Connection Management Without Retaining Information*

(Extended Abstract)

Hagit Attiya
Department of Computer Science,
The Technion, Haifa 32000, Israel,
hagit@cs.technion.ac.il

Shlomi Dolev
Department of Computer Science,
Texas A&M University,
College Station, TX 77843,
shlomi@cs.tamu.edu

Jennifer L. Welch
Department of Computer Science,
Texas A&M University,
College Station, TX 77843,
welch@cs.tamu.edu

Abstract

Managing a connection between two hosts in a network is an important service to provide in order to make the network useful for many applications. The two main sub-problems are the management of serial incarnations of a connection and the transfer of messages within an incarnation. This paper investigates whether it is necessary for connection management protocols to retain state information across node crashes and between incarnations.

1 Introduction

A major problem in the area of communication protocols is managing connections between two hosts across a wide-area network. Each connection between two specific hosts may have many incarnations over time. The task of managing connections between two hosts consists of two components *incarnation managment* and *message transfer*. Incarnation managment is responsible for the management of serial incarnations of connections, while message transfer is responsible for the transfer of messages between the hosts within an incarnation. Each host is associated with a node in the network; one node is the *sender* and the other is the *receiver*. The sender and receiver interact with the hosts and carry out a protocol to *establish* an incarnation of the connection, *transfer* messages from the sender's host to the receiver's host, and eventually *release* the connection. It is possible that a new incarnation of the connection will be requested later. Connection management forms the *transport layer* in the OSI network hierarchy and is built on top of the *network layer*. Transport protocols are at the heart of any wide-area communication network and form the basis for common protocols such as electronic mail, remote procedure call, talk, ftp, and rlogin.

A connection management protocol keeps information about the state of a connection in a *connection record* [30]. Most incarnation management protocols depend on having connection records available after a crash and before opening a new incarnation. But there are costs associated with maintaining connection records. A common technique to retain connection records across a crash is to keep them in stable storage, whose contents are unaffected by a crash. Even in systems which provide stable storage, updating it typically requires long access time and using it burdens the

*This research was supported by the US-Israel Binational Science Foundation, by TAMU Engineering Excellence funds, by NSF Presidential Young Investigator Award CCR-9396098, by Technion V.P.R. funds from B. and G. Greenberg Research Fund (Ottawa), and by the fund for the promotion of research at the Technion.

protocol. In addition, over a long period of time each host may have connections with many other hosts[1], while at any specific moment only a few of these connections have active incarnations. It can become prohibitively expensive to keep connection records indefinitely on each potential or past connection. (Cf. the protocol in [19] that uses synchronized clocks in order to avoid keeping information about inactive connections.) Consequently, we want to know whether it is necessary for the sender and receiver to retain connection records across crashes and between incarnations.

Since connection management is such a basic task, it is important to develop a rigorous basis for precise understanding of the issues concerning the appropriate network assumptions, the desired protocol specifications, and the interactions between the two. This theory should capture the important features of the problem, and still be simple enough that results can be proved formally. We make a step in this direction, focusing on the issue of retaining information for establishing and releasing connections, as well as for message transfer.

For our results, we assume a connectionless network layer, that is, a network layer that only supports primitives to send and receive packets, but does no error checking or flow control (e.g., IP in the TCP/IP protocol suite [10]). We assume that the network does not duplicate packets, and also that any corrupted packets are detected and discarded. In this environment, packets can be lost due to congestion, fragmentation, or a crash of an intermediate node. The network layer may or may not deliver packets in FIFO order. We discuss these assumptions in more detail below.

Our formal treatment of the connection management problem is unique among theoretical work in modeling the ability to release a connection based on network misbehavior. Actual networks can "throw in the towel" and decide a connection should be terminated because of poor performance (cf. [25, pp. 377–378], and the usage of ICMP packets in TCP/IP [10, 28]). To capture this phenomenon, our formal model includes a special NProblem event by which the underlying network indicates that some severe packet transportation problem has occurred, e.g., all packets in transit were lost, or no packet is received during "too long" a time. A protocol that uses these indications can close the connection (and stop delivering messages) when some criterion on packet transportation is below its threshold. The use of NProblem makes it easier to design a protocol, since the nodes can "give up" and close a connection when it occurs. Not surprisingly, it makes proving impossibility results more difficult[2]. In order to make our results as strong as possible, we prove our impossibility results for protocols that rely on NProblem, while our protocols do not rely on NProblem to release connections.

[1] For example, the Internet, which runs TCP/IP, connects more than 200,000 hosts [10].

network	loss	no loss
non-FIFO	NO (Theorem 2.1)	YES (protocol)
FIFO	YES (full paper)	YES

Figure 1: IM, Information not Retained Between Incarnations

network	loss	no loss
non-FIFO	strong NO	weak NO
FIFO	strong NO (full paper)	weak NO (Theorem 2.2)

Figure 2: IM, Information not Retained Across Crashes

We first consider incarnation management (IM) when information is not retained between incarnations and nodes do not crash. The results for this case are summarized in Fig. 1. They indicate that correct incarnation management is not possible for poorly behaved networks. Specifically, we show that correct incarnation management is not possible for networks which are not FIFO and can lose packets, but it is possible for networks either that are FIFO, even if packets can be lost, or that do not lose packets, even if delivery is non-FIFO.

We then consider incarnation management in the presence of crashes, without stable storage. Our results for this case are summarized in Fig. 2. In this case incarnations cannot be managed correctly, although the severity of the misbehavior exhibited depends on the specific network assumptions. In more detail, we show that incarnations cannot be managed correctly, even if the network is FIFO and does not

[2] For example, one of the techniques in [13]—losing all the packets in transit and still requiring the protocol to perform a particular task—cannot be used (or requires more care).

network protocol	non-FIFO	FIFO
fixed	NO	NO (Theorem 3.1)
variable	NO (Theorem 3.2)	YES (full paper)

Figure 3: MT Allowing Fixed or Variable Grace Period, Network Loses Packets

network protocol	non-FIFO	FIFO
FIFO	NO (Corollary 3.4)	YES (protocol)
non-FIFO	YES (protocol)	YES (protocol)

Figure 4: MT, Network does not Lose Packets

lose packets. The error demonstrated is that events of the two ends of the connection are not properly interleaved. In particular one host may transfer messages assuming there is an active incarnation while the other will not participate in this incarnation. Since such "one-sided" connection could be considered non-hazardous, (and thus in Fig. 2 we use the term "weak NO" for it), we then show impossibility for FIFO networks that can lose packets. In this case, the error demonstrated is to fuse together into one incarnation at the receiver the delivery of messages that were transmitted at the sender on behalf of separate incarnations (in Fig. 2 we use the term "strong NO" for such behavior since it demonstrates a severe violation of the requirements). This result assumes that the protocol is either history-independent or finite-state.

We then turn to the subproblem of transferring messages within an incarnation. Ideally, every message transmitted by the sender's host is delivered to the receiver's host exactly once, and no messages are delivered that were not previously sent. In order to focus on the issues related to message transfer (MT), we assume a single infinitely long incarnation that remains open even if crashes occur.

We first consider the case where the capacity (the number of packets that can be in transit at any given time) of the network is bounded. For this case, we have a message transfer protocol which can withstand crashes without stable storage. The protocol works even if the network can lose packets and is not FIFO but it is inefficient if the capacity is large. The protocol can be made more efficient for any capacity when the network is FIFO.

We then consider the message transfer problem when the network has unbounded capacity and can lose packets. One of the main factors in this case is whether the protocol is allowed a "grace period" after a crash, during which messages that are transmitted do not necessarily have to be delivered. Our results for this case are summarized in Fig. 3. They imply that message transfer is possible if and only if the network is FIFO and the protocol is allowed a variable length grace period. In more detail, we show that if the grace period must be fixed, then there is no protocol for this problem, even if the network is FIFO and the message delivery need not be. We further show that if the grace period is allowed to be variable, then there is a protocol for this problem that guarantees FIFO delivery of messages if the network is FIFO but can lose packets. If the network is not FIFO then there is no protocol for this problem.

Finally, we consider unbounded capacity networks that do not lose packets. We show that in non-FIFO networks, achieving FIFO message delivery is impossible in the presence of node crashes, even if the liveness property to be satisfied is very weak. In the other three cases, there are simple protocols for message transfer; see Fig. 4.

The statements of several of our impossibility results formalize beliefs held by practitioners. Known "folk" theorems in the practical community argue that unless some non-trivial timing assumptions are satisfied, nodes must keep connection records indefinitely and possess stable storage that can withstand crashes (cf. [25, Ch 6]). Roughly speaking, the argument is based on the "delayed duplicates" attack, in which old duplicate packets somehow collect in the network and are then delivered to a node in such a way as to trick it into thinking a connection is open when it is not. It is true that (connectionless) network protocols do not detect and eliminate duplicates, but where can these duplicates come from and how can they be collected? For any reasonable type of hardware, duplicates are only caused by some protocol at some layer retransmitting a packet. The popular network protocol IP does not itself do retransmissions[3]; thus the

[3]Cf. [10], p. 98, where the specification of the protocol does

only other possibility is that they come from the data link layer (which is concerned with reliable transmission between adjacent nodes over a physical link). But duplicates at the data link layer are easily avoided.[4] Thus the assumption of arbitrarily delayed duplicates collecting in the network is too pessimistic in many reasonable situations.

Our results imply that even under a seemingly more benign assumption about the network (namely, no spontaneous duplicates), it is in many cases possible to collect duplicates so as to attack protocols for incarnation management and message transfer. This holds even for relaxed protocol requirements, including the existence of a grace period and the possibility of releasing a connection upon network misbehavior.

On the positive side, our bounded capacity message transfer protocol implies a FIFO data link protocol which can withstand crashes even without stable storage, provided that the capacity of the physical link is bounded. This protocol stands in contrast to the result of [13] (see also [5]) that stable storage is needed in order to provide exactly-once delivery if nodes can crash. The impossibility proof of [13] hinges on the ability to keep an unbounded number of packets inside the physical link: Roughly speaking, this number is related to the number of packets used to deliver the first message after a crash, and thus is protocol depended. In reality, the number of packets in transit at a time on a physical link, which is a function of the transmission rate, the length of the link, and the speed of light, *is* bounded and in fact is typically very small (e.g., cf. [14] pp. 24-29, 51-52). Moreover, the buffers used to store the packets at the sender and the receiver have bounded capacity as well. Note that those facts do *not* imply synchrony; a packet can be stored for an unbounded time in a buffer. To model this situation we restrict our view only to the subset of all possible executions in which the number of packets in transit at a time never exceeds the capacity. We use the above fact to show in contrast to [13] that for a physical link with a known upper bound on the capacity, there exists a protocol.

not include retransmission, although implementations are not prohibited from retransmitting.

[4] The alternating bit protocol with stable storage to tolerate node crashes could be used [5]; the amount of stable storage needed at a node is just one bit for every adjacent link, which is feasible, unlike keeping a connection record for thousands of connections. Alternatively, our bounded capacity protocol, which does not rely on stable storage, can be used.

Our results that assume a non-FIFO network are clearly applicable to the environment of a connectionless network layer, where packets are routed independently. The results that assume a FIFO network layer are theoretically interesting since the lower bounds are made technically stronger. They also have practical relevance to a connection-oriented network layer that can control routing decisions so as to ensure FIFO delivery. They are also relevant to data link initialization procedures, which provide a reliable connection between nodes that are physically connected.

Connection management has been studied intensively in the practical literature and many ingenious protocols have been suggested (e.g., [8, 9, 24, 25, 27, 30, 31]). All these protocols rely on some combination of timers, packet delay bounds, synchronized clocks and unique incarnation identifiers; it has been argued informally that some combination of these assumptions is necessary [31]. Our work complements these protocols by identifying precisely which assumptions on the system are necessary and sufficient, and exactly which requirements from the protocol it is impossible to achieve in certain cases. Other work that complements ours is the vast literature on verification of communication protocols, including, for example, [15, 17, 23]. These papers concentrate on verifying known protocols rather than investigating whether the assumptions they rely on are necessary.

There is some prior work on impossibility results for connection management. Our impossibility result for message transfer, where fixed grace period is allowed, on a FIFO network that can lose packets improves the result in [13], which assumed the stronger progress property that starting *immediately* after the last crash, all messages transmitted must be delivered. Belsnes [7] studies how many packet exchanges are necessary in order to manage incarnations, under various requirements and assumptions. LeLann and LeGoff [18] show that a connection cannot be established by protocols of a particular form. Other theoretical studies of communication protocols have mostly concentrated on the data link layer [1, 3, 13, 16, 20, 22, 26, 29]. Most of this work concerns implementing protocols using *bounded size* packets, an issue we do not address. Our protocols for message transfer on bounded-capacity FIFO networks use an idea from self-stabilizing protocols for cleaning the system with a new label. (This idea was first employed in [11] and later used in e.g. [2], [12].)

Due to space constraints, most details are excluded in the body of this extended abstract. Full details can be found in [4].

2 Incarnation Management

Problem Definition: The system consists of two nodes S (sender) and R (receiver), a host at each node, and a network connecting S and R. See Figure 5; an action labeling an arrow entering (leaving) a component is an input (output) action of the component. S and R form the connection management protocol. The network supports sending packets (Send_S and Send_R), receiving packets (Recv_S and Recv_R) and might lose packets (Lose).

Ideally what happens is that the host at S requests a connection (Req-Con_S), S communicates this to R and R checks with its host (Req-Con_R). If R's host is agreeable (Con_R), R communicates this to S, who then tells its host (Con_S). Now the host at S transmits messages (Transmit) and S and R run a message transfer protocol so that the messages are delivered to R's host (Deliver). The host at either S or R can unilaterally decide to end the connection (Req-Dis_S or Req-Dis_R); once S and R have terminated the connection, the hosts are notified (Dis_R and Dis_S). There is a third way that a connection can be released: if the network is experiencing problems, either S or R is notified (NProblem_S and NProblem_R) and then they are free to decide whether to terminate any existing connection. We assume that NProblem_X occurs if and only if some predicate on network behavior is true; we restrict attention here to *loss(2)-only* predicates that are true only if at least two packets are lost. Crashes are modeled with Crash actions for S and R, which cause the node's state to return to a special "recovering" state. S and R can also take internal steps (e.g., to retransmit a packet).

No Crashes But No Storage Between Incarnations: We first consider a system in which nodes do not crash but the nodes do not keep information in between incarnations of a connection; we call such protocols *amnesic*. We show that there is an amnesic protocol if and only if the network is FIFO. (The full paper contains a careful definition of amnesic; it is a little delicate in order to rule out S and R "cheating" by storing incarnation information in packets that are in transit.)

The theorem is proved using a technique, called *stealing*, to prove impossibility results for non-FIFO losing networks. Stealing does not rely on node crashes or spontaneous duplications. We take a specific "reference" execution and we replay successively longer prefixes of the reference execution. Each replay ends with the loss of a different packet. Because the protocol must tolerate the loss of a single packet, it must gracefully finish any pending task. Yet it is possible that the lost packet was not lost but only delayed in the non-FIFO network. Thus we can steal a single packet from each replay and keep it in the network. Then we deliver the packets we have collected to the receiver, tricking it into acting as if the reference execution is executed. A similar technique is used in [1].

Theorem 2.1 *There is no amnesic incarnation management protocol for non-FIFO losing networks, for*

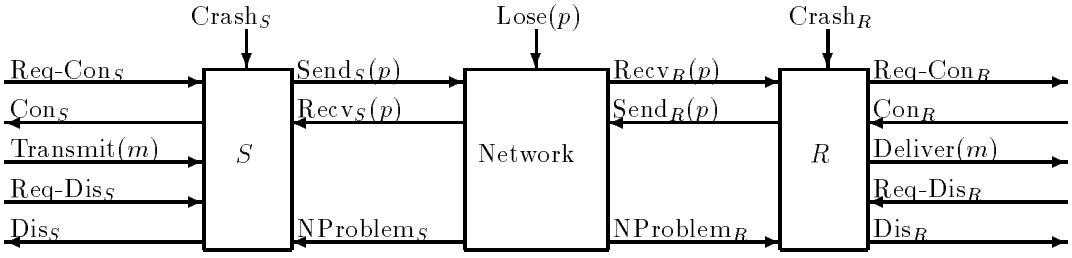

Figure 5: System architecture; hosts are not shown.

any loss(2)-only NProblem predicates.

Sketch of proof: Assume in contradiction there is such a protocol. The definition of a correct protocol implies that there exists a "ping-pong" schedule γ with no Lose events whose restriction to host actions[5] is Req-Con$_S$ Req-Con$_R$ Con$_R$ Con$_S$ Transmit(m) Deliver(m) for some message m. (A schedule is ping-pong, roughly speaking, if only one packet is ever in transit at a time.)

Let $s_1, \ldots, s_k$ be the sequence of packets sent by S in γ. We inductively build a schedule $\gamma_1\delta_1\epsilon_1 \ldots \gamma_k\delta_k\epsilon_k$ as follows. Let γ_i be the prefix of γ through the sending of s_i.

Assume $\rho_{i-1} = \gamma_1\delta_1\epsilon_1 \ldots \gamma_{i-1}\delta_{i-1}\epsilon_{i-1}$ is a schedule after which S and R are in their initial states and $s_1, \ldots, s_{i-1}$ are in transit. We show the analogous statement for i.

Then $\rho_{i-1}\gamma_i$ is a schedule since S and R start γ_i in the same states that they end ρ_{i-1} in and the network is non-FIFO.

We now define δ_i. If Transmit(m) is not in γ_i, then let δ_i be the empty string. Suppose Transmit(m) is in γ_i. We use the definition of a correct protocol to show that there exists an extension δ_i in which s_i is never received yet there is a matching Deliver(m). This is true because s_i might have been lost, the network is non-FIFO, yet the loss of a single packet cannot close the connection (since the NProblem predicates are loss(2)-only).

We now define ϵ_i to be Req-Dis$_S$ followed by the events needed to close the incarnation, followed by enough Send events to empty the outgoing packet queues at S and R. (The Send events are needed by the precise definition of amnesic.) At the end of ϵ_i, S and R are in their initial states. The packets $s_1, \ldots, s_i$ are still in transit.

Note that ρ_k is a schedule with no Lose events after which $s_1, \ldots, s_k$ are in transit. Thus ρ_kRecv$_R(s_1) \ldots$ Recv$_R(s_k)$ is a schedule and it has an extension in which only R takes steps that ends with Deliver(m). However, in each $\gamma_i\delta_i\epsilon_i$ making up ρ_k there is a matching delivery in δ_i if the message m is transmitted in γ_i. Thus the last Deliver(m) has no matching Transmit, which is a violation. ■

[5] Req-Con, Con, Transmit, Deliver, Req-Dis, and Dis

In the full paper we show that there is an amnesic FIFO incarnation management protocol when the network is FIFO but can lose messages. Moreover this protocol uses bounded memory and only releases incarnations if explicitly requested to do so by either host; i.e., it doesn't depend on any NProblem events. The message transfer liveness requirement satisfied is that the sequence of messages delivered is equal to the sequence transmitted.

Roughly speaking the incarnation management protocol synchronizes S and R on the incarnation by using the header 2 for packets that indicate the opening of a connection. Then, similarly to [6], the alternating bits 0 and 1 are used within an incarnation, both to transfer data items and to synchronize disconnections.

The protocol requires R to continue sending acknowledgments for the packets it receives even after Dis$_R$ and before the next incarnation begins. Note that such a behavior does not violate the amnesic property of the system because R does not need to have any information on the closed connection in order to acknowledge incoming packets.[6] In the full paper we show that this behavior is necessary for any such protocol.

Crashes are Possible: We present three impossibility results for incarnation management when nodes can crash but have no stable storage. They all indicate that connection management is not possible; they differ in the particular bad behavior exhibited and the particular restrictions placed on the structure of the protocol.

All three of the results are proved using a technique called *pumping*. (This technique is also used in two results concerning message transfer.) Pumping is a generalization of the main technique used to prove the impossibility result of [13]. Unlike the stealing technique, pumping does not rely on non-FIFO or losing behavior of the network, although it does assume node crashes. Instead of stealing packets one at a time, we get the desired sequence of packets in transit by strictly alternating between S and R and each time replaying a little more of each one's history from the reference schedule and crashing the other one (thus "pumping up" the sequence of packets in transit).

[6] Tanenbaum [25, page 399] does not consider the possibility of this behavior, although no reason is given.

The next theorem shows that even if the network is completely reliable, never losing packets and delivering them in FIFO order, there is no protocol. The contradiction is reached by showing that any proposed protocol can be forced into an execution in which S replays its side of an incarnation while R remains disconnected and then vice versa. S and R are executing correctly considered in isolation, but they are not correctly synchronized (interleaved) with each other.

Theorem 2.2 *There is no incarnation management protocol for FIFO non-losing networks in the presence of node crashes.*

While the above result shows a violation of the specification for incarnation management, one might argue that opening a "one-sided" connection is not so bad. In particular, data is not being delivered in the wrong incarnation. This reasoning motivates our second and third impossibility results, which show that any protocol that is history-independent (defined in the full paper) or finite state can be tricked into fusing together two separate connections at S into one at R. That is, messages from an earlier incarnation are delivered during a later incarnation, thus violating the message grouping property. This is a severe violation since different incarnations can be established on behalf of completely different application programs. These results rely on the network losing packets.

3 Message Transfer

We now consider the problem of recovering from crashes while guaranteeing some sort of reliable message transfer within a single infinitely long incarnation. We are interested in whether or not stable storage is required in the presence of crashes[7]. Thus throughout this section we assume nodes can crash.

The model is simplified by considering only the actions Transmit, Deliver, Send, Recv, Step, Crash, and optionally Lose. A *message transfer* protocol ensures that every message delivered was previously transmitted. An *exactly-once* message transfer protocol guarantees that if there is finite number of Crash events, then every Transmit following the last Crash has a matching Deliver.

[7]Since the message transfer problem does not consider multiple incarnations, studying message transfer *without* node crashes would not provide any possibilities for failing to retain information according to our definition.

Bounded Capacity: We now assume that at any time, at most *cap* packets are in transit from S to R (or vice versa), as will usually be the case when S and R are communicating directly over a physical link. Our interest in this assumption was motivated by the observation that several impossibility results for message transfer rely on the ability to collect an unbounded number of packets in the network. In particular, [13] shows that exactly-once non-FIFO message transfer is not possible for FIFO reliable networks in the presence of crashes. In contrast, we describe in the full paper an exactly-once FIFO message transfer protocol for a non-FIFO losing network in the presence of crashes if the capacity of the network is bounded. The protocol requires that $\Omega(cap)$ packets be sent for each message to be transferred. Thus if *cap* is more than a small constant, the protocol is inefficient.

If, in addition to having bounded capacity, the network is FIFO then there is a more efficient exactly-once message transfer protocol, which we now describe. This protocol has the pleasing property that the number of packets sent depends on the capacity only after a crash has occurred. Note that $2cap$ is an upper bound on the total number of packets that can be in transit in both directions.

When recovering from a crash, S repeatedly sends (initializing) packets $(init, 1)$ until it receives $(ack, 1)$; then, S repeatedly sends $(init, 2)$ until it receives $(ack, 2)$ and so on, until S repeatedly sends $(init, 2cap + 2)$ and receives $(ack, 2cap + 2)$. At this point, S starts to transmit messages as follows: To transmit a message m, S repeatedly sends packets $(m, 1)$ until it receives $(ack, 1)$; then, S repeatedly sends $(m, 2)$ until it receives $(ack, 2)$. Once $(ack, 2)$ is received, S is ready to send the next message m' in the same manner; that is, S sends $(m', 1)$ until it receives $(ack, 1)$; then, S sends $(m', 2)$ until it receives $(ack, 2)$.

R acknowledges any incoming packet with header h by sending the packet (ack, h). When R recovers from a crash, it first waits for a non-*init* packet with header 1. Only at this point, R starts to deliver messages as follows: R delivers a message m to its host only upon receiving $(m, 1)$ and immediately afterwards $(m, 2)$.

The correctness of the protocol is based on the fact that at the time of a crash by S, there will always be at least one header h among $\{1, \ldots, 2cap + 2\}$ that is not in transit. Once S reaches h in its post-Crash "cleaning" (sending *init* packets), the system is in a good state from which correct behavior follows.

Losing Networks: We now consider networks that can lose packets and whose capacity is unbounded. We allow an exactly-once message-transfer protocol to have a "grace period" after a crash during which messages need not be delivered. If the length of the grace period (in terms of the number t of messages that do not have to be delivered) is fixed, then there is no protocol, even if the network is FIFO. However, if the length of the grace period can be a function $t()$ of the current state of the system (i.e., is "variable"), then there is a protocol if and only if the network is FIFO.

Theorem 3.1 *For any $t \geq 0$, there is no non-FIFO exactly-once message transfer protocol with t-fixed grace period for a FIFO losing network in the presence of node crashes.*

The proof of Theorem 3.1 (which uses the pumping technique) no longer holds when the grace period can be variable length. In fact, as we show in the full paper, for losing networks there is an exactly-once protocol with $t()$-variable grace period if the underlying network is FIFO. The protocol is an adaptation of a self-stabilizing algorithm from [12].

We now show that the assumption that the network provides FIFO delivery of packets is essential in order to obtain an algorithm, even if the grace period is variable. This result relies on the assumption that the number of packets that can accumulate in the network is not bounded. The proof uses the stealing technique.

Theorem 3.2 *There is no non-FIFO exactly-once message transfer protocol with $t()$-variable grace period for a non-FIFO losing network in the presence of node crashes, for any function t.*

Non-Losing, but Non-FIFO, Networks: We now assume that the network never loses packets but might deliver them arbitrarily out of order. Under this assumption, FIFO exactly-once message transfer is not possible. Note that Theorem 3.2 does not imply this result, since that theorem relied on having a losing network; yet this result does not imply Theorem 3.2, since this result only shows the impossibility of a *FIFO* protocol. In fact, when the network does not lose packets and the ordering properties of the protocol match those of the network, there is a trivial protocol which guarantees (1) FIFO exactly-once message transfer on a FIFO non-losing network and (2) non-FIFO exactly-once message transfer on a non-FIFO non-losing network.

The next theorem implies that, without stable storage, providing FIFO behavior for messages, when packets are not delivered in FIFO manner, is impossible in the presence of host crashes, even if the network does not lose packets and the protocol only has to deliver *some* messages. In more detail, a message transfer protocol is *at-most-once* if whenever there is a finite number of Crashes and an infinite number of Transmits, then there is an infinite number of Delivers.

Theorem 3.3 *There is no FIFO at-most-once message transfer protocol for a non-FIFO non-losing network in the presence of node crashes.*

Corollary 3.4 *There is no FIFO exactly-once message transfer protocol with $t()$-variable grace period for non-FIFO non-losing networks in the presence of node crashes.*

4 Discussion

We have studied the necessity of retaining information between incarnations and across node crashes for two aspects of the connection management problem: incarnation management and message transfer. We proved that when state information is not saved between incarnations, the problem is solvable if and only if the network is FIFO. We also showed that incarnation management is not possible in the presence of crashes without stable storage. Furthermore, we showed that message transfer is possible in the presence of crashes without stable storage when packets can be lost if and only if the network is FIFO *and* the protocol is allowed a variable grace period after a crash during which it need not deliver messages.

When packets are not lost, message transfer is possible if and only if either the network is FIFO or the protocol need not be. On the positive side, we have devised a data link initialization procedure that can withstand node crashes without stable storage, provided that the capacity of the physical link is bounded.

Our work forms another step in formalizing issues that arise at the transport layer in communication protocols. To the best of our knowledge, ours is the first theoretical study of this problem to incorporate the following practical features: indication of severe network misbehavior, grace period after a crash and bounded capacity of the physical links.

Many interesting issues remain to be studied, including flow control and buffering, analysis of the time-based techniques used in practical connection management protocols, and problems that arise in trying to do a clean disconnect [10, 25]. We hope that our model, definitions and techniques will be of help in continuing in these directions. In particular, we believe the NProblem action, coupled with appropriate predicates, can be used to encapsulate timer-based mechanisms used to detect severe errors.

Another interesting aspect is to explore quantitative considerations such as the number of packets that have to be sent in the cases where incarnation management or message transfer is possible. For example, it would be interesting to know whether $\Omega(cap)$ packets are required in order to clear the connection after a crash, when the network is FIFO and the capacity, cap, is bounded.

Acknowledgments: We thank Reuven Cohen, Alan Fekete, Michael Kate, Steve Liu and Rinat Rappoport for helpful discussions.

References

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang and L. Zuck, *Reliable Communication over Unreliable Channels,* Technical Report YALEU/DCS/TR-853, Department of Computer Science, Yale University, October 1992. To appear in *Journal of the ACM.*

[2] Y. Afek and G. M. Brown, "Self-Stabilization of the Alternating-Bit Protocol," *Proceedings of the 8th IEEE Symposium on Reliable Distributed Systems,* (1989), pp. 10–12.

[3] A. Aho, A. Wyner, M. Yannakakis and J. Ullman, "Bounds on the Size and Transmission Rate of Communication Protocols," *Computers and Mathematics with Applications,* Vol. 8, No. 3, pp. 205–214, 1982.

[4] H. Attiya, S. Dolev and J. L. Welch, *Memory Requirements for Connection Management,* Technical Report LPCR #9316, Department of Computer Science, The Technion, Haifa, June 1993.

[5] A. Baratz and A. Segall, "Reliable Link Initialization Procedures," *IEEE Transactions on Communication*, Vol. T-COM-36, No. 2, pp. 144–152, February 1988.

[6] K. Bartlett, R. Scantlebury, and P. Wilkinson. "A Note on Reliable Full Transmission over Half-Duplex Links," *Communications of the ACM,* 12(5):260-261.

[7] D. Belsnes, "Single-Message Communication," *IEEE Transactions on Communication*, Vol. T-COM-24, No. 2, pp. 190–194, February 1976.

[8] E. W. Biersack and D. Feldmeier, "A Timer-Based Connection Management Protocol with Synchronized Clocks and its Verification," to appear in *Computer Networks and ISDN systems.*

[9] D. Cheriton, "Sirpent(TM): A High Performance Internetworking Approach," *Proc. ACM SIGCOMM,* pp. 158–169, 1989.

[10] D. Comer, *Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture,* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[11] E. W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Communications of the ACM,* Vol. 17, No. 11, pp. 643-644, 1974.

[12] S. Dolev, A. Israeli and S. Moran, "Resource Bounds for Self Stabilizing Message Driven Protocols," *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing,* pp. 281-293, August 1991.

[13] A. Fekete, N. Lynch, Y. Mansour and J. Spinelli, "The Impossibility of Implementing Reliable Communication in the Face of Crashes," to appear in *Journal of the ACM*. Also: Technical Memo MIT/LCS/355.c, Laboratory for Computer Science, Massachusetts Institute of Technology, September 1991.

[14] F. Halsall, *Data Communications, Computer Networks and OSI,* Addison-Wesley, 1988.

[15] J. F. Kurose and Y. Yemini, "The Specification and Verification of a Connection Establishment Protocol using Temporal Logic," in *Protocol Specification, Testing and Verification II* (C. A. Sunshine, Ed), North-Holland, New-York, 1982.

[16] R. Ladner, A. LaMarca and E. Tempero, "Counting Protocols for Reliable End-to-End Transmission," Technical Report 92-10-06, Department of Computer Science, University of Washington, 1992.

[17] S. S. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Trans. on Software Engineering,* Vol. 10, pp. 325–342, July 1984.

[18] G. LeLann and H. LeGoff, "Verification and Evaluation of Communication Protocols," *Computer Networks,* Vol. 2, pp. 50–69, 1978.

[19] B. Liskov, L. Shrira and J. Wroclawski, "Efficient At-Most-Once Messages Based on Synchronized Clocks," *ACM Trans. on Computers*, Vol. 9, No. 2, pp. 125–142.

[20] N. Lynch, Y. Mansour and A. Fekete, "The Data Link Layer: Two Impossibility Results," *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing,* pp. 149–170, August 1988.

[21] N. Lynch and M. Tuttle, "An Introduction to Input/Output Automata," *CWI Quarterly*, Vol. 2, No. 3, pp. 219–246, September 1989.

[22] Y. Mansour and B. Schieber, "The Intractability of Bounded Protocols for non-FIFO Channels," *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing,* pp. 59–72, August 1989.

[23] S. L. Murphy and A. U. Shankar, "Connection Management for the Transport Layer: Service Specification and Protocol Verification," *IEEE Trans. on Communications,* Vol. 39, No. 12, pp. 1762–1775, December 1991.

[24] C. A. Sunshine and Y. K. Dalal, "Connection Management in Transport Protocols," *Computer Networks,* Vol. 2, pp. 454–473, 1978.

[25] A. Tanenbaum, *Computer Networks*, 2nd ed., Prentice Hall, 1988.

[26] E. Tempero and R. Ladner, "Tight Bounds for Weakly Bounded Protocols," *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing,* pp. 205–209, August 1990.

[27] R. S. Tomlinson, "Selecting Sequence Numbers," *Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop,* pp. 11–23, 1975; in *ACM Operating Systems Review*, Vol. 9, No. 3, 1975.

[28] *Transmission Control Protocol*, DARPA Network Working Group Report RFC-793, University of Southern California, September 1981.

[29] D.-W. Wang and L. Zuck, "Tight Bounds for the Sequence Transmission Problem," *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing,* pp. 73–83, August 1989.

[30] R. W. Watson, "Timer-based Mechanisms in Reliable Transport Protocol Connection Management," *Computer Networks,* Vol. 5, pp. 47–56, 1981.

[31] R. W. Watson, "The Delta-t Transport Protocol: Features and Experience," *Proc. IEEE Conf. on Local Computer Networks,* pp. 399–407, 1989.