choice for reliable applications, especially those operating over bandwidth-constrained, error-prone networks. This result also suggests that within a clean-

[1] Obviously, this full-proof correctness assumes that the MPL guarantee from the underlying network is not violated. Otherwise, one can only show correctness with high probability.



(a) Normal case (b) Premature retransmission (c) Acknowledgment loss

Figure 1: Two-Packet Protocol

slate transport architecture, explicit connection messages for data reliability are not needed, and so a simple common packet interface based on Delta-t—rather than TCP vs. T/TCP vs. UDP, *etc.*—can be provided to support both transactional and bulk, reliable and unreliable (unacknowledged) applications.

Organization of the Paper:

The rest of the paper is organized as follows: Section 2 reviews various approaches to reliable transport. Section 3 presents our simulation model and results comparing four reliable transport approaches (including Delta-t and TCP) under varying packet loss probability, and varying channel delays that may cause premature retransmissions. We review related work in Section 4, and conclude the paper in Section 5.

2. RELIABLE TRANSPORT APPROACHES

We describe the basic operation of different reliable transport approaches for the worst-case scenario of reliably sending a single message per conversation between a single sender and a single receiver, over a channel that may lose or re-order messages.[2] We say “worst case” since information from successive packets in a stream can only help the transport protocol, *e.g.*, to identify a missing packet in the stream sequence or to keep the connection state alive (refreshed).

In what follows, we review four approaches to reliable transport [2] that we evaluate in this paper. They represent a spectrum of solutions where the amount of explicit connection-management messages and the use of connection-state timers vary: (1) the *two-packet* (DATA and its ACK) protocol has no connection-state timers nor explicit connection-management messages, (2) the *three-packet* protocol augments the two-packet protocol with an explicit connection-management CLOSE message, (3) the *five-packet* (TCP) protocol augments the three-packet protocol with explicit connection-management (SYN and SYN+ACK) messages and a connection-state timer at the sender, and (4) the *Delta-t* protocol augments two-packet using only connection-state timers at both the sender and receiver. Delta-t and its predecessor (two-packet) represent soft-state protocols, three-packet represents a hard-state protocol, whereas five-packet represents a hybrid hard-/soft-state protocol.

Note that although, from a correctness standpoint, we note below that two-packet and three-packet may result in duplicate connections being accepted, we include them in our study to quantify, from a performance standpoint, how much relative duplication they may cause for the benefit of a simpler connection management.

Due to lack of space, we refer the reader to [7] for detailed pseudo-codes (protocol state machines) of all protocols.

2.1 Two-Packet Protocol

To detect data (packet) loss, this protocol uses positive acknowledgments. When there is data to send, the sender opens a connection to the receiver and transmits the data message. Opening a connection means that control information is kept about the connection, which we refer to as *state information*. When the receiver receives the data message, it opens a connection, delivers the data message to the application, sends an acknowledgment message back to the sender, and immediately closes the connection. Closing the connection means removing the state information of the connection. A normal conversation is illustrated in Figure 1(a).

If the sender does not receive the acknowledgment within an estimated retransmission timeout (RTO) duration, then it retransmits the data message. Figure 1(b) illustrates the case where the retransmission timeout value is underestimated, thus the sender prematurely retransmits the data message. Since the receiver closes the connection right after it sends the acknowledgment, it can not distinguish a premature retransmission (duplicate) from new data (new connection). Thus, the receiver accepts and delivers a duplicate to the application.

Another scenario that causes data duplication is when the network (channel) loses the acknowledgment. Figure 1(c) illustrates this case. If the acknowledgment is lost, the sender retransmits the data message after RTO.

In [2], the correctness of the two-packet protocol is studied in detail, including the case of data messages falsely acknowledged (*i.e.*, without being actually delivered) and hence lost. This latter problem is solved by introducing sequence numbers [13]. The sender appends to each new data message a new sequence number that has not been recently used in its communication with the receiver. A sequence number is not re-used until all messages with that sequence number (including duplicates) have left the network. Note that this *implicitly* requires knowledge of some Maximum Packet Lifetime (MPL) guaranteed by the network. Thus, the two-packet protocol (augmented with sequence numbers) does not lose data but may accept duplicates.

2.2 Three-Packet Protocol

To solve the duplication problem due to acknowledgment loss, this protocol augments the two-packet protocol with an acknowledgment for the ACK, which can be thought of as an explicit CLOSE connection-management message sent by the sender. When there is data to send, the sender opens a connection to the receiver and transmits the data message. When the receiver receives the data message, it opens a connection, delivers the data message to the application, sends an acknowledgment message back to the sender, and waits for the CLOSE message from the sender before clearing the connection-state. When the sender gets the acknowledgment, it transmits the CLOSE message to the receiver and closes the connection. The receiver in turn closes the connection once it gets the CLOSE message. Despite the extra CLOSE message, this protocol does not solve the duplication problem. If a delayed retransmission of a data message arrives at the receiver right after the receiver closes the connection, the receiver wrongly opens a new connection and accepts a duplicate.

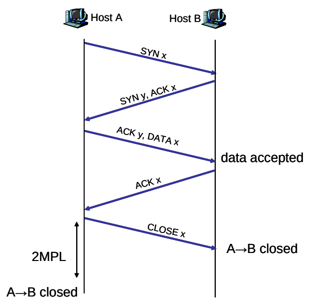
2.3 Five-Packet Protocol

To avoid data duplication, two additional explicit connection-management messages are introduced to open a connection. Fig-

[2] Throughout the paper, we use the terms “message” and “packet” interchangeably. When we refer to “single-message” or “multi-message” conversation/transfer/communication scenario, then we mean *data* messages.

ure 2(a) illustrates a normal conversation of the protocol (ala TCP). The sender transmits a synchronization SYN message to initiate the connection. The receiver responds to the SYN message with a SYN+ACK message. The sender then transmits the data message, which also acknowledges the receiver's SYN, thus synchronizing the sender and receiver, ensuring that the initial SYN message is not a duplicate (from an old connection). Upon receiving the acknowledgment for its data, the sender transmits an explicit CLOSE message and closes the connection. Upon receiving the CLOSE message, the receiver closes its end of the connection.

TCP follows this five-packet protocol. However, in TCP, after the sender sends the CLOSE message, it does not immediately close the connection, rather it waits for 2×MPL to make sure that there is no packet in the network that belongs to this connection [11].



(a) Normal case (b) Connection abort

Figure 2: Five-Packet Protocol

2.4 Delta-t Protocol

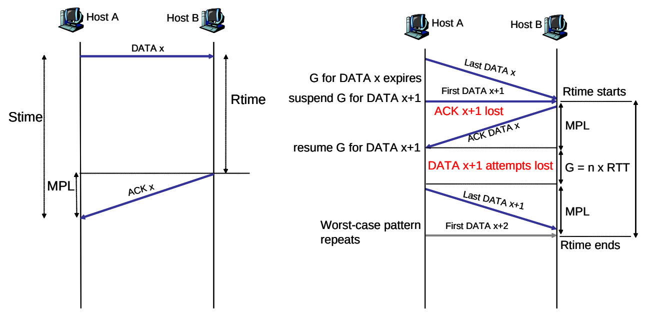
As noted above, the transport protocol inevitably assumes, either implicitly or explicitly, that the underlying network (channel) provides a guarantee on the Maximum Packet Lifetime (MPL). The Delta-t protocol [14] thus exclusively relies on connection-management (state) timers that are bounded by MPL. Delta-t is basically a two-packet protocol, augmented by state timers at both the sender and receiver to solve the problem of data duplication. Unlike the five-packet protocol, there is no explicit (separate) messages to open and close the connection.

The sender and the receiver state timers are set to guarantee that none of the messages (including duplicates) of the active connection will arrive to the ends after they close the connection. Figure 3(a) illustrates the connection state lifetime at the sender and the receiver. The sender starts its state timer whenever it sends a data message (new or retransmission). The connection at the sender should be open long enough—denoted by $Stime$—to receive the acknowledgment, which could be transmitted in the worst-case right before the receiver state lifetime—denoted by $Rtime$—expires. Since the lifetime of a packet is bounded by MPL, we have the following relationship:

$$Stime = Rtime + MPL \tag{1}$$

The receiver starts its connection-state timer whenever it receives (and accepts) a new data message. The receiver state timer should be running long enough to receive all possible retransmissions of the data message in the presence of an unreliable (lossy) channel. This allows the receiver to catch (recognize) duplicates of the data message. The connection is closed at the receiver after the last possible acknowledgment for the connection is sent. Figure 3(b), reproduced from [6], illustrates the worst-case multi-message conversation between the sender and receiver[3]. Denote by G, the maximum time a sender keeps retransmitting a data message before it gives up and aborts the connection. If n is the maximum number of retransmissions for each data message, then $G = n \times RTO \approx n \times RTT$. According to the Delta-t protocol [6], each data packet has a timer initialized to G when it is first transmitted. Whenever a data packet's G-timer expires, the G-timers of all other data packets are frozen hoping to successfully get the acknowledgment, otherwise the connection is aborted and the application is informed.

[3] For simplicity, we assume that the receiver does not delay sending its acknowledgment.



(a) State Timers (b) Setting Timers

Figure 3: Delta-t Protocol

Figure 3(b) shows the multi-message scenario when a new data packet (whose sequence number is $x+1$) is received instantly, so in the worst case, $Rtime$ is started as early as possible. Due to consecutive losses, the G-timer of the previous data packet (whose sequence number is x) *expires* while waiting for the acknowledgment ACK x for its last retransmission attempt, which in the worst case, will take MPL to arrive. At this time instant, Delta-t [6] freezes the G-timers of all oustanding packets, thus data packet $x+1$ has not yet used up its maximum delivery time G. Now when ACK x arrives, in the worst case, due to ACK losses, data packet $x+1$ keeps getting retransmitted until all its G is consumed by the time its last retransmission is sent, which in the worst case, takes another MPL to arrive at the receiver. This worst-case pattern repeats with data packet $x+2$, which causes the receiver's state timer to be re-started (refreshed). Given this worst-case scenario, a Delta-t receiver sets its $Rtime$ as follows:

$$Rtime = 2 \times MPL + G \tag{2}$$

Thus, substituting $Rtime$ in Equation (1), we have:

$$Stime = 3 \times MPL + G \tag{3}$$

3. SIMULATION

3.1 Simulation Model

We use event-based simulations to compare four protocols—two-packet, three-packet, five-packet and Delta-t—in terms of correctness, robustness and performance.

In our simulation model, all types of messages may get lost with probability p, or delayed in the underlying channel. We use a two-state Markovian channel-delay model with a short-delay state and a long-delay state. The mean of short and long channel delays are 250 and 1000 milliseconds, respectively.[4] If the channel is in the short (long) channel-delay state for a message, then with probability 0.8 it will stay in the same state for the subsequent message, or with probability 0.2 it will transit to the long (short) channel-delay state. For any message, the delay is upper bounded by the Maximum Packet Lifetime, MPL, which is set to 2 minutes.

New connections arrive according to a Poisson process at the rate of 10 connections/second. For all protocols, the sequence number for each connection is randomly chosen, uniformly from the

[4] This yields a range of RTT that is consistent with Internet measurements [1].

range [0, 10000], and we set the maximum number of retransmission attempts for *any* message to five.

In the following subsections we present and discuss our simulation results. Each plot is obtained by averaging ten independent runs, and each run attempts to establish 1000 connections. All results are shown with 95% confidence intervals—in some plots, the intervals are too small to be visible.

3.2 Summary of Main Observations

• *Delta-t is more robust than five-packet (ala TCP)* under high packet loss probability and highly variable channel delays. The extra explicit connection-management messages of five-packet make it vulnerable to connection aborts, resulting in increased percentage of aborted connections (and hence, data).

• *Delta-t yields higher goodput (rate of successfully established connections) than five-packet (ala TCP)* under high/variable packet loss/delay conditions. Thus, Delta-t can provide better support for applications that are delay-sensitive as well. On the other hand, five-packet relies on explicit connection-management (handshaking) messages to verify that a received SYN message is not a duplicate (from an old connection). This makes five-packet (ala TCP) quite vulnerable under bad network conditions.

• *Delta-t has less implementation complexity*—it has less number of protocol states[5] , and no separate connection-management messages.

• From a correctness standpoint, both Delta-t and five-packet (ala TCP) guarantee correct no-loss/no-duplication behavior. On the other hand, two-packet and three-packet can accept duplicate connections. But, from a performance standpoint, three-packet cuts the amount of duplication to about half that of two-packet at the expense of doubling message overhead. They both provide higher goodput than Delta-t and TCP, and lower message overhead compared to TCP. Thus, *if the application can handle duplicates itself, depending on the level of duplication that can be tolerated, three-packet may be more attractive than two-packet.*

3.3 Performance Metrics and Results

We consider the following metrics for evaluating the performance of the different connection management schemes. As noted in Section 1, connection management is separate from data-transfer functions such as error / congestion / flow control. However, given that connection management may piggyback signaling information over data / acknowledgements, inconsistent connection-states may result in data loss or duplication. In our scenario of single-message connections, all these metrics are to be considered connection management specific, *i.e.,* duplicate connections delivering duplicate data, or aborted connections causing application data not to be delivered may happen due to inconsistent connection-states at the sender and receiver, or failure to open a connection.

• *Percentage of Correctly Received Data*: Receiving a data message correctly means that the data message is accepted *exactly once* by the receiver. In other words, the data message was neither lost nor duplicated.

• *Percentage of Duplicate Data*: Duplicating a data message means that the receiver mistakenly accepted the data message more than once.

• *Percentage of Lost Data*: A data message is lost if it is lost in the network (channel) and an acknowledgment from a previous connection (with the same sequence number) is mistakenly associated with it.

• *Percentage of Aborted Data*: A data message is aborted (*i.e.,* not delivered to the receiving application) if it exceeds its retransmission limit, or its associated connection is aborted because the retransmission limit of any connection-management message is exceeded.

• *Message Rate*: We define it as the total number of messages sent—data, connection-management messages, acknowledgments and retransmissions—per time unit.

• *Message Overhead*: We define it as the average number of connection-management messages, acknowledgments and retransmissions sent during a connection.

• *Goodput*: We define it as the rate of *new* (unique) successfully established connections from the sender to receiver.

In the following plots, we do not show the percentage of lost data, since there was no data loss for all protocols. This is because for each connection, we use a new sequence number that is randomly chosen from a large range. That makes it *unlikely* that an (old) acknowledgment from a previous connection carries the same sequence number as a new data message that gets lost in the channel, such that it is wrongly assumed to have been successfully delivered.

To model the variability in channel delay and its impact on the estimation of round-trip time (RTT), which in turn affects the per-packet Retransmission Timeout (RTO), we assume that RTO is exponentially distributed with mean 1250 milliseconds. (This value is twice the average RTT over the simulated two-state delay channel.) We plot our performance metrics for varying packet loss probability. (Due to lack of space, we refer the reader to [7] for additional simulation results varying RTO.)

Figure 4(a) shows that as the packet loss probability increases, the percentage of correctly received data generally decreases (three-packet is the exception as we explain later). This is because the percentage of aborted messages increases due to the per-message limit on number of retransmissions. Delta-t's performance remains almost unaffected, showing very high resiliency to packet loss. On the other hand, the performance of five-packet precipitously degrades once the packet loss probability exceeds 0.25. This is because of five-packet's use of explicit connection-management messages, SYN and SYN+ACK, which when continually lost and their retransmission limit exceeded, the connection establishment fails and so data delivery is aborted.

Consistent with the correctness of Delta-t and five-packet, Figure 4(b) shows that both do not accept duplicates. For the three-packet protocol, data duplication decreases as the packet loss probability increases, since premature retransmissions that cause duplicates are lost in the channel. This behavior of three-packet results in increasing the percentage of correctly received data. On the other hand, under two-packet, the percentage of duplicate data increases as packet loss probability increases due to the loss of acknowledgments, which triggers more retransmissions and hence duplicates.

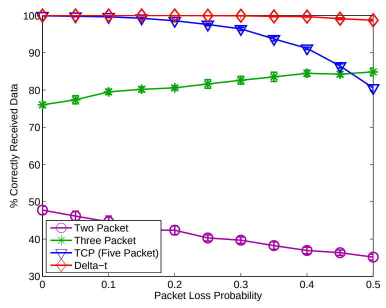
Figure 4(c) shows the probability of aborting data increases as the packet loss probability increases. This is because the sender gives up delivering a message if it continues to be lost and its retransmission limit is reached. Five-packet (ala TCP) is the least robust among all protocols.

Figure 4(d) shows that the message rate increases in all protocols as the packet loss probability increases. Five-packet protocol has the highest message rate due to explicit control messages whereas Delta-t has the lowest message rate among all protocols.

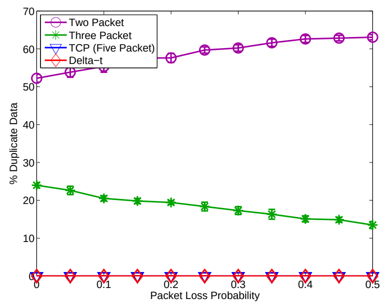
The number of messages exchanged during the lifetime of a connection is shown to increase in Figure 4(e), for all protocols, as the packet loss probability increases, because of increased retransmissions. Delta-t and two-packet have the lowest message overhead. In general, TCP can amortize the overhead of connection management messages by allowing long-lived flows and thus achieve as low overhead as Delta-t. However, this approach unnecessarily brings more complexity to connection management. Delta-t natively establishes and removes connection states in a soft-state fashion.

The goodput is shown in Figure 4(f). For all protocols, except for five-packet, the goodput does not change much as the packet loss probability increases—although time to successfully complete a connection increases, the number of concurrent active connections also increases, yielding similar goodput. On the other hand, five-packet (ala TCP) suffers from increased percentage of aborted connections (data), noticeably beyond a packet loss probability of 0.25, which results in less data being delivered to the receiving ap-

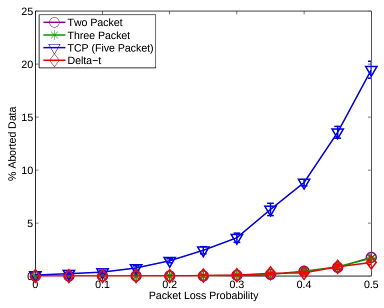
[5] Not to be confused with the states of our common analytical model, where we abstract many protocol states.



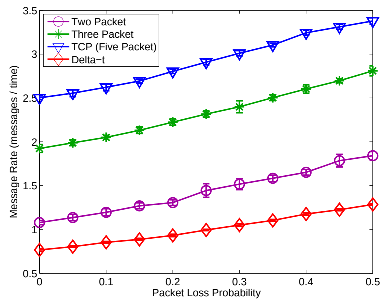
(a)



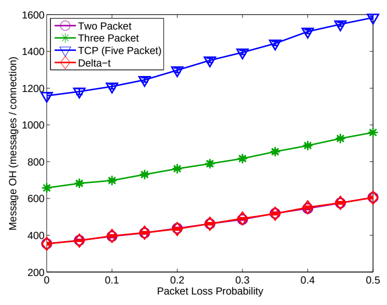
(b)



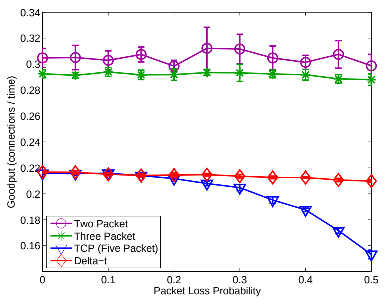
(c)



(d)



(e)



(f)

Figure 4: Effects of Varying Packet Loss Probability.

plication, yielding lower goodput.

4. RELATED WORK

Approaches to connection management for reliable transport have been studied since the 70s from a *correctness* point of view. Belsnes [2] studied the correctness of different end-to-end protocols, such as two-packet, three-packet, four-packet and five-packet (without the sender's connection-state timer). Watson [14] built on the two-packet protocol and designed Delta-t, a pure timer-based protocol for reliable connection management. TCP [11] is fundamentally a five-packet exchange protocol, with an added connection-state timer at the sender to ensure that the sender does not close the connection before the receiver does and all packets (including duplicates) have died out. Other work (*e.g.*, [12, 9, 10]) studied variants of timer-based and explicit connection-management (handshake-based) protocols, and combinations thereof, again from a correctness point of view.

None of these prior studies investigated reliable connection management from a *performance* point of view. To the best of our knowledge, this paper presents a first performance comparison across a spectrum of reliable transport solutions. We evaluated various approaches in terms of many metrics, stressing them to assess their robustness to extreme network conditions.

5. CONCLUSION

This paper presents the first performance and robustness comparison of a spectrum of reliable transport approaches, from pure soft-state (ala Delta-t), to pure hard-state (three-packet), and hybrid hard-/soft-state (ala TCP). Our results show that a soft-state (SS) approach is more robust to high packet losses and channel delay variations as it does not rely on explicit handshaking messages for opening and closing connections. An SS approach can more easily establish its connections and deliver its data reliably. Thus, an SS approach represents the best choice for reliable applications, especially those operating over bandwidth-constrained, error-prone networks. Future work includes developing a new transport architecture based on an SS approach—where explicit connection management for reliability is not needed— that exposes a simpler common packet interface than what we have today (UDP vs. TCP vs. T/TCP *etc.*) to both reliable and unreliable (unacknowledged), bulk and transactional applications.

6. REFERENCES

[1] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP Round-Trip Times. In *Proceedings of the* 3^{rd} *ACM SIGCOMM Conference on Internet Measurement (IMC'03)*, pages 279–284, New York, NY, USA, 2003. ACM.

[2] D. Belsnes. Single-Message Communication. *IEEE Transactions On Communications, Vol. COM-24*, 1976.

[3] D. J. Bernstein. http://cr.yp.to/syncookies.html.

[4] G. Boddapati, J. Day, I. Matta, and L. Chitkushev. Assesing the Security of a Clean-Slate Internet Architecture. Technical Report BUCS-TR-2009-021, CS Department, Boston University, 2009.

[5] J. Day, I. Matta, and K. Mattar. Networking is IPC: A Guiding Principle to a Better Internet. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–6, New York, NY, USA, 2008. ACM.

[6] J. G. Fletcher and R. W. Watson. Mechanisms for a Reliable Timer-Based Protocol. *Computer Networks*, 2:271–290, 1978.

[7] G. Gursun, I. Matta, and K. Mattar. On the Performance and Robustness of Managing Reliable Transport Connections. Technical Report BUCS-TR-2009-014, CS Department, Boston University, April 17 2009.

[8] J. C. S. Lui, V. Misra, and D. Rubenstein. On the Robustness of Soft State Protocols. *ICNP '04: Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 50–60, 2004.

[9] U. Maheshwari. HULA: An Efficient Protocol for Reliable Delivery of Messages. Technical report, Cambridge, MA, USA, 1997.

[10] RFC1644. T/TCP – TCP Extensions for Transactions, July 1994.

[11] RFC793. Transmission Control Protocol, September 1981.

[12] A. Shankar and D. Lee. Minimum-latency Transport Protocols with Modulo-N Incarnation Numbers. *IEEE/ACM Transactions on Networking*, 3:255–268, 1995.

[13] R. Tomlinson. Selecting Sequence Numbers. *ACM SIGCOMM/SIGOPS Interprocess Communications Workshop*, 9(3), 1975.

[14] R. Watson. Timer-Based Mechanisms in Reliable Transport Protocol Connection Management. *Computer Networks*, 5:47–56, 1981.