On the Performance and Robustness of Managing Reliable Transport Connections

GONCA GURSUN IBRAHIM MATTA KARIM MATTAR
{goncag, matta, kmattar}@cs.bu.edu
Computer Science
Boston University, MA

Technical Report BUCS-TR-2009-014
April 17, 2009

***Abstract*—**

We revisit the problem of connection management for reliable transport. At one extreme, a pure soft-state (SS) approach (as in Delta-t [9]) 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 timer-based management of the connection's state. In this paper, we consider the worst-case scenario of reliable single-message communication, and develop a *common* analytical model that can be instantiated to capture either the SS approach or the HS+SS approach. We compare the two approaches in terms of goodput, message and state overhead. We also use simulations to compare against other approaches, and evaluate them 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. On the other hand, SS requires more memory to keep connection states, which reduces goodput. Given memories are getting bigger and cheaper, SS presents the best choice over bandwidth-constrained, error-prone networks.

I. INTRODUCTION

Reliable end-to-end transport communication has been studied since the 70's and various mechanisms have made their way into TCP [6], 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 [9] 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 [2], 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 timer-based management of the connection's state. Thus, TCP's approach can be viewed as a hybrid hard-state/soft-state (HS+SS) protocol.

Given the recent interest in clean-slate network architectures, it is imcumbent on us 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:

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?

Though over a decade ago, we have seen many pioneering work in the area of reliable transport—see [8], [1], [2], [9], [7] 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 develop a common analytical model that can be instantiated to capture either the SS approach or the HS+SS (five-packet exchange) approach. This analytical model specializes the general model of Ji *et al.* [3] for signaling protocols to connection management for reliable transport. We compare the two approaches in terms of goodput, message and state overhead. We also use simulations

[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.

to compare against other approaches, and evaluate them 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. On the other hand, the cost of SS is an increase in memory requirement due to an additional connection-state timer at the receiver, which reduces goodput. Given memories are getting bigger and cheaper, SS presents the best choice over bandwidth-constrained, error-prone networks.

Organization of the Paper:

Section II reviews four approaches to reliable transport, including SS (ala Delta-t) and HS+SS (ala TCP). Section III presents a Markov model that captures the behavior of either SS or HS+SS for reliable connection management. We use this analytical model to compare SS and HS+SS. We use a more detailed simulation model in Section IV, to obtain simulation results comparing all four reliable transport approaches under varying packet loss probability, and varying channel delays that may cause premature retransmissions or violations in the MPL network guarantee. Section V reviews related work. We discuss some practical issues in Section VI, and conclude the paper in Section VII.

II. 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 [1] 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* 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* 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.

The Appendix contains the detailed pseudo-codes (protocol state machines) of all four protocols.

A. 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.

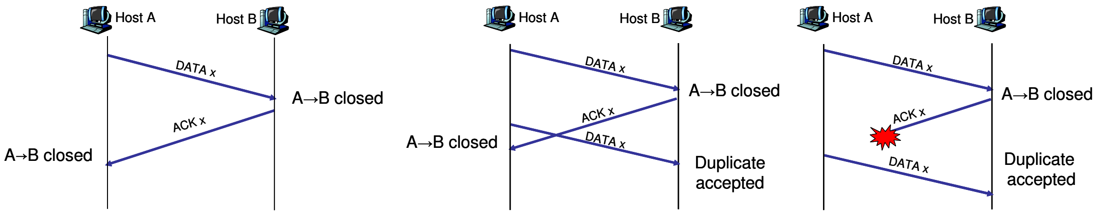
In [1], the two-packet protocol is studied in detail, including the case of data messages falsely acknowledged (*i.e.*, without being actually delivered) and hence lost. This latter problem is solved by introducing sequence numbers [8]. The sender appends to each new data message a new sequence number 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.

B. 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

[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.

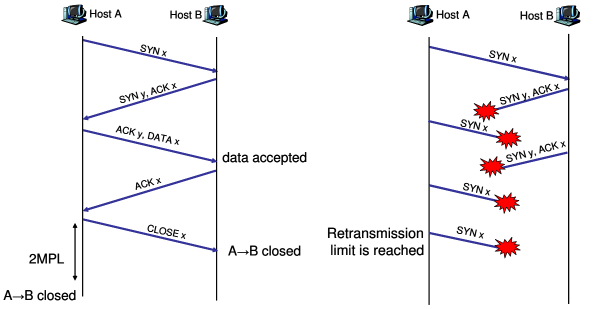


(a) Normal case (b) Premature retransmission (c) Acknowledgment loss

Fig. 1: Two-Packet Protocol

connection and accepts a duplicate.

C. Five-Packet Protocol



(a) Normal case (b) Connection abort

Fig. 2: Five-Packet Protocol

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

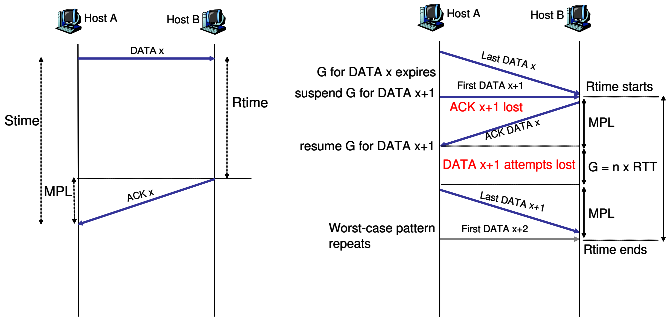
TCP follows this five-packet protocol. However, in TCP, after the sender sends the CLOSE message, it does not immediately close the connection, rather it waits for 2×MPL to make sure that there is no packet in the network that belongs to this connection [6].

D. 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 [9] 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:

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



(a) State Timers (b) Reproduced from [2]

Fig. 3: Delta-t Protocol

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 [2], 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 [2], 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

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

acknowledgment, otherwise the connection is aborted and the application is informed.

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 [2] 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:

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

III. Analytical Model

A. Model Description

In this section we develop a Markov chain model, shown in Figure 4, whose state transition rates can be instantiated to capture the behavior of either the five-packet protocol (ala TCP) or the Delta-t protocol. The ability of instantiating both protocols in a common model underscores that reliable transport approaches represent a spectrum of solutions that we should study to better understand the fundamental cost/performance tradeoffs. Our model specializes the general model of [3] for signaling protocols to connection management for reliable transport.

In our model, a state is a two-dimensional tuple representing whether the connection is established at the sender and receiver. The symbol "$\star$" denotes that state has been initialized at this end, whereas "$-$" denotes that state has not yet been installed at this end. Table I lists the parameters of the protocols and the underlying network channel. All time variables are assumed to be exponentially distributed. Table II gives the state transition rates. In our model, we assume a lossy FIFO network (channel), and that in the five-packet protocol, data is sent piggybacked on the initial SYN message. Though we capture the possible loss and retransmission of the initial message (SYN+DATA in five-packet and DATA in Delta-t), for simplicity, we assume that remaining control packets, which are much smaller in size, are not lost. Thus, we do not have to worry about receiving (and possibly accepting) duplicates at the receiver—we study this aspect by simulation later in Section IV.

- Markov state $(\star,-)_1$ captures the initial stage when the sender attempts to initialize a connection with the receiver. The sender transmits either a SYN+DATA message (in five-packet) or a DATA message (in Delta-t).

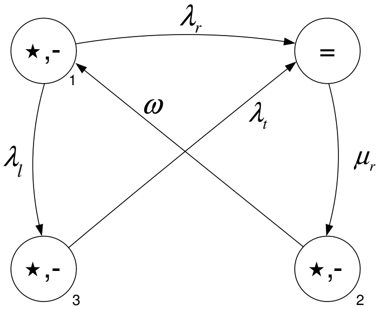


Fig. 4: Markov Model

- Markov state $(\star,-)_3$ captures the case when the sender's first attempt to initialize the connection failed. This happens when the first SYN+DATA (in five-packet) or DATA (in Delta-t) is lost. In this state, the sender keeps retransmitting the initial message. Note that this is an inconsistent state since there is no corresponding connection state yet established at the receiver.
- Markov state $=$ captures the case when the receiver gets the initial message (SYN+DATA or DATA). This is a *consistent* state where both the sender and receiver have the state information of the connection between them. Henceforth all control messages exchanged are transmitted in this state, which lasts until the receiver closes the connection.
- Markov state $(\star,-)_2$ captures the case when the connection is closed at the receiver whereas it is still open at the sender. In reliable transport protocols, to avoid inconsistency, the sender should not close the connection before the receiver does [2]. In our model, we assume that connection-state timers are set correctly so that the sender always closes after the receiver does.

TABLE I: Parameter Definitions

Parameter	Definition
p	Packet loss probability
D	Channel delay
RTO	Retransmission timeout
MPL	Maximum packet lifetime
$Rtime$	Connection-state lifetime at receiver for Delta-t
C	Connection-state lifetime at receiver for five-packet

At the initial state $(\star,-)_1$, the initial message arrives to the receiver with probability $(1-p)$ or gets lost with probability p. The first case is modeled by a transition from state $(\star,-)_1$ to $=$ with rate $\lambda_r = (1-p)/D$, where D is the channel delay. The second case is modeled by a transition from state $(\star,-)_1$ to $(\star,-)_3$ with rate $\lambda_l = p/D$. Note that both λ_r and λ_l are the same for both five-packet and Delta-t protocols.

In the $(\star,-)_3$ state, the sender keeps retransmitting the initial message. A successful retransmission causes a transition from $(\star,-)_3$ to $=$ with rate λ_t. Since the probability of successful message arrival is $(1-p)$ and the sender retransmits the message every RTO, $\lambda_t = (1-p)/RTO$. Again, λ_t is the same for both protocols.

TABLE II: Transition Rates

Transition Rates	Definition	Five-Packet Protocol	Delta-t Protocol
λ_r	Arrival rate of initial message at receiver	$(1-p)/D$	$(1-p)/D$
λ_l	Loss rate of initial message	p/D	p/D
λ_t	Successful retransmission rate of initial message	$(1-p)/RTO$	$(1-p)/RTO$
μ_r	Connection-state removal rate at receiver	$1/C$	$1/Rtime$
ω	Connection-state removal rate at sender	$1/MPL$	$1/MPL$

In the $=$ state, the sender and receiver exchange all control messages (ACK in Delta-t, and SYN+ACK, ACK and CLOSE in five-packet), completing the delivery of the data. The receiver then closes the connection and clears the connection state. We denote by $1/\mu_r$ the average lifetime of the connection state at the receiver. For the five-packet protocol, $1/\mu_r = C$, where C is the time between receiving the SYN+DATA message and the CLOSE message. For Delta-t, $1/\mu_r = Rtime$, where $Rtime = 2 \times MPL + G$ [9] (cf. Section II). Closing the connection at the receiver causes the transition from state $=$ to $(\star,-)_2$ with rate μ_r.[4]

In state $(\star,-)_2$, the sender's connection-state timer expires with rate ω. For both protocols, $1/\omega = MPL$ so that the sender does not close the connection before a last message sent by the receiver can potentially arrive—this takes, in the worst case, MPL.

In our model, we assume that there is no time between two consecutive connections. As soon as the sender closes the connection, it starts a new one which causes the transition from $(\star,-)_2$ to $(\star,-)_1$. This allows us to compute, for each protocol, the maximum rate of establishing connections (*i.e.* goodput, defined in Equation 4), by considering the message rate at state $(\star,-)_1$ where new (single data-message) connections are started.

Table II summarizes the state transition rates for five-packet and Delta-t.

B. Model Solution and Performance Calculations

Using our Markov model, we can derive the following performance metrics:

- *Goodput* ϑ: rate of successfully establishing connections, or equivalently, rate of successfully delivering data packets since we assume one data packet per connection.
- *Message rate* φ: total transmission rate of messages, including data and control messages. This metric reflects a protocol's communication and processing overhead.
- *Receiver connection-state lifetime* η: fraction of the connection lifetime during which connection-state is maintained at the receiver. This metric captures a protocol's memory requirement at the receiver.

Let π_i denote the steady-state probability of being in state i. A new connection is established when the system is in state $\pi_{(\star,-)_1}$. Therefore, for both protocols, the goodput ϑ, is computed as the message rate in the $\pi_{(\star,-)_1}$ state. Since the average message rate in this state is $\lambda_r + \lambda_l = 1/D$, then:

$$\vartheta = \pi_{(\star,-)_1}/D \tag{4}$$

The average message rate for five-packet is obtained by multiplying the probability of being in each state by the message rate at that state. In state $(\star,-)_1$, the message rate is $\lambda_r + \lambda_l = 1/D$. In state $(\star,-)_3$, the message rate is the rate of retransmitting the initial message, which is $\frac{1}{RTO}$. In state $=$, since we assume that the remaining four (control) messages of the five-packet exchange are successfully transmitted, this happens over four channel delays, thus the message rate in this state is $\frac{4}{4D} = \frac{1}{D}$. Finally, in state $(\star,-)_2$, no messages are sent since the sender simply waits for MPL before clearing its connection-state. Thus, the message rate for five-packet is given by:

$$\varphi_{five} = \frac{1}{D}\pi_{(\star,-)_1} + \frac{1}{RTO}\pi_{(\star,-)_3} + \frac{1}{D}\pi_{=} \tag{5}$$

Similarly, the message rate for Delta-t is computed as follows:

$$\varphi_{delta} = \frac{1}{D}\pi_{(\star,-)_1} + \frac{1}{RTO}\pi_{(\star,-)_3} + \frac{1}{Rtime}\pi_{=} \tag{6}$$

Note that for delta-t, in state $=$, only the acknowledgment for the initial DATA message is sent during the connection-state lifetime at the receiver, thus the message rate is $1/Rtime$.

The receiver maintains a connection-state only in the $=$ state. Given that on average, each connection lasts for $\frac{1}{\vartheta}$, and the fraction of time that the receiver has a state for that connection is $\pi_{=}$, then the connection-state lifetime at the receiver is given by:

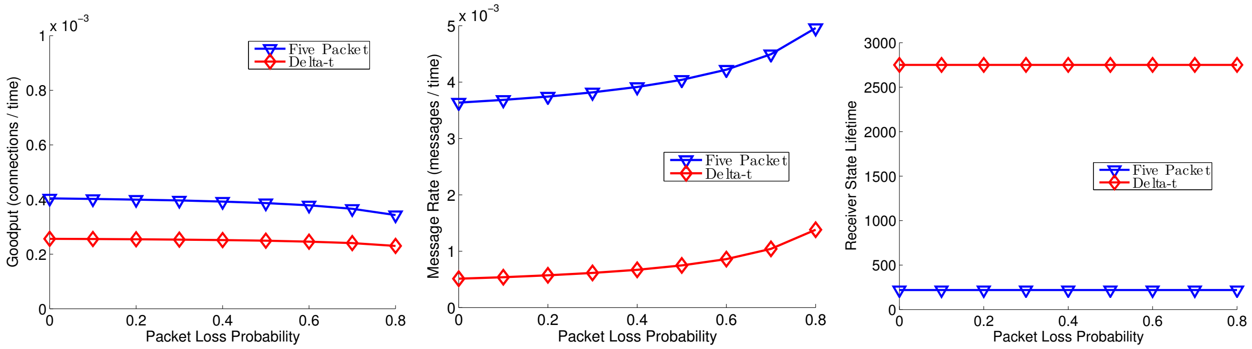
$$\eta = \frac{1}{\vartheta}\pi_{=} \tag{7}$$

C. Analytical Model Results

We show results comparing five-packet and Delta-t for the following mean parameter values: D=55 time-units, RTO=110 time-units, the connection-state lifetime at the receiver for five-packet—which starts after the initial SYN message— C=4 × D to send the remaining four messages, and MPL=$\alpha \times D$ where we set α to 20.

Figure 5 shows the performance metrics as a function of packet loss probability. Figure 5(a) shows that, as expected, goodput decreases (albeit slightly) as the packet loss probability increases. The five-packet protocol has a higher goodput than Delta-t. The reason is that under five-packet, the average lifetime of a connection is shorter, because of shorter lifetime of the connection-state at the receiver ($C < Rtime$), at the expense of explicit synchronization (connection-management) messages.

[4] The setting of μ_r and ω is what makes our model specific to connection management for reliable transport, specializing the general model of [3] for signaling protocols.



(a)

(b)

(c)

Fig. 5: Analytical Model Results.

As expected, message rate is directly proportional to packet loss probability as illustrated in Figure 5(b). The message overhead is higher with five-packet due to the extra explicit connection-management messages.

Given that in our model we assumed that the only message that can get lost is the initial message (SYN+DATA in five-packet and DATA in Delta-t) of the connection, once the initial message is successfully received, the connection-state lifetime at the receiver is not affected by the packet loss probability. Figure 5(c) compares the receiver's connection-state lifetime of both protocols—the ratio of Delta-t's to that of five-packet is given by:

$$\frac{Rtime}{C} = \frac{2MPL + G}{4D} \tag{8}$$

Since typically $G \ll MPL$, and we take $MPL = \alpha \times D$, we have:

$$\frac{Rtime}{C} \approx \frac{2\alpha D}{4D} = \frac{\alpha}{2} \tag{9}$$

Thus for α = 20, the connection-state lifetime at the receiver under Delta-t is ten times that of five-packet.

In summary, this simple analysis exposes the fundamental tradeoff between message overhead and memory requirement. Delta-t has lower message overhead, but keeps connection-state longer which reduces goodput. In the following simulation sections, we relax the assumption that only the initial message is lost and consider a wide range of channel loss rates and delays.

IV. Simulation

A. 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 10 and 100 time units, respectively. 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 200 time units. We assume there is no waiting time between two successive (one-message) connections.

For all protocols, the sequence number for each connection is randomly chosen, uniformly from the 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 10000 connections. All results are shown with 95% confidence intervals—the intervals are very small due to the large number of connections we simulate.

B. Summary of Observations

Before presenting our simulation results in detail, we summarize our main observations:

- Delta-t is more robust than five-packet under high packet loss probability. By *robustness*, we mean that performance does not precipitously degrade under worse loss/delay conditions [4]. The extra explicit connection-management messages of five-packet make it vulnerable to connection aborts, resulting in increased percentage of aborted data/connections.
- The five-packet protocol is also vulnerable to delays experienced by its connection-management messages, again causing increased percentage of aborted data/connections.
- The correctness of both Delta-t and five-packet, in terms of no data loss and no duplication, is guaranteed as long as the network guarantees MPL. Although five-packet relies less on timers, it still uses a connection-state timer at the sender, which requires the MPL guarantee.
- Robustness of Delta-t comes at the price of lower goodput compared to five-packet. This is due to Delta-t's maintenance of additional connection-state at the receiver, for a long enough time period that guarantees no duplicates are accepted. 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).
- Delta-t has less implementation complexity—it has less

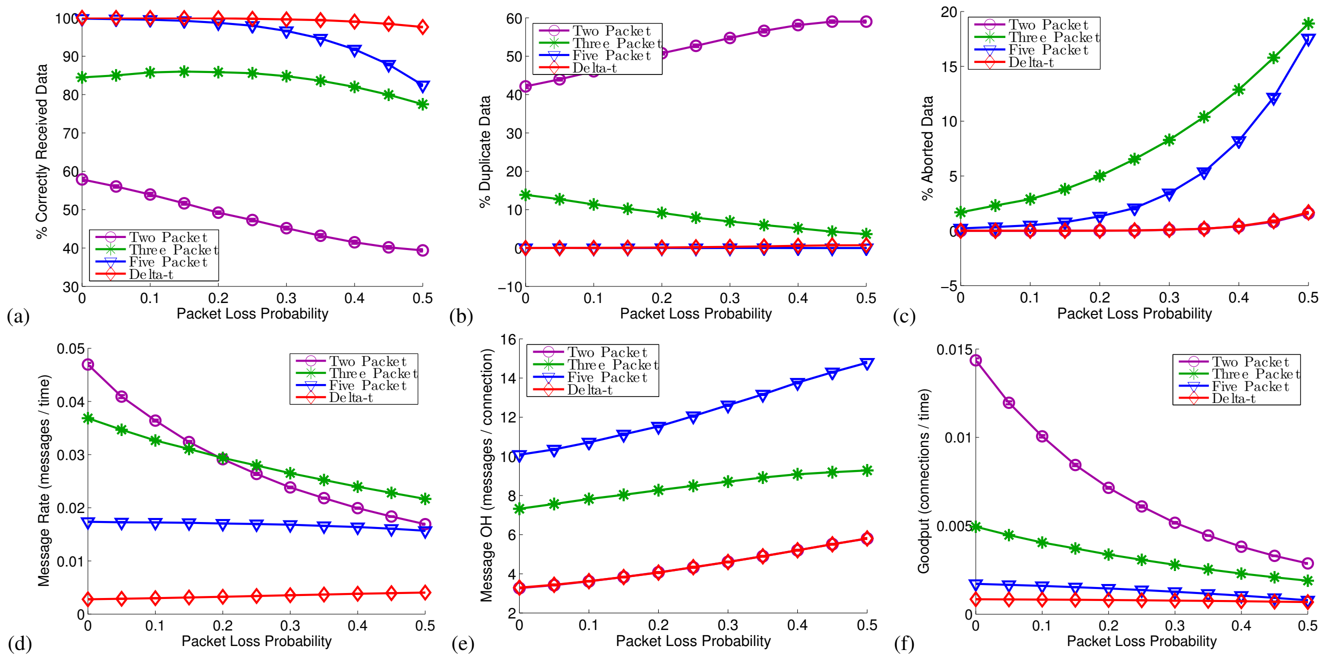


Fig. 6: Effects of Varying Packet Loss Probability.

number of protocol states[5] , and no separate connection-management messages.

- The two-packet protocol guarantees that data is not lost, but duplicates may be accepted. It has the same low message overhead as Delta-t but has higher goodput. Thus if the application can handle duplicates itself, then it may be worthwhile to use the two-packet protocol.

C. Performance Metrics

- *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) data messages delivered to the application at the 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.

D. Set 1: Effects of Packet Loss Probability

For this first set of results, 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 110 time units. (This value is the average RTT over the simulated two-state delay channel.) We plot our performance metrics for varying packet loss probability.

Figure 6(a) shows that as the packet loss probability increases, the percentage of correctly received data generally decreases. 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 0.3. 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. Figure 2(b) illustrates this scenario.

Consistent with the correctness of Delta-t and five-packet,

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

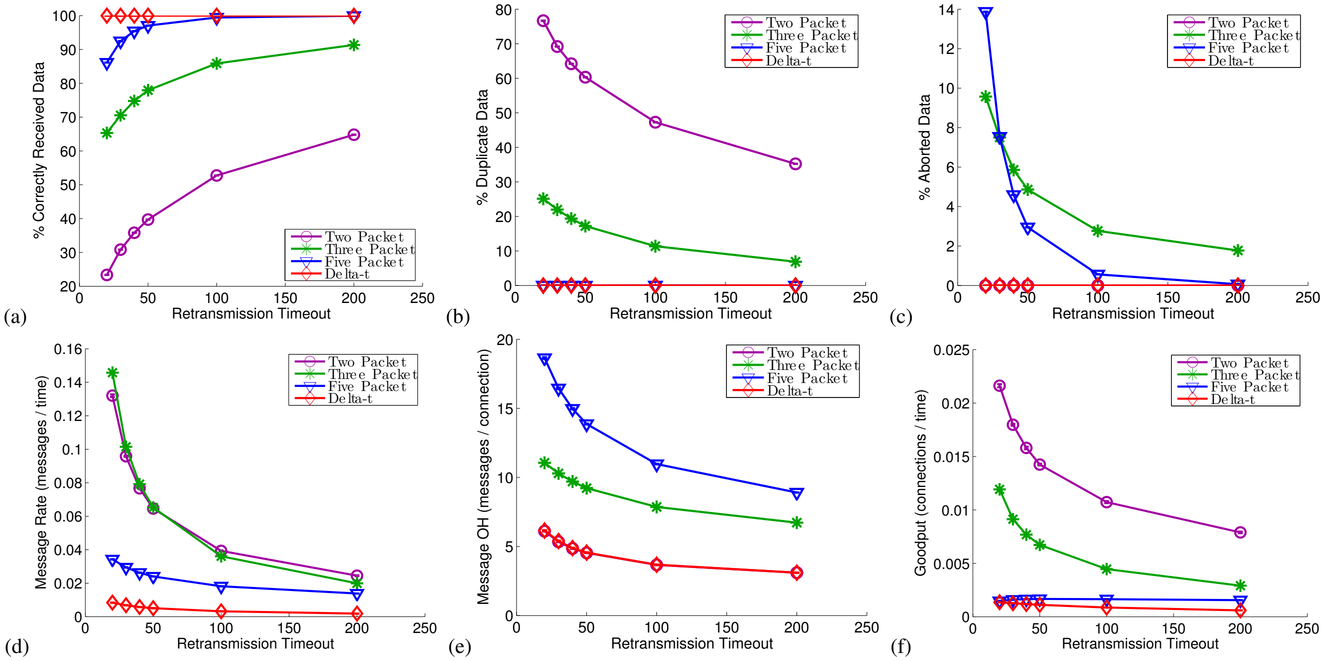


Fig. 7: Effects of Varying Retransmission Timeout.

Figure 6(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. 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 6(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. The Delta-t and two-packet protocols are the most robust among all protocols.

Figure 6(d) shows that the message rate for two-packet and three-packet decreases as the packet loss probability increases. On the other hand, the message rate stays almost constant for Delta-t and five-packet since both their average connection lifetime and number of messages exchanged during that lifetime equally increase as the packet loss probability increases.

The number of messages exchanged during the lifetime of a connection is shown to increase in Figure 6(e), for all protocols, as the packet loss probability increases, because of increased retransmissions. Delta-t and two-packet have the lowest message overhead.

The goodput is shown in Figure 6(f). For all protocols, the goodput decreases as the packet loss probability increases. This is because of (1) increased failure in establishing connections and delivering data, and (2) for those successfully established connections, their lifetime increases due to increased retransmissions. Since the average lifetime of a connection is shorter under five-packet than Delta-t, five-packet's goodput is higher.

E. Set 2: Effects of Retransmission Timeout

In this second set of results, we fix the packet loss probability p to 0.1, and we plot our performance metrics for varying RTO.

Figure 7(a) shows that, except for Delta-t, the percentage of correctly received data decreases for lower RTO (*i.e.*, when RTO is underestimated). This is because when RTO is low, there are more premature retransmissions. This increases the percentage of duplicates under two-packet and three-packet, as seen in Figure 7(b). Under five-packet, low RTO increases the percentage of aborted connections, and consequently data, as seen in Figure 7(c). This is because SYN or SYN+ACK messages get prematurely retransmitted and their retransmission limit exceeded.

Delta-t is the most resilient to underestimated RTO with respect to all performance metrics. Delta-t is least affected since a connection is opened *instantly* at the sender once the sender sends a new data message. And the receiver *instantly* opens its side of the connection once it receives the data message. From then on, the sender and receiver stay synchronized, until the state timers expire. Five-packet is only resilient to duplication (Figure 7(b)), which is expected given its provably correct no-loss/no-duplication behavior. Two-packet, like Delta-t, does not suffer from aborted connections (Figure 7(c)) since it does not rely on explicit connection-management messages.

Under all protocols, lower RTO causes premature retransmissions, which increase both the total number of messages sent (message rate in Figure 7(d)) and the message overhead (Figure 7(e)).

Figure 7(f) shows that the goodput of two-packet and three-packet slightly decreases as RTO increases due to slower

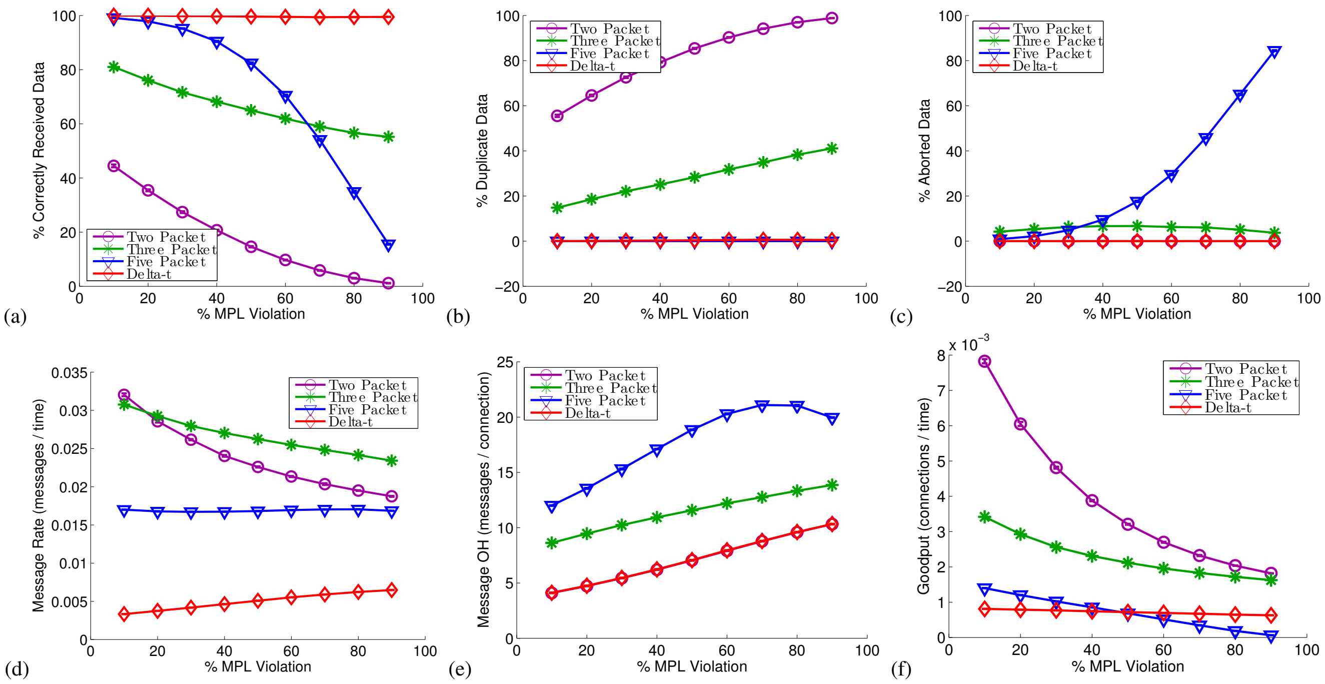


Fig. 8: Effects of Varying MPL Violation (2MPL).

reaction to losses and hence increased connection lifetime. The goodput of five-packet and Delta-t is lower than that of other protocols, especially under lower RTO values—five-packet aborts more connections (and hence data messages), whereas Delta-t maintains connection states for longer periods of time to forgo explicit connection-management (handshaking) messages.

F. Set 3: Effects of MPL Violation

In this set of results, we investigate the effects of MPL violations. We plot our metrics varying the percentage of messages that experience an excessive network delay of 2×MPL. We fix the packet loss probability p to 0.1, and mean RTO to 110 time units.

Figure 8(a) shows the high resiliency of Delta-t to MPL violations if the violation is up to 2×MPL. This is because Delta-t's connection-state timers are set conservatively (about one extra MPL worth) to also support the case of multi-message transfers. The performance of five-packet precipitously degrades due to higher number of aborted connections (Figure 8(c)), as a result of some connection establishment messages (SYN and SYN+ACK) experiencing excessive network delays (2×MPL) and reaching their retransmission limit.

Both Delta-t and five-packet behave correctly and do not accept duplicates (Figure 8(b)). As shown in Figures 8(d)–(e), similar to Figures 6(d) and (e), Delta-t, being a soft-state approach, has the lowest message rate and overhead. Unlike Figure 6(f), Delta-t's goodput gets even closer to that of five-packet under excessive network delays. When five-packet's connection-management messages experience such high channel delays, they get prematurely retransmitted and their retransmission limit exceeded, causing connection aborts.

V. Related Work

Approaches to connection management for reliable transport have been studied since the 70s from a correctness point of view. Belsnes [1] 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). Belsnes showed that all protocols may lose or duplicate messages under certain network conditions. He also concluded that if duplicates can be dealt with (or tolerated) by applications, then the two-packet protocol is the best choice since it has less message overhead.

Watson [9] built on the two-packet protocol and designed Delta-t, a pure timer-based protocol for reliable connection management. Watson discussed the need for bounding exactly three parameters: (1) a maximum packet lifetime (MPL), (2) a maximum time during which the sender keeps trying to retransmit a message (G), and (3) a maximum time before an acknowledgement is sent by the receiver (UAT). Connection-state timers at the sender and receiver are functions of these parameters. Given MPL is typically much larger than G and UAT, state timers can be simply expressed in terms of MPL.

TCP [6] 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. This implicitly assumes a network guarantee of MPL.

Other work (*e.g.*, [7], [5]) studied variants of timer-based and explicit connection-management (handshake-based) protocols, and combinations thereof, again from a correctness point of view. An example of such protocol variants has the receiver maintain limited connection state and resort to handshaking with the sender (as in TCP) only in suspected cases of duplication.

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, including purely timer-based (soft-state), purely handshake-based (hard-state), and hybrid timer-/handshake-based.

We developed a common analytical model that exposed the fundamental tradeoff between a pure timer-based approach (ala Delta-t) and a hybrid timer-/handshake-based approach (ala TCP). We evaluated various approaches in terms of many metrics, stressing them to assess their robustness to extreme network conditions. Recently, there has been great interest in understanding similar protocol design tradeoffs in a quantitative manner. Ji $et\ al.$ [3] and Lui $et\ al.$ [4] studied such tradeoffs for reservation/signaling protocols. Our work specializes the general model of [3] to connection management for reliable transport.

VI. Practical Considerations

- We found that Delta-t is most robust to severe network conditions since it does not rely on extra connection-management messages. This robustness comes at the cost of increased memory requirement for keeping connection state at the receiver. This connection state is kept for 2×MPL. Given that memories are getting bigger and cheaper, we believe this additional memory requirement is not a concern. For example, given reasonable assumptions on the connection arrival rate λ, say 10 connections per second, MPL of say, 120 seconds, a typical connection-state size S of 500 bytes, then the average *total* memory for active connections required by a Delta-t's receiver (server) is $\lambda \times S \times (2 \times MPL) = 10 \times 500 \times (2 \times 120)$ = 1.2M bytes. This memory requirement is easily accommodated given that in a typical server today, the total memory space allocated for maintaining connection states is approximately 100M bytes.
- Delta-t requires both the sender and receiver to maintain connection-state timers. These timers are only loosely coupled—they are started upon the reception of packets. Thus clock synchronization is not a problem.
- In this paper, we considered single-message communication since it constitutes a worst-case scenario in terms of message overhead and other performance metrics. We believe that our results readily extend to the case of multi-message communication.
- In our model, we assumed sequence numbers are chosen uniformly from a large range. In practice, sequence numbers may be generated from a clock [6] and so a sequence number may only be re-used after a long time (several minutes). This is especially true if sequence numbers are associated with packets and not bytes. This makes it unlikely that packets from different incarnations of the same connection would carry the same sequence number, thus creating confusion.
- Because Delta-t keeps connection states for longer periods of time, it is not able to establish connections between the same sender and receiver (*i.e.*, using the *same* connection identifier) as fast as five-packet (ala TCP). However, Delta-t is able to successfully establish almost every one of these connections, and so delivers close to 100% of its data messages. Thus, in practice, given many concurrent conversations and a large space of connection identifiers to assign them, we expect Delta-t to reliably deliver data in a much more timely manner. This would provide better support for applications that are delay-sensitive as well.

VII. Conclusion

This paper presents the first performance and robustness comparison of a spectum 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 analytical and simulation 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. The cost of such performance/robustness advantage is additional memory for connection states. Given memories are getting bigger and cheaper, 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, that exposes a simpler common interface than what we have today (UDP datagrams vs. TCP connections), to both reliable and unreliable, bulk and transactional applications.

VIII. Acknowledgment

We would like to thank John Day for various discussions on aspects of this work and for his encouragement.

This work has been partially supported by National Science Foundation awards: CISE/CCF #0820138, CISE/CSR #0720604, CISE/CNS #0524477, CNS/ITR #0205294, and CISE/EIA RI #0202067.

References

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

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

[3] P. Ji, Z. Ge, J. Kurose, and D. Towsley. A Comparison of Hard-State and Soft-State Signaling Protocols. *SIGCOMM '03*, 2003.

[4] 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.

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

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

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

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

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

Appendix

We use a notation similar to [5]. The information at the sender (receiver), denoted by s (r), contains the *protocol state*, *sequence number*, etc., which are denoted by *s.state* (*r.state*), *s.seqnum* (*r.seqnum*), etc. For each state, there are possible transitions. Transition inputs may be directives from the higher layer (application), such as Send, Receive, events such as receiving any kind of packet (msg[...], ack[...], etc.), retransmission timeout for any packet (dataPacketTimeout, ackPacketTimeout, etc.), starting or refreshing timers (startPacketTimer, refreshStateTimer, etc.), state timeout at the ends (stateTimeout), or exceeding the retransmission limit for a packet (rtxLimitExceed). Actions may be assignments such as *s.state := ...*, or events such as sending any kind of packet. Unexpected inputs are denoted by *other*.

We next show the pseudo-codes of the various transport protocols for single-message conversation.

A. Two-Packet Protocol

```
s.state = CLOSED

Send(d) → s.data := d, s.seqnum := x, s.state := MSG,
          msg[s.seqnum, s.data], startPacketTimer
other   → Error


s.state = MSG

dataPacketTimeout → msg[s.seqnum, s.data],
                    refreshPacketTimer
ack[s.seqnum]     → s.state := CLOSED
rtxLimitExceed    → s.state := CLOSED
other             → Error


r.state = DEFAULT

msg[seqnum, data] → r.seqnum := seqnum,
                    r.data := data,
                    ack[r.seqnum]
```

B. Three-Packet Protocol

```
s.state = CLOSED

Send(d) → s.data := d, s.seqnum := x, s.state := MSG,
          msg[s.seqnum, s.data], startPacketTimer
other   → Error


s.state = MSG

dataPacketTimeout → msg[s.seqnum, s.data],
                    refreshPacketTimer
ack[s.seqnum]     → ackack[s.seqnum],
                    s.state := CLOSED
rtxLimitExceed    → s.state := CLOSED
other             → Error


r.state = CLOSED

msg[seqnum, data] → r.seqnum := seqnum,
                    r.data := data, ack[r.seqnum],
                    r.state := ACKED
other             → Error


r.state = ACKED

msg[r.seqnum, data] → ack[r.seqnum]
ackack[r.seqnum]    → r.state := CLOSED
rtxLimitExceed      → r.state := CLOSED
ackPacketTimeout    → ack[r.seqnum], refreshPacketTimer
other               → Error
```

C. Five-Packet Protocol

```
s.state = CLOSED

Send(d) → s.data := d, s.seqnum := x, syn[s.seqnum],
          s.state := SYNC, startPacketTimer
other   → Error


s.state = SYNC

synack[y, s.seqnum] → s.acknum := y, s.state := MSG,
                      msg[s.seqnum, s.acknum, s.data]
synPacketTimeout    → syn[s.seqnum], refreshPacketTimer
rtxLimitExceed      → s.state := LINGER, startStateTimer,
                      close[s.seqnum]
other               → Error


ack[s.acknum, s.seqnum] s.state = MSG

ack[s.acknum, s.seqnum] → startStateTimer, close[s.seqnum]
                          s.state := LINGER
dataPacketTimeout       → msg[s.seqnum, s.acknum, s.data],
                          refreshPacketTimer
rtxLimitExceed          → s.state := LINGER,
                          startStateTimer, close[s.seqnum]
other                   → Error


s.state = LINGER

ack[s.acknum, s.seqnum] → close[s.seqnum]
stateTimeout            → s.state := CLOSED
other                   → Error


r.state = CLOSED

syn[x] → r.seqnum := y, r.acknum := x, r.state := SYNC,
         synack[r.seqnum, r.acknum], startPacketTimer
other  → Error
```

r.state = SYNC

```
msg[r.acknum, r.seqnum, data] → r.data := data,
                                r.state := ACKED,
                                ack[r.seqnum,r.acknum],
                                startPacketTimer
synackPacketTimeout           → synack[r.seqnum,r.acknum],
                                refreshPacketTimer
rtxLimitExceed                → r.state := CLOSED
other                         → Error
```

r.state = ACKED

```
msg[r.acknum,r.seqnum,data] → ack[r.seqnum,r.acknum],
                              startPacketTimer
close[r.acknum]             → r.state := CLOSED
ackPacketTimeout            → ack[r.seqnum,r.acknum],
                              refreshPacketTimer
rtxLimitExceed              → r.state := CLOSED
other                       → Error
```

D. Delta-t Protocol

s.state = CLOSED

```
Send(d) → s.data := d, s.seqnum := x,
          msg[s.seqnum, s.data], s.state := MSG,
          startPacketTimer, startStateTimer
other   → Error
```

s.state = MSG

```
dataPacketTimeout → msg[s.seqnum, s.data],
                    refreshStateTimer,
                    refreshPacketTimer
ack[s.seqnum]     → s.state := LINGER
rtxLimitExceed    → s.state := LINGER
other             → Error
```

s.state = LINGER

```
stateTimeout → s.state := CLOSED
other        → Error
```

r.state = CLOSED

```
msg[seqnum, data] → r.data := data, r.seqnum := seqnum,
                    ack[r.seqnum], r.state := LINGER,
                    startStateTimer
other             → Error
```

r.state = LINGER

```
msg[r.seqnum, data] → ack[r.seqnum], r.state = LINGER
stateTimeout        → r.state = CLOSED
other               → Error
```