Trade-offs between Message Delivery and Quiesce Times in Connection Management Protocols

Jon Kleinberg* Hagit Attiya† Nancy Lynch‡

Abstract

The problem of implementing reliable message delivery using timing information is considered. Two important parameters, from the point of view of system performance, are the time required to deliver a message and the time that elapses between periods of *quiescence*, in which a processor returns to an initial state and deletes all earlier connection records. It has been frequently observed that there is no known protocol which simultaneously optimizes both these quantities; in this paper we prove such trade-offs precisely in the form of lower bounds. Despite the simple nature of the problem, the relationships among these lower bounds are quite subtle, in that they depend critically on the level of synchronization in the processors' clocks. We consider three basic timing models: asynchronous processors, processors that have (approximately) synchronized clocks, and processors with clocks that read different values but run at (approximately) the same rate. We mainly focus on networks that can duplicate and re-order packets; at the end, we also consider message loss and processor crashes.

1 Introduction

Reliable message delivery lies at the heart of fault-tolerant communication between parties on a distributed network. We are interested here in problems of connection management, in which a *sender* S wishes to open a connection to a remote *receiver* R, transmit information, and later release the connection. Connection management constitutes the *transport* layer of the OSI hierarchy (see e.g. [15]).

Protocols based on the transport layer are the basis for ftp, telnet, remote procedure calls, and a number of other common primitives. In a large network, each sender will typically maintain a number of such sessions in parallel; moreover, there can be a number of different *incarnations* of a session with a single receiver, as the connection is opened, closed, and opened again. In a network subject to faults such as packet

*Laboratory for Computer Science, MIT, Cambridge MA 02139 USA, kleinber@theory.lcs.mit.edu. Author is supported by an ONR Graduate Fellowship.

†Department of Computer Science, The Technion, Haifa 32000, Israel, hagit@cs.technion.ac.il. Partially supported by grant No. 92-0233 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, Technion V.P.R.—Argentinian Research Fund and the fund for the promotion of research in the Technion.

‡Laboratory for Computer Science, MIT, Cambridge MA 02139 USA, lynch@theory.lcs.mit.edu. Partially supported by NSF grant 9225124-CCR, AFOSR grant F49620-94-1-0199, and ARPA grant N00014-92-J-4033.

duplication, it is important to maintain connection records keeping track of which packets have already been received, acted on, and so forth. At the same time, with many parallel sessions going on, a processor cannot maintain its entire history for very long. So a processor will periodically *quiesce*, deleting past connection records.

Minimizing both the time required for message delivery and the amount of time information is maintained before quiescence is important for optimizing the performance of such a communication subsystem. Message delivery time determines the latency of packet transmission, especially in short incarnations such as remote procedure calls. Time until quiescence affects how much information must be stored at each node. A large number of protocols have been proposed to optimize these parameters (e.g. [4, 8, 14, 15, 16, 18]). Viewed on a spectrum that ranges from near-optimal message delivery time to near-optimal quiescence time, we have a number of clock-based protocols at the former extreme (e.g. [8]) and the canonical three-packet handshake [4, 16] at the latter.

Clock-based protocols are in general based on the *maximum packet lifetime* (MPL) of the network; this is simply the longest amount of time that any copy of a packet can remain undelivered in the network. R can deliver very quickly if it is prepared to maintain a record for a length of time equal to the MPL; in this way, it can be sure that after quiescence it will never receive a duplicate copy of the current message.

The three-packet handshake, on the other hand, operates with very little overhead in terms of clocks or connection records. Instead, each processor has a source of *unique identifiers* (UID's) — that is, a way to generate an abstract identifier that has never been used before. Now each message is sent using a three-way exchange: first S sends a UID x; R generates a UID y and replies with $\langle x, y\rangle$; and finally S sends the message together with y. In this way, R can be sure that it is not delivering a duplicate; unfortunately, it now takes three times as long to deliver the initial message in the connection.

Thus in practice, protocols tend to be inefficient either in delivery or in quiescence; and it is a common belief that there is some sort of inherent trade-off in these performance measures. In this paper, we prove such trade-offs precisely in a number of natural settings, by demonstrating non-trivial lower bounds on delivery and quiescence times. Most of these trade-offs are essentially the best possible, in the sense that we show protocols whose performance guarantees nearly match the lower bounds.

The relationships among our time bounds turn out to be more subtle than the simple nature of the problem would seem to suggest. The trade-offs one can achieve depend critically on the amount of synchrony in the clocks of the processors, and it is mainly from this perspective that we study the problem.

We express time bounds in terms of two main parameters. The first of these is μ, the maximum packet lifetime, which has been introduced above. The other parameter is d_e, the maximum packet delay *in a specific execution* e; this is simply the supremum of the times that elapse between the sending and the receipt of packets in e. Our motivation in bounding delay in terms of d_e, a quantity that S and R cannot know *a priori*, is the following: we wish to be able to prove bounds that hold *for every* execution of a protocol, not just in a worst-case sense. Thus, for instance, while it is correct to say that the time required before delivery by the three-packet handshake is at most 3μ, one can make the stronger statement that the time required is at most $3d_e$ in execution e. In this way, one can consider whether a given protocol has the following desirable property: in "good executions" (those with $d_e \ll \mu$), the time required is small relative to d_e.

In a network that can duplicate and re-order messages, our main results are the following. (Note that we will sometimes write d_e as d when the execution is clear from context.)

- Section 3: If the processors have no access to clocks and must eventually quiesce, it can take at least time $3d$ to deliver some message.
- Section 4: If S and R have clocks that each run at the rate of real time but whose values differ by some arbitrary translation, then there must be some be some execution in which it takes time at least $3d$ to deliver *or* an execution in which it takes time at least μ to quiesce. This is the basic trade-off result, showing that one must have either a delay before delivery as in the 3-packet handshake, or a delay before quiescence as in the simple timer-based protocol.
- Section 4.2: If we introduce uncertainty, assuming that the rate of increase of the clocks is always within a factor of ρ of real time, then the lower bound increases to $3d$ for delivery or $\rho^2\mu - 3\rho^2 d$ for quiescence; again, this is essentially tight.
- Section 5: We also consider the case in which S and R have ε-synchronized clocks: each always holds a value that is within ε of real time, but no assumption is made about their rates at any given instant. Here we define a novel family of protocols which, for any pre-specified $\alpha \geq 1$, allows for message delivery within time $(1+\frac{2}{\alpha})d+O(\varepsilon)$ and quiescence within time $(\alpha+2)d+O(\alpha\varepsilon)$. The key point here is that neither value depends on the MPL μ. We give a lower bound showing that this trade-off is essentially the tightest possible when $\varepsilon = 0$; finding a nearly matching lower bound for arbitrary $\varepsilon > 0$ is left as an open problem.

At the end, in Section 6, we consider two other types of network failures — message loss and processor crashes. In each case, we mention natural assumptions under which the problem can be reduced, from the protocol designer's point of view, to the case of duplication/re-ordering alone. Also, we consider a simple probabilistic model of message loss in which each packet has an independent probability of being lost. We prove a trade-off here between the expected number of packets sent and the expected time to deliver a message; determining the relationship between these two performance measures more precisely appears to be an interesting question.

The theoretical research on this problem has been much less voluminous than the practical work mentioned above. Harvey and Lynch [7] propose a number of the problems considered here, and obtain some of the initial results on duplication. In the fully asynchronous setting, Fekete, Lynch, Mansour, and Spinelli, and Afek et. al., prove impossibility results for different types of reliable communication [5, 1]. Wang and Zuck [17] consider the sequence transmission problem, in which a sender must transmit a specified sequence of data to a receiver over a faulty channel. Further results in an asynchronous model, based on the minimum amount of information that must be maintained between connections, are proved by Attiya, Dolev, and Welch [2]; their paper deals with the existence or non-existence of connection management algorithms, and not with bounds on quiescence time. Attiya and Rappoport [3] consider the connection management problem primarily in the asynchronous setting, focusing on the amount of information exchange that must take place between parties establishing a connection.

2 Preliminaries

I was feeling kinda lonesome and blue
and needed somebody to talk to.
So I called up the operator time,
just to hear a voice of some kind.
"When you hear the beep it will be three o'clock."
She said that for over an hour and I hung up.
— Bob Dylan,
Talking World War III Blues.

We are considering the problem of at-most-once message delivery between a sender S and a receiver R communicating over an unreliable channel. For our purposes, the sender is interested in transmitting a single message to the receiver; the receiver is required to deliver the message eventually (subject to some liveness requirements discussed below), and never to deliver it a second time.

We represent the system as a collection of four interacting automata (plus the network, which is also modeled as an automaton):

- U_S and U_R are the two users at the opposite ends of the connection; U_S wants to send a message to U_R.
- S and R are the network interfaces for U_S and U_R respectively.

In general, we will not be concerned with the structure of U_S and U_R; U_S simply provides inputs to S, consisting of messages that it wants delivered. Formally, S are R are timed I/O automata of the type considered by Lynch and Vaandrager [12], augmented with liveness properties as in Gawlick et. al. [6]. (See also [10] for background on general I/O automata.) Speaking informally, the fundamental property of S and R is that their states consist of an *internal component* and a *clock component*. The clock is simply a monotone increasing (and unbounded) continuous function of real time; processors can allow specified amounts of time to pass on the clock (modeled as a *time-passage action*) and perform certain actions when the clock reaches a specified value. For technical reasons, the state of the automaton always contains a $\underline{\text{now}}$ component, which gives the value of real time; however, the automaton cannot directly make use of this component of its state.

Each of S and R begins with the *internal component* of its state equal to initial values σ_S^0 and σ_R^0 respectively; no local actions are enabled in these states. Quiescence is modeled as a transition of the internal component of a processor's state to this initial value; the clock component of its state is not affected. Similarly, when a processor crashes, the internal component of the state reverts to the initial value, but the clock component is not affected.

Processors S and R communicate by means of packets sent across the network; we will say that a processor executes the action $send(p)$ to send a packet, and $receive(p)$ when a new packet arrives. We will assume a total *cause function* Γ, which maps each $receive(p)$ action to a $send(p)$ action for the same packet. Note that in a network that can duplicate packets, this function Γ can be many-to-one — that is, $receive(p)$ can be executed several times for a single packet p sent once. However, we will assume that each packet can be duplicated only a finite number of times. Finally, we reserve the terms "transmit" and "deliver" for the higher-level primitives executed by the users U_S and U_R on messages.

Since we will be concerned with the delivery of a single message, we assume that U_S will provide a single input to S (i.e. the message) at the beginning of any given execution, after which U_S and U_R do not provide any inputs to S and R respectively. Thus, in any execution e, the inputs to S, beginning in σ_S^0, will consist of an initial input u^* from U_S, followed by a sequence of packets $r_1, r_2, \ldots$ from R; the inputs to R, beginning in σ_R^0, will simply consist of a sequence of packets $s_1, s_2, \ldots$ from S. For an execution e and a processor P, we use the standard notation $e|P$ to denote the *projection* of e on P — this is the sequence obtained by projecting all states of the system onto those of P, removing actions not belonging to P, and collapsing consecutive subsequences corresponding to time-passage actions of P. If e and e' are two execution fragments, such that e ends with the same state (and value of real time) with which e' begins, we write ee' to denote the "concatenated execution" in which the common state at the juncture between the two executions appears only once.

If the network can only duplicate and re-order packets, then informally one can state fairly simply the correctness conditions for our message-delivery problem. We say that a pair of automata (S, R) constitute a *message-delivery protocol* if in every execution e beginning with the input of a message from U_S to S, there is exactly one action in e in which R delivers the message to U_R, and at least one action following this delivery in which R quiesces. When we consider processors with clocks, we also consider the amount of time that elapses from the input of the message to S to the action in which R delivers and to the first subsequent action in which it quiesces.

Our timing assumptions concern the properties of the clock components that we can guarantee. The three basic timing models are

(i) Asynchronous processors. Neither S nor R has access to a clock.

(ii) Translated clocks. The clocks of S and R run at the rate of real time, but they are offset by some arbitrary amount from each other (and from real time). We will also consider the case of drifting clocks, in which the clocks of S and R run at a rate between $\frac{1}{\rho}$ and ρ times that of real time (the drift rate can also vary with time).

(iii) ε-Synchronized clocks. The clocks of both S and R are always within ε of real time (and hence always within 2ε of each other).

In the latter two models we will assume, except where noted otherwise, that processors know the maximum packet lifetime μ; recall that this is defined as the maximum amount of time that can elapse between events $send(p)$ and $receive(p)$ that are related by the cause function Γ. This is a global bound that holds for all executions. For a *specific* execution e of a protocol, one can simply compute the maximum, over all p that are received, of the time between $send(p)$ and $receive(p)$; this quantity is the longest packet delay to occur in execution e, and we denote it by d_e. (Sometimes we will write it simply as d when the execution is clear from context.) Note that d_e is always at most μ, but it can be substantially less in "well-behaved" executions.

If the network guarantees that, for example, one out of every k packets will not be lost, for some fixed k, then one can incorporate enough re-sends into a protocol to ensure that each packet a processor wants to send will get through — this notion is discussed further in Section 6. Thus, for the bulk of our results (Sections 3, 4, and

5), we consider networks that only duplicate and re-order messages. In Section 6 we consider other types of failures, specifically message loss and processor crashes. The results are somewhat different with these failure models; however, as suggested above, we can state in both cases fairly mild assumptions that result in natural reductions to the case of duplication/re-ordering only.

3 The Asynchronous Model

Here we assume that the processors do not have access to clocks, and that there is no MPL. We show that if the processors are provided with a source of *unique identifiers* (UID's) — making the three-packet handshake possible — it can take time at least $3d$ to deliver the message. Thus the three-packet handshake is in fact optimal in this setting.

We are dealing with asynchronous processors, so the clock component of the state can take only a single value (and hence is trivial). Without giving the processors any additional power (i.e. UID's), we can show that at-most-once delivery is impossible. A similar result is proved in [2].

Proposition 1 *Consider the asynchronous model without UID's. If R must eventually quiesce, then at-most-once delivery is not possible.*

Proof. Consider, by way of contradiction, a protocol for at-most-once delivery in this model. First consider an execution e in which R delivers a message and quiesces. For notational purposes, suppose that e begins at real time 0 and ends with the quiescence of R at real time T; of course, the processors do not have access to these values of real time.

We now construct an execution e' in which R delivers the message twice. A prefix of e' is equal to e; at the end of this prefix, R is back in its initial state σ_R^0. The remainder of e' will consist of an execution fragment e'_1 which we construct so that $e'_1|R = e|R$, as follows.

Suppose that in e, R receives packets $s_1, \ldots, s_k$ from S, which arrive at times $0 \le t_1 \le \cdots \le t_k \le T$. Since we assume that the network can duplicate packets, we can construct the execution fragment e'_1 so that it begins at time T, and the inputs to R consist of the packets $s_1, \ldots, s_k$, with packet s_i arriving at time $T + t_i$. Since S has not necessarily quiesced, it may be continuing to send packets to R; however, we will have all such packets arrive after time $2T$. So in the interval $[T, 2T]$, R begins in its initial state, in which no local actions are enabled, and the only inputs it receives are the replayed packets. Thus we can construct e'_1 in the interval $[T, 2T]$ such that $e'_1|R = e|R$; specifically, R delivers the message in e'_1. Thus, in the execution $e' = ee'_1$, R delivers the message twice. ■

The key point in the above impossibility proof is that R has no way to distinguish its state following quiescence from its state at the beginning of the execution; we can therefore construct an execution in which it delivers the message a second time when presented with the appropriate sequence of duplicate packets. Introducing UID's changes the picture considerably.

We model UID's as follows. The state of a processor P is augmented with an additional component: an infinite set I_P of abstract identifiers. Since only the internal component of P's state is reset when it quiesces or crashes, this collection of UID's survives such events. The UID's can be copied and included in messages; however, the only operations a processor can perform on them are the following.

(i) *generate*(), which nondeterministically returns a new UID u and deletes it from the set I_P (thus the UID component of P's state undergoes a transition from I_P to $I_P - \{u\}$; this ensures that u will never be used again).

(ii) If x and y are UID's, then $same(x, y)$ returns *true* iff $x = y$ and *false* otherwise.

In particular, P cannot directly "read" the current UID component of its state. In our case, we additionally assume that the sets I_S and I_R are disjoint (so R for example will not confuse a UID from S with one of its own).

In subsequent sections we will be dealing with processors that have clocks, with values that can be added, subtracted, and so on; here, however, we consider UID's as they are used in practice in the absence of clocks: as abstract identifiers that can only be included in messages and compared with one another to test equality. Note that if a processor receives a packet π, it can determine whether it has previously received π only *in the interval since its last quiescence.*

If there are UID's, then the three-packet handshake allows for the delivery of a message M within time $3d$. We are concerned with the delivery of a single message; hence for our purposes, the three-packet handshake can be implemented as follows.

(i) On input $\langle M \rangle$ from U_S, S generates a UID x and sends the packet $\langle x \rangle$.

(ii) If R receives $\langle x \rangle$ and has not previously received $\langle x \rangle$ since its last quiescence, then R generates a UID y and replies with the packet $\langle x, y \rangle$.

(iii) If S receives $\langle x, y \rangle$, and since its last quiescence it has sent a packet $\langle x \rangle$ and not received $\langle x, z \rangle$ for any z, then S replies with the packet $\langle y, M \rangle$. (If S receives $\langle x, y \rangle$ and this condition is not met, it sends an error message to R to allow R to quiesce.)

(iv) If R receives $\langle y, M \rangle$, and since its last quiescence it has sent a packet $\langle x, y \rangle$ and not received $\langle y, M' \rangle$ for any M', then R delivers the message M.

In the case of single-message delivery, and assuming the network can only duplicate and re-order, S and R can quiesce at the end of the third and fourth lines respectively. For more complicated types of failures, some additional acknowledgement is needed before quiescence. Analyzing this protocol in detail is not something we will undertake here (see [9] and [13] for detailed analyses); for now we simply note that it is a very robust way to deliver messages within time $3d$ and quiesce immediately.

The following lower bound shows that in an asynchronous system, there is no way to improve on the worst-case performance of this protocol.

Theorem 1 *Assume an asynchronous system with UID's, and suppose that R must eventually quiesce. Then for every message-delivery protocol and every positive d there is an execution e with $d_e = d$ in which at least time $3d_e$ elapses before the delivery of the message.*

Proof. Assume that the claim does not hold for some protocol and some choice of d; we derive a contradiction. In execution e, S sends its first packet at real time 0, and all packets take time $d_e = d$ to arrive. Thus R delivers at time $t < 3d$ and then quiesces

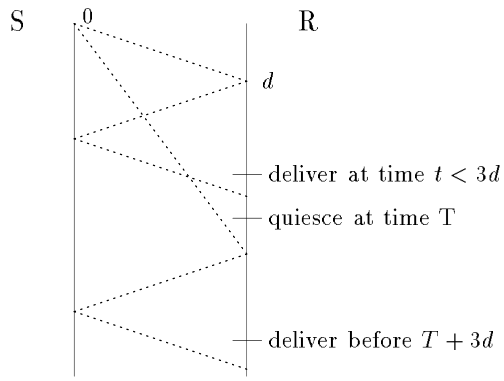

Figure 1: The skeleton of a lower bound proof

at time $T \geq t$. Let u_S, u_R denote the (finite) sets of UID's generated by S and R respectively before time T. A key property of execution e is the following. Since R begins in its initial state σ_R^0 in which no local actions are enabled, it does not send any packets before real time d; thus S receives no packets from R prior to real time $2d$. Therefore, since the only packets received by R before delivering the message are sent by S in the interval from real time 0 to $t - d < 2d$, the only UID's included in packets received by R before delivering belong to the set u_S.

Now we construct an execution f to be identical to e, except that the UID component of R's state begins with the value $I_R - u_R$ (note that this is still an infinite set, as required). That is, S sends its first packet at real time 0, all packets take time $d_f = d$ to arrive, and S generates the same set of UID's in the interval $[0, t-d]$. Since S and R can only perform $same(x, y)$ tests on the UID's, and the UID's in packets sent by S before real time $2d$ all belong to u_S, we can construct f so that $e|S = f|S$ in the interval $[0, t-d]$, R delivers the message at real time t, and quiesces at real time T.

Finally, we construct an execution e' in which R delivers the message twice; the outline of this construction is depicted in Figure 1. A prefix of e' is equal to e; at the end of this prefix, R has returned to its initial state σ_R^0, and the value of real time is T. Recall that no local actions are enabled in the initial state of R, and its current set of UID's is $I_R - u_R$. We now complete e' with an execution fragment e'_1 as follows. As in the previous proof, we replay all packets sent by S in the interval $[0, t-d]$, and have any new packets sent by S arrive after real time $T + t$. So the only inputs R is receiving in the interval $[T, T+t]$ are the replays from S; since $e|S = f|S$ in the interval $[0, t-d]$, we can construct e'_1 so that $e'_1|R$ in the interval $[T, T+t]$ is equal to $f|R$ in the interval $[0, t]$. Thus R delivers the message in the fragment e'_1, and hence delivers it twice in the execution $e' = ee'_1$. ■

4 Translated Clocks

When we assume clocks that run at the rate of real time but are translated by some arbitrary amount, the main result is a trade-off showing that *either* it takes time at least $3d$ to deliver the message, *or* it takes time at least μ to quiesce. If we weaken the timing guarantee so that clocks are only running at a rate within ρ of real time, then the lower bound on the time to quiesce in this trade-off is multiplied by a factor of ρ^2. There are simple protocols nearly matching these lower bounds in all cases.

4.1 Clocks at the Rate of Real-Time

Here, we assume that the two processors have clocks that run at the rate of real time, but their values are shifted by an unknown amount. The clocks of S and R can be represented by functions

$$\gamma_S,\ \gamma_R : \Re \rightarrow \Re.$$

Our assumption can then be expressed by saying that for all values t_1 and t_2 of real time, we have

$$\gamma_R(t_1) - \gamma_R(t_2) = \gamma_S(t_1) - \gamma_S(t_2) = t_1 - t_2.$$

To specify local times at S and R, we will sometimes use the terms S-time and R-time respectively; that is, S-time t (resp. R-time t) is equal to real time $\gamma_S^{-1}(t)$ (resp. $\gamma_R^{-1}(t)$).

Recall that we measure the time to quiesce from the first *send* event. If the value μ of the MPL is known, there is a natural algorithm that allows for immediate delivery (i.e. delay d), but requires time $\mu + d$ to quiesce. This is simply the following rule: as soon as R gets the first packet, it delivers the message; it then counts off μ on its clock before quiescing. Of course, the 3-packet handshake still allows for R to deliver and quiesce within time $3d$. The following lower bound shows that one cannot simultaneously improve on both these quantities.

Theorem 2 *Consider a system with translated clocks and a known value of μ; let d be a constant satisfying $0 < d < \frac{1}{3}\mu$. For any message-delivery protocol there is an execution e with $d_e = d$ for which at least time $3d_e$ elapses before the delivery of the message, or at least time μ elapses before the quiescence of R.*

Proof. Assume that the claim does not hold for some protocol and some choice of d; we derive a contradiction. First we construct an execution e in which both clocks start at 0, and all packets take time $d_e = d$. By our assumption about the protocol, we can have R deliver the message at some time $t < 3d$, and quiesce at time $T < \mu$. The execution e ends with the quiescence of R.

We now construct execution f. In f, the clock of S starts at 0, the clock of R starts at $\mu - d$, and $d_f = d$. The first packet is sent by S at S-time 0 (which is the same as R-time $\mu - d$); thus by our assumption about the delivery bound, R delivers by R-time $\mu - d + t' < \mu + 2d$. Since no local actions are enabled in the initial state of R, S does not receive a packet from R until S-time $2d > t' - d$, and so we can construct f so that $e|S = f|S$ in the interval $[0, t' - d]$ of S-time.

Finally, we construct an execution e' in which R delivers the message twice. A prefix of e' is equal to e. The remainder of e' is an execution fragment e'_1 constructed as follows. We replay the packets sent by S in the interval $[0, t' - d]$ of S-time, and have any other packets sent by S take time μ to arrive at R. At R-time T, R is in its initial state because it has just quiesced; between R-time T and μ, R receives no packets from S and hence remains in its initial state. Since no local actions are enabled in this initial state, and the inputs received by R in the interval $[\mu, \mu + t' - d]$ of R-time are the same as they are in f, we can construct e'_1 in the interval $[\mu, \mu + t' - d]$ of R-time so that

$e_1'|R = f|R$ in this interval. Thus, R delivers the message in e_1', and hence delivers the message twice in execution $e' = ee_1'$. ■

Suppose that there exists a value for μ that holds in all executions, but the processors do not know this value. Then using the technique of the previous proof, we could construct execution e' by replaying the packets of S *regardless* of how long R waits before quiescing (if R quiesces after T units of real time, we simply choose $\mu > T$). Thus we obtain an execution in which R delivers twice, simply assuming that the time until delivery is strictly less than $3d_e$; in this way, one can prove the following result.

Theorem 3 *Consider a system with translated clocks in which the value of μ is not known, and suppose R must eventually quiesce. Then for every message-delivery protocol and every positive d, there is an execution e with $d_e = d$ for which at least time $3d_e$ elapses before the delivery of the message.*

For the remainder of the paper we will assume that the value of μ is known.

4.2 Drift in Translated Clocks

In this section, we assume translated clocks, and weaken the guarantee that the two clocks run at the same rate. Thus, for a clock γ, our guarantee is that

$$\forall x, y \in \Re,\ x < y\ :\ \frac{1}{\rho} \leq \frac{\gamma(y) - \gamma(x)}{y - x} \leq \rho$$

A clock with this property will be called ρ-drifting.

In this setting, the 3-packet handshake still provides for delivery and quiescence within time $3d$. At the other extreme, there is a natural algorithm that ensures delivery in time d and quiescence within $\rho^2\mu + d$ units of real time: R delivers immediately and then counts off $\rho\mu$ on its clock.

Again, we can prove that one cannot improve substantially on both bounds at the same time.

Theorem 4 *Consider a system with ρ-drifting clocks; let d be a constant satisfying $0 < d < \frac{1}{3}\mu$. For any message-delivery protocol there is an execution e with $d_e = d$ for which at least time $3d_e$ elapses before the delivery of the message, or at least time $\rho^2\mu - 3\rho^2 d_e$ elapses before the quiescence of R.*

Proof. Assume that the claim does not hold for some protocol and some choice of d; we derive a contradiction. Set $\tau = \frac{d}{\rho}$.

First we construct an execution e in which both clocks start at 0 and run at $\frac{1}{\rho}$ times the rate of real time (that is, $\gamma_S(t) = \gamma_R(t) = \frac{t}{\rho}$). Packets take $d_e = \rho\tau = d$ units of real time to arrive. Thus, we can construct e so that R delivers in less than $3\rho\tau$ units of real time; hence it delivers before local time 3τ. Similarly, since it quiesces in less than $\rho^2\mu - 3\rho^3\tau$ units of real time, it quiesces before local time

$$t_0 = \rho\mu - 3\rho^2\tau.$$

Next we construct executions f and e' as follows. In f, the clock of S begins at 0 and the clock of R begins at $t_0 = \rho\mu - 3\rho^2\tau$. The clocks again run at $\frac{1}{\rho}$ times the rate

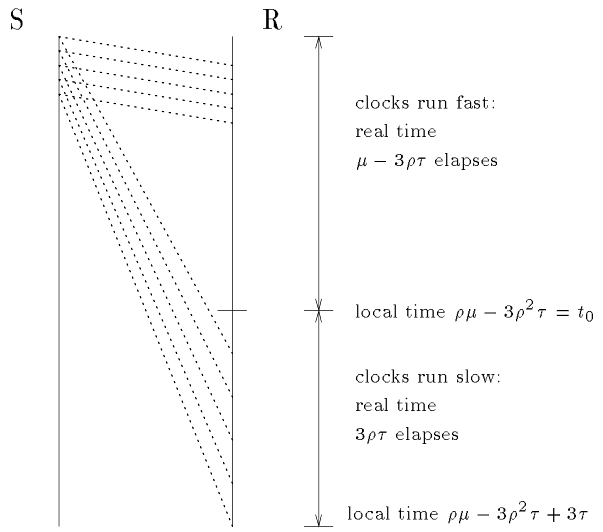

Figure 2: Final execution in Theorem 4

of real time and packets take $\rho\tau$ units of real time to arrive. We can construct f so that R delivers at local time

$$t_1 < \rho\mu - 3\rho^2\tau + 3\tau.$$

Let $t' = t_1 - t_0$; so $t' < 3\tau$. For any $x \in [0, t' - \tau]$, a packet sent by S at S-time x should be received by R at R-time $t_0 + \tau + x$. Also, since no local actions are enabled in the initial state of R, S does not receive a packet from R before S-time $2\tau > t' - \tau$; thus we can construct f so that $f|S = e|S$ in the interval $[0, t' - \tau]$ of S-time.

In execution e', both clocks start at 0 and run at ρ times the rate of real time; packets take $d_{e'} = \frac{\tau}{\rho}$ units of real time to arrive. Since the amount of local time that elapses between the sending and the receipt of every packet is the same in e and e', we can construct e' so that $e'|S = e|S$ and $e'|R = e|R$. So R quiesces before local time $\rho\mu - 3\rho^2\tau$ in e'; thus it quiesces in less than $\mu - 3\rho\tau$ units of real time. Also, we have $f|S = e'|S$ in the interval $[0, t' - \tau]$ of S-time. The execution e' ends with the quiescence of R.

Finally, we construct an execution f' in which R delivers the message twice. A prefix of f' is equal to e'. The remainder of f' is an execution fragment f'_1 which begins at real time $\mu - 3\rho\tau$ (and hence local time $\rho\mu - 3\rho^2\tau$ on both clocks). In f'_1, both clocks run at $\frac{1}{\rho}$ times the rate of real time, and replays begin arriving at R as they did in execution f. (By having any other packets sent by S take μ units of real time to arrive at R, we can ensure that such packets will not interfere with this part of the construction.) So as in execution f, a packet sent by S at S-time $x \in [0, t' - \tau]$ is received by R at R-time $t_0 + \tau + x$. For any such packet sent at S-time $x \in [0, t' - \tau]$, note that at most $\mu - 3\rho\tau$ units of real time elapse before R quiesces, and at most $\rho(x + \tau) \leq 3\rho\tau$ elapse after. Thus, no replay arrives more than μ units of real time after it was sent by S.

By our construction, the only inputs that R receives following quiescence and up to R-time $t_1 = t_0 + t'$ are replays of packets sent by S in the interval $[0, t' - \tau]$ of S-time. But we argued above that $e'|S = f|S$ in this interval; thus, we can construct the fragment f_1' so that $f|R = f_1'|R$ in the interval $[t_0, t_1]$ of R-time. Thus R delivers the message in f_1', and hence delivers it twice in the execution $f = ff_1'$. ∎

5 Approximately Synchronized Clocks

Having approximately synchronized clocks gives the processors a considerable amount of additional power; specifically, the trade-off lower bound of Section 4 no longer applies. In fact, we can show that for arbitrarily small $\delta > 0$, R can deliver within time $(1+\delta)d$ and still quiesce in an amount of time that depends only on d, and not on the MPL μ. This is in striking contrast with the bounds of the previous section.

Recall that an approximately synchronized clock γ is one that is always within ε of real time, and always monotone increasing. That is, for all t we have $|\gamma(t) - t| \leq \varepsilon$. Note that this condition implies $|\gamma_S(t) - \gamma_R(t)|$ is always at most 2ε. In this section, we will assume that $2\varepsilon < \mu$; otherwise, we essentially have the case of translated clocks as in Section 4.

In the previous section, we saw an algorithm that allowed for immediate delivery (in time d), at the cost of requiring μ for quiescence. Here we are after something quite different: we want R to wait only a little bit more than d before delivering, and still quiesce in an amount of time that depends only on d. We sketch an algorithm that achieves this now, and then define it precisely in the proof of Theorem 5.

Assume that S sends out the initial packet at local time 0 (hence at real time $\gamma_S^{-1}(0)$). Subsequently, both S and R send out time-stamped packets at regular intervals (say at intervals of c'). Since γ_S and γ_R are approximately synchronized, R can use the time-stamps on the packets it receives to maintain an estimate of the value of d_e. To be a little more concrete, define the *lag* of a packet to be the local time at which it was received, minus its time-stamp (so because γ_S and γ_R are only approximately synchronized, a packet with very small transit time could have negative lag). It is not difficult to prove that the maximum of the lags that R observes cannot be much less than d_e.

Suppose that we want R to deliver within time at most $(1+\delta)d_e$, for some small $\delta > 0$. Then R waits until its local time is roughly $(1+\delta)$ times the maximum observed lag, and then delivers the message. An important point is the following: if such a time never comes, R is not required to deliver. R now waits a much longer amount of time, roughly $(\frac{2}{\delta} + 2)d_e$, and then quiesces. What has it accomplished by waiting this long? We argue that it has provided itself with some protection against replays.

For consider an execution f in which a replay of the packet sent by S at local time 0 arrives after R wakes up from quiescence. Then the lag observed by R, and hence its estimate of d_f, will immediately become extremely large. Moreover, R now expects to receive a packet from S at regular intervals of c'; thus there are two possibilities:

(i) Following quiescence, but before R meets the criterion for delivery (i.e. its local time has not yet reached $(1+\delta)$ times d_f), some S-packet is not replayed. Then the maximum lag computed by R will increase linearly with its local time (due to the unreceived S-packet), and it will therefore never be required to deliver.

(ii) All S-packets are replayed until R meets the criterion for delivery. Then since the lag is at least $\frac{2}{\delta}d_e$, the "short" amount of time R waited before delivery was at least $\delta \cdot \frac{2}{\delta}d_e = 2d_e$, and so it must have received the replay of an S-packet in which S announces the receipt of an earlier R-packet. But this could only be possible if the S-packet is a replay (as its time-stamp is much earlier than the time at which R first started generating replies); thus R will be able to abort the connection without delivering the message a second time.

We now turn to a precise statement of the result.

Theorem 5 *For each $\alpha \geq 1$ and $c > 0$, there is a protocol in which R delivers within time $(1 + \frac{2}{\alpha})d + (4 + \frac{4}{\alpha})\varepsilon + c$ and quiesces within time $(\alpha + 2)d + (2\alpha + 6)\varepsilon + c$.*

Proof. Fix $c' \leq \frac{c}{2\alpha+5}$. Each of S and R sends a time-stamped packet to the other at increments of time c'. We can view the protocol as operating in discrete "ticks" of length c'; a *discrete R-time t* is a local time at R which is a positive integral multiple of c', with discrete S-time defined analogously. Note that the gap between two consecutive discrete R-times can be arbitrarily small (though not indefinitely), and can be as large as $c' + 2\varepsilon$. The *threshold* of R at discrete R-time t is defined to be the largest t' for which R has received all S-packets with time-stamp at most t' (i.e. it has not yet received the S-packet with time-stamp $t' + c'$); the threshold of S is defined analogously.

The first packet sent by S contains the message, as well as the current local time. Subsequent S-packets consist of the current local time and the current threshold of S. Initially, S has received no packets from R and hence reports a *trivial* threshold; after it receives its first R-packet, it reports a *non-trivial* threshold. R-packets consist simply of the current local time (to enable S to compute its threshold); the first R-packet is sent when R first receives the initial S-packet.

Assume that S sends its initial packet at discrete S-time 0. Let r_0 denote the discrete R-time at which R first receives the initial S-packet, and hence at which it sends out its first packet to S. R maintains an estimate of the current value of d_e by computing the maximum *lag* $\ell^{(t)}$ of any packet observed up to time t; this is equal to $c' + M^{(t)}$, where $M^{(t)}$ is the maximum over the following three finite sets:

(i) The set of all $r - s$, where the threshold of R at discrete R-time r is equal to s.

(ii) The set of all $s' - r'$, where the threshold value in the S-packet time-stamped s' is equal to r'.

(iii) The set of all $s' - r_0$, where the S-packet time-stamped s' reports a trivial threshold.

By definition, we say that that threshold of R at discrete R-time $r_0 - c'$ (i.e. just before it received the initial packet) is 0. So by the first rule for estimating ℓ, we have $M^{(r_0)} \geq r_0 - c'$ and hence $\ell^{(r_0)} \geq r_0$; by the third rule, we have $\ell^{(r_0)} \geq -r_0$. Thus $\ell^{(r_0)} \geq |r_0|$.

R delivers at the first discrete R-time t' when

$$t' > (1 + \frac{2}{\alpha})\ell^{(t')}$$

and quiesces at the first discrete R-time t'' when

$$t'' > (\alpha + 2)\ell^{(t'')}.$$

It then sends a *done* message to S; S quiesces immediately upon receiving this *done* message. If at any time S reports a threshold that is less than r_0, (i.e. one can conclude that R is hearing replays), R aborts the connection without delivering and sends an *error* message to S.

First we argue that for any t, the actual maximum message delay d_e is at least $\ell^{(t)} - 2\varepsilon - 2c'$. Consider the discrete R-time r at which the maximum value for $\ell^{(t)}$ was attained — i.e. the first $r \leq t$ for which $\ell^{(r)} = \ell^{(t)}$ — and suppose that it was updated using the first rule (the other cases are strictly analogous). Then the threshold of R at r must be equal to $r - \ell^{(t)} + c'$, so the S-packet from discrete S-time $r - \ell^{(t)} + 2c'$ has not yet arrived. Thus

$$\begin{aligned} d_e &> \gamma_R^{-1}(r) - \gamma_S^{-1}(r - \ell^{(t)} + 2c') \\ &\geq (r - \varepsilon) - (r - \ell^{(t)} + 2c' + \varepsilon) \\ &= \ell^{(t)} - 2c' - 2\varepsilon. \end{aligned}$$

So at the discrete R-time just before quiescence we have

$$\begin{aligned} t'' - c' &\leq (\alpha + 2)\ell^{(t'' - c')} \\ &\leq (\alpha + 2)(d_e + 2\varepsilon + 2c') \\ t'' &\leq (\alpha + 2)(d_e + 2\varepsilon) + c. \end{aligned}$$

Since the initial *send* event was at real time $\gamma_S^{-1}(0) \geq -\varepsilon$ and $\gamma_R^{-1}(t'') \leq t'' + \varepsilon$, the time required for R to quiesce is at most

$$(\alpha + 2)(d_e + 2\varepsilon) + 2\varepsilon + c.$$

Note also that the time required for S to quiesce is at most an additional d_e. A similar analysis gives the bound for the time required to deliver.

Now let us show why R will not deliver the message a second time. First we argue that R will not quiesce until it has received an S-packet with a non-trivial threshold. Let ψ denote the S-packet with minimal time-stamp that reports a non-trivial threshold, and consider a discrete R-time r at which R has not yet received ψ. Let $r - v_1$ be the timestamp of the most recent S-packet, and set $v = r - v_1 - r_0$. Then by the first rule for estimating the lag, $\ell^{(r)}$ is at least r_0 and at least v_1; by the third rule, $\ell^{(r)}$ is at least v. Thus,

$$r = r_0 + v_1 + v \leq 3\ell^{(r)} \leq (\alpha + 2)\ell^{(r)},$$

so R will not yet quiesce.

Now let ℓ^* (resp. M^*) denote the maximum value of $\ell^{(t)}$ (resp. $M^{(t)}$) over all discrete R-times t up to quiescence, and s_1 denote the time-stamp of the S-packet ψ. We claim that $s_1 \leq 2\ell^*$. Indeed, the S-packet time-stamped $s_1 - c'$ reports a trivial threshold, so by the third rule for estimating the lag, $M^* \geq s_1 - c' - r_0$, whence $\ell^* \geq s_1 - r_0$. We have already argued that $\ell^* \geq r_0$; adding, we obtain $s_1 \leq 2\ell^*$.

Finally, suppose $T > t''$ and a replay of the original message arrives at time T. We will show that if $T' \geq T$ is some time at which R has not received the replay of the S-packet ψ, it is not required to deliver. Since ψ has not been received at T', we have

$$\begin{aligned} \ell^{(T')} &\geq T' - s_1 \\ &\geq T' - 2\ell^*; \end{aligned}$$

by the fact that R quiesced by R-time T we have $(\alpha+2)\ell^* \le T$, and hence

$$\ell^* \le \frac{T}{\alpha+2}.$$

Thus

$$\begin{aligned}
\ell^{(T')} &\ge T' - 2\ell^* \\
&\ge T' - \frac{2T}{\alpha+2} \\
&\ge T' - \frac{2T'}{\alpha+2} \\
T' &\le (1+\frac{2}{\alpha})\ell^{(T')}
\end{aligned}$$

Thus R is not required to deliver until it receives a replay of ψ. But ψ reports a threshold smaller than T, which is the discrete R-time at which R first started sending packets to S following quiescence. By our rule from above, R will abort the connection in this case. Thus R never delivers the message a second time. ∎

In the case in which $\varepsilon = 0$ (so the clocks of S and R are perfectly synchronized), we can show that the trade-off implicit in the previous result is tight up to additive terms.

Theorem 6 *Consider a system with ε-synchronized clocks; let $0 < \delta < 2$ and $h(x) = \min(\mu, 2x/\delta)$. For any message-delivery protocol there is an execution e for which at least time $(1+\delta)d_e$ elapses before R delivers, or at least time $h(d_e)$ elapses before R quiesces.*

Proof. Assume that the claim does not hold for some protocol and some choice of δ; we derive a contradiction. Note that the local times of the two processors are always equal to the value of real time. Fix d small enough so that $(1+\delta)h(d) < \mu$. We first construct execution e in which both clocks start at 0 and packets take time $d_e = d$; by our assumption about the protocol, we can define e so that R delivers the message at time $t < (1+\delta)d$ and quiesces at time $T < \frac{2}{\delta}d$. Execution e ends with the quiescence of R at time T.

We now construct execution f in which both clocks start at 0 and messages take time $d_f = h(d)$. Again by our assumption about the protocol, we can construct f so that R delivers before time $(1+\delta)h(d)$. Note that in e, S receives its first packet from R at time $2d$. Thus in both e and f the only inputs it receives in the interval $[0, 2d)$ is the initial input from U_S; hence we can construct f so that $e|S = f|S$ in the interval $[0, 2d)$.

Finally, we construct an execution e' in which R delivers the message twice. A prefix of e' will be equal to e; the remainder of e' is an execution fragment e'_1 which begins at time T. We replay all packets sent by S so that they arrive after a delay of $h(d) > T$. No local actions are enabled in the initial state of R, and the only inputs R receives in the interval $[h(d), h(d)+2d)$ are these replayed packets; since $e|S = f|S$ in the interval $[0, 2d)$, we can therefore construct e'_1 so that $e'_1|R = f|R$ in the interval $[h(d), h(d)+2d)$. But $h(d) \le \frac{2}{\delta}d$, so $h(d)+2d \ge (1+\delta)h(d)$ and hence R delivers the message in e'_1. Thus it delivers the message twice in the execution $e' = ee'_1$. ∎

When $\delta = \frac{2}{\alpha}$, the time required to quiesce in Theorem 6 is at least $\min(\alpha d, \mu)$, which is close to matching the bound achieved in Theorem 5. For general $\varepsilon > 0$, we do not know how to obtain a correspondingly tight lower bound, and leave this as an open question.

6 Other Types of Failures

There are a number of possible models for message loss, some of which allow for natural reductions to the case of duplication/re-ordering only. Under a simple probabilistic model of message loss, we show the following type of trade-off: if the expected number of packets sent by a processor is bounded by a constant, then the expected time until quiescence is at least a constant fraction of the MPL μ.

Introducing crashes into the model of Section 5 changes the time bounds one can achieve. In particular, if we assume that R can crash, we obtain a lower bound of $3d$ on the time for delivery; this is in contrast to the fast algorithm presented in Section 5.

6.1 Message Loss

We must assume that the network satisfies some minimal sort of liveness guarantee, or it will not be possible to design any protocol at all. A theoretically appealing liveness formulation is the following: if a processor sends an infinite number of copies of a packet, one will get through. Unfortunately, this still does not allow one to provide any performance guarantees for a protocol with respect to real time.

We propose two kinds of liveness guarantees. The first is to say that for some fixed value of k, if a processor tries to send the same message k times, it will succeed at least once. Under this formulation, there is a general transformation from a protocol tolerating message duplication to one tolerating both message loss and duplication: whenever a processor is supposed to send a packet, it sends it k times in immediate succession.

An equally natural and more slippery kind of guarantee is the following: each packet has an independent probability p, $0 < p < 1$, of being received. Of course, this is a rather simplistic assumption, but it is one that is often made in practice, and it suggests a perspective for approaching these problems in general. Let us imagine a system with synchronized clocks, and consider the following two implementations of the 3-packet handshake.

(i) To send a packet, a processor does the following: it sends the packet, waits 2μ, and tries again if it has gotten no reply. Since three packets must get through in the 3-packet handshake, the expected running time of this implementation is $\frac{6}{p} \cdot \mu$.

(ii) To send a packet, a processor sends it over and over very rapidly until it hears a reply; then it switches to the next packet that it wants to send. The running time of this implementation is $3d$ plus additive terms depending on $\frac{1}{p}$.

Of course, the problem with the second implementation is that it uses an astronomical number of packets. One naturally observes that a whole range of implementations is possible by having a processor wait until time $h(j)$ (for some function h) to try its j^{th} re-send of the packet; this is simply the class of "back-off" algorithms. We believe that an appropriate kind of trade-off to analyze in this model is that of expected time versus expected number of packets sent. In analyzing such algorithms, we must be careful about a number of points. Two of these are

- What is the underlying set over which the expectation is taken in the above examples?
- How do we define d_e in an execution e if some packets are being lost?

We choose to set things up as follows. First of all, the lower bound we prove does not make use of duplicates, so we will assume that the channel only loses packets and does not duplicate them; note that this only strengthens the lower bound. Define a packet history $\mathcal{P}$ to be an infinite sequence of positive real numbers $t_1, t_2, \ldots$. The idea is that we will construct executions e in which the i^{th} packet sent in e takes time t_i.

Let $\mathcal{P}[i]$ denote the i^{th} element of $\mathcal{P}$. If $l = i_1 < i_2 < \cdots$ is a sequence of natural numbers, we define $\mathcal{P}^l$ to be the packet history in which every entry $\mathcal{P}[i_j]$ $(i_j \in l)$ is replaced by the special symbol *loss*; this indicates that the corresponding packet was lost. We now define Φ to be the set of packet histories $\mathcal{P}^l$ for all possible subsequences l. We can define a probability on Φ as follows. First, let π denote a finite prefix of $\mathcal{P}$, in which some of the elements have been replaced by *loss*. Consider the subset Φ^π of Φ consisting of all histories in Φ that begin with the finite prefix π. If π has length k and contains $\ell \leq k$ occurrences of the symbol *loss*, assign Φ^π a probability of $p^{k-\ell}(1-p)^\ell$. This probability now extends uniquely to the σ-algebra Σ generated by all sets of the form Φ^π; i.e. the smallest collection of sets containing all Φ^π which is closed under complement and countable union. Constructions of this sort are a central topic of [11].

An execution e is *consistent* with a packet history $\mathcal{P}^l$ if the j^{th} packet sent in e takes time t_j, or is lost if $j \in l$. An execution tree E is a function which maps each $\mathcal{P}^l \in \Phi$ to an execution $E(\mathcal{P}^l)$. E must satisfy the following two properties.

(i) $E(\mathcal{P}^l)$ is consistent with $\mathcal{P}^l$.

(ii) If $\mathcal{P}^l$ and $\mathcal{P}^{l'}$ are the same on a finite prefix π of length j, then $E(\mathcal{P}^l)$ and $E(\mathcal{P}^{l'})$ are the same through the value of real time corresponding to the sending of the $(j+1)^{\text{st}}$ packet.

For some index set L, let $\Phi' = \{\mathcal{P}^l : l \in L\}$ denote the set of all packet histories for which R delivers the message in $E(\mathcal{P}^l)$. Every $\mathcal{P}_l \in \Phi'$, contains some finite prefix π_l of length j_l such that by the time the ${j_l}^{\text{th}}$ packet is sent in $E(\mathcal{P}^l)$, R has delivered the message. By property (ii) of the fuction E, this means that R delivers the message in every execution induced by a packet history in Φ^{π_l}. Thus,

$$\Phi' = \bigcup_{l \in L} \Phi^{\pi_l}.$$

But since there are only countably many finite prefixes π_l, this is a countable union of members of Σ, which is therefore in Σ. Thus, the set of packet histories inducing executions in which R delivers the message is measurable, and we can compute its probability.

Similarly, we can compute the expected time to deliver the message and the expected number of packets sent before the delivery of the message (these could be infinite if, for example, the measure of Φ' is not 1). For an execution tree E associated with a packet history $\mathcal{P}$, let d_E denote the least upper bound of the set of packet delays t_i in $\mathcal{P}$.

As an example of the sort of lower bound one might try to prove in this model, we show the following result; speaking informally, it says that every algorithm either uses an expected number of packets that is super-constant, or requires an expected amount of time to deliver that is a constant fraction of μ.

Theorem 7 *Let p denote the probability of each packet being received, and let c be a positive constant. Then there is a constant $\delta > 0$ such that the following holds for every message-delivery protocol: there is an execution tree in which the expected number of packets sent is at least c, or for every positive $d \leq \mu$, there is an execution tree E with $d_E = d$ for which the expected time to deliver is at least $\delta\mu$.*

Proof. We claim that the theorem holds with $\delta = (1-p)^{-c}$. We consider a protocol for which the expected number of packets sent is at most c in all execution trees; for each $d \leq \mu$ we construct such an execution tree E with $d_E = d$ for which the expected time to deliver is at least $\delta\mu$.

Suppose that both clocks start at 0. Let $\mathcal{P}_\mu$ denote the packet history in which all packets take time μ; we construct an execution tree E_μ consistent with $\mathcal{P}_\mu$. For any subsequence l, the only input received by S in the interval $[0, \mu)$ in $E_\mu(\mathcal{P}^l_\mu)$ is the initial input from U_S; R receives no inputs in the interval $[0, \mu)$ in $E_\mu(\mathcal{P}^l_\mu)$. Thus we can construct E_μ so that for all l, l', $E_\mu(\mathcal{P}^l_\mu)|S = E_\mu(\mathcal{P}^{l'}_\mu)|S = e_S$ and $E_\mu(\mathcal{P}^l_\mu)|R = E_\mu(\mathcal{P}^{l'}_\mu)|R = e_R$ in the interval $[0, \mu)$. Thus, by our assumption about the expected number of packets sent, the total number of packets sent by S and R in this interval must be at most c, in every execution $E_\mu(\mathcal{P}^l_\mu)$.

Now for any positive $d \leq \mu$, let $\mathcal{P}_d$ denote the packet history in which all packets take time d. We now construct an execution tree E_d consistent with $\mathcal{P}_d$. Let $\mathcal{P}^*_d$ denote the packet history in which every packet is lost. (Of course, R need not deliver the message in $E_d(\mathcal{P}^*_d)$.) In the interval $[0, \mu)$ in $E_d(\mathcal{P}^*_d)$, S receives only the initial input from U_S and R receives no inputs at all; thus we can construct $E_d(\mathcal{P}^*_d)$ so that $E_d(\mathcal{P}^*_d)|S = e_S$ and $E_d(\mathcal{P}^*_d)|R = e_R$ in the interval $[0, \mu)$, where e_S and e_R are the execution prefixes defined in the previous paragraph.

But this says that in the interval $[0, \mu)$ in $E_d(\mathcal{P}^*_d)$, S and R send a total of at most c packets. Let π denote the finite prefix of length c consisting of c copies of the symbol *loss*; that is, Φ^π is the subset of Φ in which the first c packets are all lost. Then by the definition of an execution tree, $E_d(\mathcal{P}^l_d)$ and $E_d(\mathcal{P}^*_d)$ are the same in the interval $[0, \mu)$, for *every* $\mathcal{P}^l_d$ that belongs to Φ^π.

Since the probability of Φ^π is $(1-p)^{-c} = \delta$, and the time before delivery is clearly at least μ in every execution $E_d(\mathcal{P}^l_d)$ for $\mathcal{P}^l_d \in \Phi^\pi$, the expected time for R to deliver in the execution tree E_d is at least $\delta\mu$. ■

6.2 Crashes

In this section, we will assume that clocks are synchronized, that messages can be duplicated but not lost, and that R but not S can crash. Note that quiescence is not needed in our lower bounds — we can force untimely quiescence using a crash.

We begin by noting the following general principle. Suppose that R is able to maintain the time of its last crash (in stable storage), and that it is not required to deliver any message whose initial packet was sent before this time. Then there is a general reduction to a protocol that tolerates only message duplication: each packet is labeled with the time of the initial packet in the current exchange, and R simply throws away any packet for which this label is less than the time of its last crash.

If R does not know the time of its last crash, then we prove a lower bound that contrasts with the algorithm of Theorem 5.

Theorem 8 *Consider a system with synchronized clocks, in which messages can be duplicated and R can crash, and suppose that R does not know the time of its last crash. For every message-delivery protocol, there is an execution e for which at least time $3d_e$ elapses before the delivery of the message.*

Proof. Assume that the claim does not hold; we derive a contradiction. Choose $d < \frac{1}{9}\mu$. We first construct execution e in which both clocks start at 0 and packets take time $d_e = d$. By our assumption about the protocol, we can construct e so that R delivers the message at time $t < 3d$; we then have R crash immediately. Note that the only input S receives in the interval $[0, t-d]$ is the initial input from U_S.

We now construct an execution e' so that both clocks start at 0, packets from S take time d, and packets from R take time μ. Since S receives no inputs from R in the interval $[0, t-d]$, we can construct e' so that $e'|S = e|S$ in the interval $[0, t-d]$; we can also ensure that $e'|R = e|R$ in the interval $[0, t]$, since R will be getting the same inputs from S in this interval in the two executions. Thus R delivers at time t in e'.

We construct execution f in which both clocks start 0 and packets take time $d_f = t$. By assumption, we can construct f so that R delivers at time $t' < 3t$; we then have R crash immediately. Note that since packets take time t, the only input S receives in the interval $[0, t'-t]$ is the initial input from U_S; thus we can construct f so that $f|S = e'|S$ in the interval $[0, t'-t]$.

Finally, we construct execution f' in which R delivers the message twice. A prefix of f' is equal to e'. The remainder of f' is an execution fragment f'_1 which begins at time t with R in its initial state. We replay the packets sent by S in the interval $[0, t'-t]$ so that each takes time t to arrive; we have all other packets take time μ. Thus, the only inputs received by R in the interval $[t, t']$ are the replays of these packets; since $e'|S = f|S$ in the interval $[0, t'-t]$, we can therefore construct f'_1 so that $f'_1|R = f|R$ in the interval $[t, t']$. Thus, R delivers the message in f'_1, and hence delivers it twice in the execution $f' = e'f'_1$. ■

7 Conclusion and Open Problems

We have studied the time bounds one can achieve for message delivery and quiescence with a spectrum of different synchrony assumptions. These results both provide the first precise formulation of some lower bound trade-offs inherent in the problem of message delivery, and reveal some of the relationships between the different types of timing guarantees that are possible in such systems.

A number of possible directions for future work remain open. At the most concrete level, we do not have a lower bound for the case of approximately synchronized clocks that comes close to matching our upper bound when ε is relatively large compared to d.

For the case of message loss and crashes, we feel that a more comprehensive collection of trade-offs can be developed, as we have done in the case of duplication; much of the problem here may lie in finding an appropriate model. Our model involving independent probability of message loss could be explored further; there are also a number of more complicated and more realistic models that could be developed. We are also interested in obtaining tight bounds for the case of processor crashes; one possible approach is the "pumping" technique of Fekete et. al. [5]. The effects of stable storage in the crash model is another direction that could be investigated (knowing the time of the last crash was a simple example in this direction.)

Acknowledgement

The third author would like to thank Arthur Harvey, George Varghese, and Tony Lauck at DEC for discussions out of which some of this work began.

References

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, L. Zuck, "Reliable communication over an unreliable channel," to appear in *Journal of the ACM.* (Also Technical Memo MIT/LCS/TM-447.)

[2] H. Attiya, S. Dolev, J. Welch, "Connection management without retaining information," Technical Report LPCR 9316, Laboratory for Parallel Computing Research, Dept. of Computer Science, The Technion, June 1993.

[3] H. Attiya, R. Rappoport, "The level of handshake required for establishing a connection," *Proc 8th Internation Workshop on Distributed Algorithms*, 1994.

[4] D. Belsnes, "Single message communication," *IEEE Transactions on Communications*, 24(1976).

[5] A. Fekete, N. Lynch, Y. Mansour, J. Spinelli, "The impossibility of implementing reliable communication in the face of crashes," *Journal of the ACM*, 40(1993), pp. 1087–1107.

[6] R. Gawlick, R. Segala, J. Søgaard-Andersen, N. Lynch, "Liveness in timed and untimed systems," *Proc. 21st International Colloquium on Automata, Languages, and Programming*, 1994.

[7] A. Harvey, N. Lynch, Notes on connection management, DEC, 1990.

[8] B. Liskov, L. Shrira, J. Wroclawski, "Efficient at-most-once messages based on synchronized clocks," *ACM Transactions on Computer Systems*

[9] J. Søgaard-Andersen, N. Lynch, B. Lampson, "Correctness of communications protocols: a case study," Technical Report MIT/LCS/TR-589, November 1993.

[10] N. Lynch, M. Tuttle, "Hierarchical correctness proofs for distributed algorithms," *Proc. 6th ACM Symposium on Principles of Distributed Computing*, 1987, pp. 137–151. (Full version in Technical Report MIT/LCS/TR-387.)

[11] N. Lynch, I. Saias, R. Segala, "Proving time bounds for randomized distributed algorithms," *Proc. 13th ACM Symposium on Principles of Distributed Computing*, 1994.

[12] N. Lynch, F. Vaandrager, "Forward and backward simulations part II: timing-based systems," Technical Memo MIT/LCS/TM-487.b, September 1993.

[13] L. Murphy, A.U. Shankar, "Connection management for the transport layer: service specification and protocol verification," *IEEE Trans. on Communications*, 39(1991), pp. 1762–1775.

[14] C. Sunshine, Y. Dalal, "Connection management in transport protocols," *Computer Networks*, 2(1978),.

[15] A. Tanenbaum, *Computer Networks*, Prentice-Hall, 1988.

[16] R. Tomlinson, "Selecting sequence numbers," *ACM Operating Systems Review*, 3(1975).

[17] D. Wang, L. Zuck, "Tight bounds for the sequence transmission problem," *Proc. 8th ACM Symposium on Principles of Distributed Computing*, 1989, pp. 73–83.

[18] R. Watson, "The Delta-t transport protocol: features and experience," *Proc. IEEE Conference on Local Computer Networks*, pp. 399–407, 1989.