On the Performance and Robustness of Managing Reliable Transport Connections

GONCA GURSUN IBRAHIM MATTA KARIM MATTAR
Computer Science Department
Boston University, Boston, MA 02215
{goncag, matta, kmattar}@cs.bu.edu

***Abstract*—We revisit the problem of connection management for reliable transport. At one extreme, a pure soft-state (SS) approach (as in Delta-t [11]) 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 a *common* analytical model that can be instantiated to capture either the SS approach or the HS+SS approach, we argue that although HS+SS may seem more attractive due to its lower memory requirement for keeping connection states, memory is not a concern in today's computers. Using a more detailed simulation model, 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.**

I. INTRODUCTION

Reliable end-to-end transport communication has been studied since the 70's and various mechanisms have made their way into TCP [8], 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 [11] 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 [3], 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 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?*

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.* In this paper, we focus only on connection management, assuming single-message communication.

Though over a decade ago, we have seen many pioneering work in the area of reliable transport—see [10], [2], [3], [11], [9] 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*. [5] for signaling protocols to connection management for reliable transport, and so in this paper, we are concerned with unique issues related to data loss / abort / duplication due to inconsistent connection-states at the sender and receiver or failure to establish a connection. The model considers a

[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.

simplified setting of a single active connection at any given time between the sender and receiver, *i.e.*, a new connection is blocked until the connection-state (memory) associated with the previous connection is released. Under this simplified setting, SS is found to have lower message overhead compared to HS+SS, at the expense of reduced goodput due to its longer holding time of the connection-state at the receiver. Thus, HS+SS may seem more attractive in terms of goodput and memory requirement. However, in today's computers, memory is not a concern. We then consider a more detailed simulation model where more than one connection can be active between a sender and receiver. 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.

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. Section V reviews related work. Section VI concludes the paper.

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 reorder messages.[2] We say "worst case" since information from successive packets in a stream can only help connection management, *e.g.*, 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* 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.

[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.

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 [4] for detailed pseudo-codes (protocol state machines) of all 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).

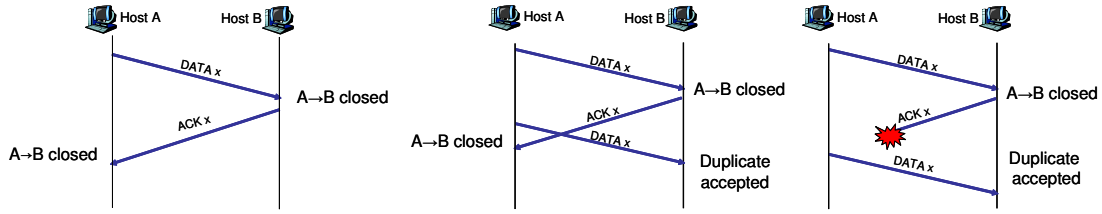
If the sender does not receive the acknowledgment within an estimated retransmission timeout (RTO) duration, then it retransmits the data message. Figure 1(b) illustrates the case where the retransmission timeout value is underestimated, thus the sender prematurely retransmits the data message. Since 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 [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 [10]. 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 reused 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,



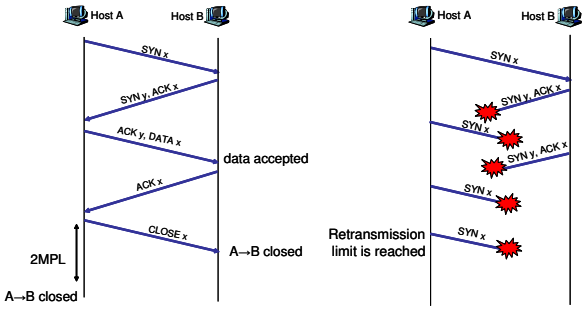
(a) Normal case (b) Premature retransmission (c) Acknowledgment loss

Fig. 1: Two-Packet Protocol

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.

C. Five-Packet Protocol



(a) Normal case (b) Connection abort

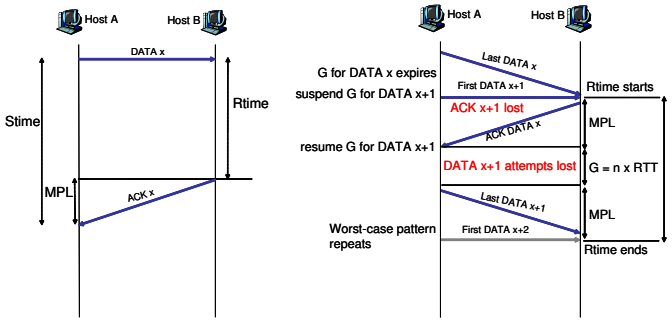
Fig. 2: Five-Packet Protocol

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 [8].

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 [11] 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. [3]



(a) State Timers (b) Reproduced from [3]

Fig. 3: Delta-t Protocol

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 [3], illustrates the worst-case multi-message conversation between

[3] Although 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.

the sender and receiver[4] . 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 $G = n \times RTO \approx n \times RTT$, where RTT denotes the round-trip time. According to the Delta-t protocol [3], 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.

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 [3] 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 signaling model of [5] to connection management for reliable transport, and so in this paper, we are concerned with unique issues related to data loss / abort / duplication due to inconsistent connection-states at the sender and receiver or failure to establish a connection. The model considers a simplified setting of a single active connection at any given time between the sender and receiver, *i.e.*, a new connection is blocked until the connection-state (memory) associated with the previous connection is released. Later, in Section IV, we remove this simplification and allow multiple active connections between a sender and receiver.

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).
- 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 [3]. 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 TCP

At the initial state $(\star, -)_1$, the initial message arrives at 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

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

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	$1/MPL$	$1/MPL$

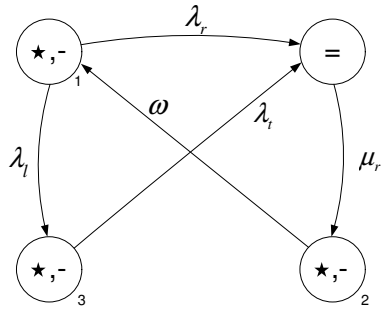


Fig. 4: Markov Model

the message every RTO, $\lambda_t = (1-p)/RTO$. Again, λ_t is the same for both protocols.

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 $Rtime = 2 \times MPL + G$ [11] (cf. Section II). Closing the connection at the receiver causes the transition from state $=$ to $(\star,-)_2$ with rate μ_r.[5]

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 this model, we assume that there is no waiting 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 $(\star,-)_1$. Therefore, for both protocols, the goodput ϑ, is computed as the message rate in state $(\star,-)_1$. 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 (*i.e.*, $C = 4 \times D$), 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

The above analytical model can be solved to obtain results comparing five-packet and Delta-t. Consider the following

[5] The setting of μ_r and ω is what makes our model specific to connection management for reliable transport, specializing the general signaling model of [5].

mean parameter values: D=250 msec, RTO=1250 msec, and $MPL = \alpha \times D$ where we set α to 480 (yielding a typical MPL value of 2 minutes).

As expected, for both protocols, we observe that message rate is directly proportional to packet loss probability, *i.e.*, message rate increases as the packet loss probability increases, because of retransmissions. (Plots are not shown due to lack of space.) The message overhead is higher under five-packet due to its extra explicit connection-management messages (five to eight times that of Delta-t).

Given that the above model assumes that the only message that can get lost is the initial message of the connection (SYN+DATA in five-packet and DATA in Delta-t), once the initial message is successfully received, the connection-state lifetime at the receiver is not affected by the packet loss probability.

Comparing 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 = 480, the connection-state lifetime at the receiver under Delta-t is 240 times that of five-packet.

Recall that the above analytical model also assumes that a new connection is blocked until the connection-state (memory) associated with the previous connection is released, *i.e.*, a single active connection is allowed at any given time between a sender and receiver. Because of this, we observe that the five-packet protocol has higher goodput than (about 1.5 times) that of 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. Also, as expected, for both protocols, we observe that goodput decreases (slightly) as the packet loss probability increases.

In summary, this simple analysis exposes a fundamental tradeoff between message overhead and memory requirement. Delta-t has lower message overhead, but keeps connection-state longer, which, under this simplified model, reduces goodput because a new connection gets blocked until memory for the connection-state of the previous connection is released. This may make an HS+SS approach (ala TCP) appear attractive. However, in a more realistic setting and given that memory is not a concern in today's computers, the conclusion might be different. So, in the next section, we consider a more detailed simulation model where: (1) we relax the assumption that only the initial message is lost and consider a wide range of channel loss rates and delays, and (2) we allow multiple connections to be active at a time between a sender and receiver. The second point is justified in practice because:

- Memory is not a concern in today's computers. 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.
- In practice, many concurrent conversations can be established between a sender and receiver given a large enough space of connection identifiers to assign them.

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 250 and 1000 milliseconds, respectively.[6] 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 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.

B. Summary of Observations

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

- *Delta-t is more robust than five-packet (ala TCP)* under high packet loss probability and low retransmission time-out values (or highly variable channel delays). By *robustness*, we mean that performance does not precipitously degrade under worse loss/delay conditions [6]. 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).
- Robustness of Delta-t comes at the price of keeping the connection-state at the sender and receiver for longer time compared to five-packet. This is to guarantee no duplicates are accepted. Since memory requirement is not a concern in today's computers, longer duration of connection-states is not an issue. And *Delta-t yields*

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

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[7] , 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.*

C. Performance Metrics

We consider the following metrics for evaluating the performance of the different connection management schemes. As noted in Section I, 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.

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 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.

Figure 5(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. Figure 2(b) illustrates this scenario.

Consistent with the correctness of Delta-t and five-packet, Figure 5(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 5(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 5(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 5(e), for all

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

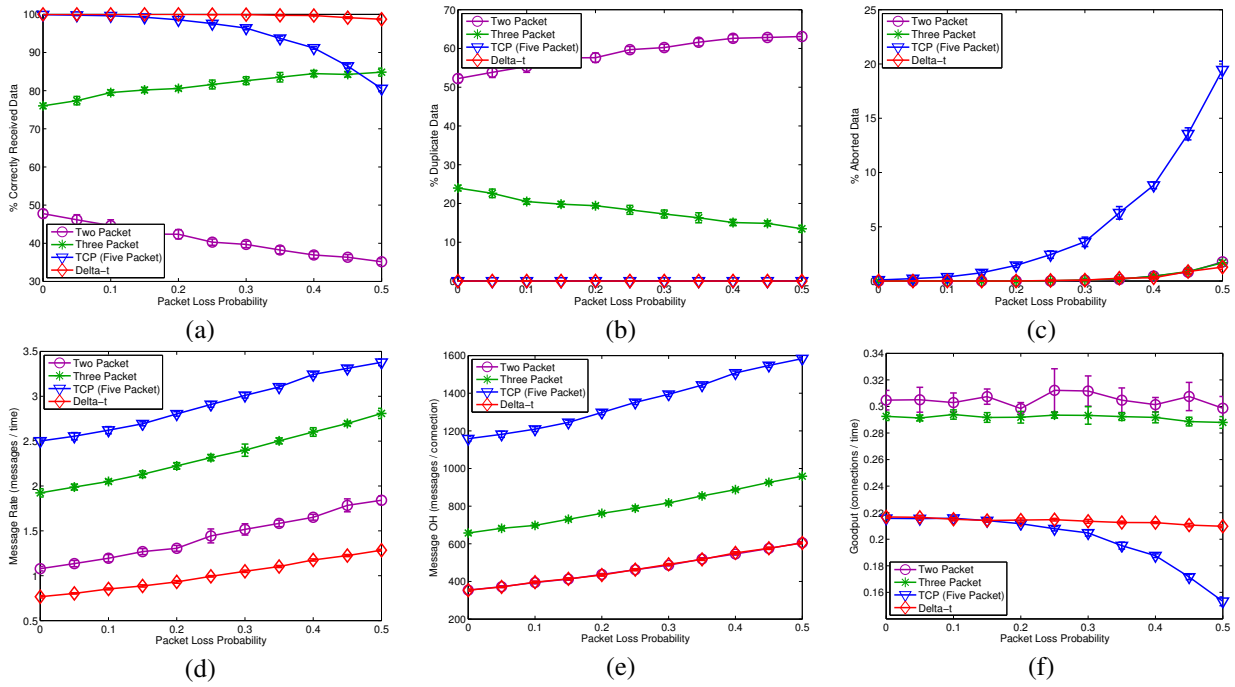


Fig. 5: Effects of Varying Packet Loss Probability.

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 5(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 application, yielding lower goodput.

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 6(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 6(b). Under five-packet, low RTO increases the percentage of aborted connections, and consequently data, as seen in Figure 6(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 connection-state timers expire. Five-packet is only resilient to duplication (Figure 6(b)), which is expected given its provably correct no-loss/no-duplication behavior. Two-packet and three-packet, like Delta-t, do not suffer from aborted connections (Figure 6(c)) since they do not rely on explicit connection-opening messages.

Under all protocols, lower RTO causes premature retransmissions, which increase both the total number of messages sent (message rate in Figure 6(d)) and the message overhead (Figure 6(e)).

Figure 6(f) shows that the goodput of two-packet and three-packet does not change much for varying RTO. The goodput of five-packet and Delta-t is lower than that of other protocols—the price of providing correct no-loss/no-duplication behavior. Five-packet uses explicit connection-management (handshaking) messages, whereas Delta-t forgoes explicit handshaking by maintaining connection-states for longer periods of time. Under lower RTO, the goodput of Delta-t is higher than that of five-packet. This is because five-packet aborts more connections (and hence data messages) when the retransmission limit of connection-opening messages (SYN and SYN+ACK) is exceeded due to increased number of premature retransmissions.

V. 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 [11] built on the two-packet protocol and designed Delta-t, a pure timer-based protocol for reliable

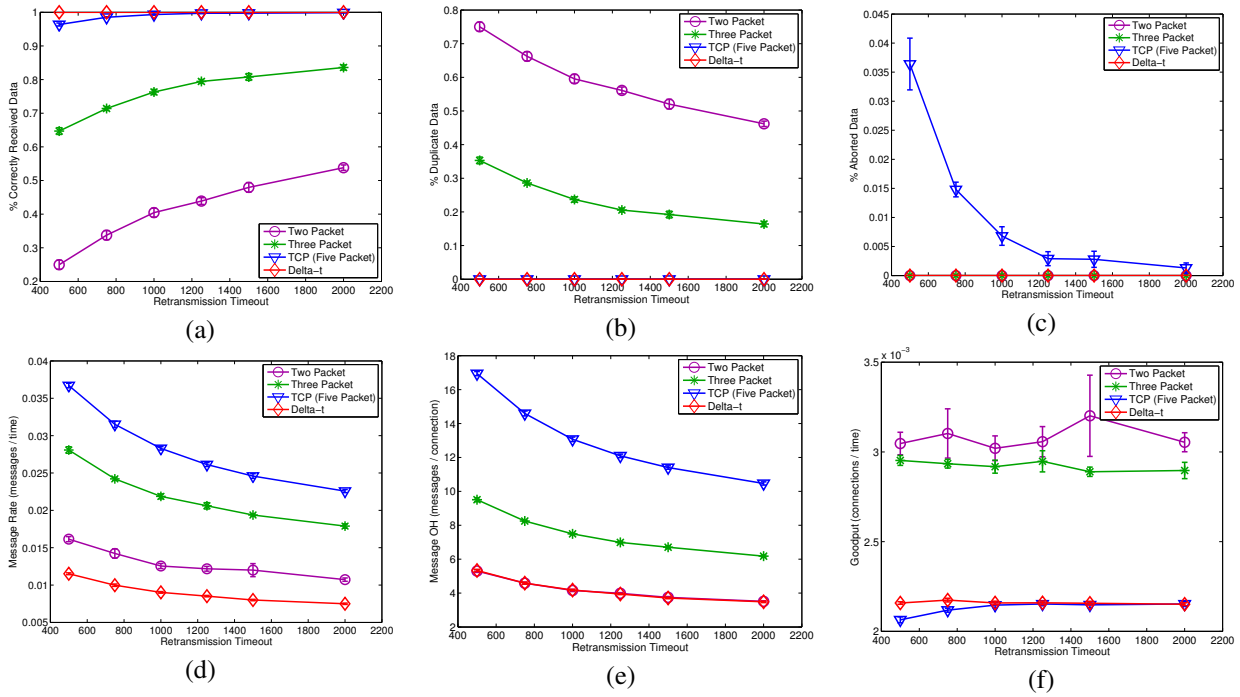


Fig. 6: Effects of Varying Retransmission Timeout.

connection management. TCP [8] 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.*, [9], [7]) 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. Recently, there has been great interest in understanding similar protocol design tradeoffs in a quantitative manner. Ji *et al.* [5] and Lui *et al.* [6] studied such tradeoffs for general reservation/signaling protocols. Our work specializes the general signaling model of [5] to connection management for reliable transport, and so in this paper, we are concerned with unique issues related to data loss / abort / duplication due to inconsistent connection-states at the sender and receiver or failure to establish a connection.

VI. 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 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. Though SS may have not looked attractive in the past due to its additional memory requirement for keeping connection-states, memory is no longer a concern. 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, 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.

References

[1] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP Round-Trip Times. In *Proceedings of the 3rd 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] J. G. Fletcher and R. W. Watson. Mechanisms for a Reliable Timer-Based Protocol. *Computer Networks*, 2:271–290, 1978.

[4] 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.

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

[6] 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.

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

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

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

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

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