A Formal Venture into Reliable Multicast Territory

Carolos Livadas Nancy A. Lynch
Lab. for Computer Science
Massachusetts Institute of Technology

November 6, 2002

Abstract

In this paper, we present a formal model of the reliable multicast service that ensures eventual packet delivery with, possibly, some timeliness guarantees. This model dictates precisely what it means to be a member of the reliable multicast group and which packets are guaranteed delivery to which members of the group. Moreover, it is reasonable, implementable, and broad; that is, it captures the intended behavior of a large collection of reliable multicast protocols. We also present a formal model of the Scalable Reliable Multicast (SRM) protocol [1]. We show that our model of SRM is safe, in the sense that it is a faithful implementation of our model of the reliable multicast service; that is, it may only deliver appropriate packets to each member of the reliable multicast group. We also show that, under certain constraints, the implementation is live, in the sense that it guarantees the timely delivery of the appropriate packets to the appropriate members of the reliable multicast group.

1 Introduction

With the increasing use of the Internet, multi-party communication and collaboration applications are becoming mainstream. Reliable multicast is a communication service that facilitates such applications. In the recent past, a slew of protocols have been proposed to reliably multicast packets efficiently [1–4, 7, 8]. However, reliability in the multicast setting has assumed many meanings, ranging from in-order eventual delivery to timely delivery where a small percentage of packet losses is tolerable. The many notions of reliability stem from the varying assumptions regarding the communication environment and the goals and requirements of the applications to which particular reliable multicast protocols cater.

Most often, the behavior of reliable multicast protocols is described informally. To our surprise, a protocol's description is seldom accompanied by a precise definition of its reliability guarantees. In its simplest form, reliability is informally defined as the eventual delivery of all multicast packets to all group members; other notions of reliability include ordering, no-duplication, and timeliness guarantees. Although intuitive, this simplistic reliability definition does not precisely specify which packets are guaranteed delivery to which members of the group, especially when the group membership is dynamic. Moreover, protocol descriptions put little emphasis on the behavior, or the analysis of the behavior, of the protocol when the group membership is dynamic, either due to failures or frequent joins and leaves. As hosts become more mobile, a better understanding of the behavior of such services and protocols in the context of a dynamic group membership is increasingly important.

In this paper, we present a formal model of the reliable multicast service, which we henceforth refer to as the *reliable multicast specification* (RMS). Specifying the reliable multicast service is not straightforward. The plethora of reliable multicast protocols cater to diverse applications that impose diverse correctness and performance requirements. Clearly, capturing the functionality of all reliable multicast protocols using a single specification would be quite complex and unwieldy. Our reliable multicast service specification formalizes the behavior of a number of protocols, such as SRM [1] and LMS [7], that strive to provide eventual delivery with, possibly, some timeliness guarantees. We stipulate that, in the context of dynamic group membership, membership is intrinsically intertwined with reliability; that is, membership and reliability must be addressed together. Thus, our specification dictates precisely what it means to be a member of a reliable multicast group and which packets are guaranteed delivery to which members of the reliable multicast group. We parameterize our specification with a delivery latency bound, which specifies an upper bound on the latency incurred to reliably deliver multicast packets. This parameterization results in a reliable multicast service specification that encompasses the behavior of a collection of reliable multicast protocols, some with loose and others with potentially stringent timeliness guarantees.

We also present a formal model of the Scalable Reliable Multicast (SRM) protocol [1]. Our model of SRM, which we henceforth refer to as the *reliable multicast implementation* (RMI), involves several components with distinct functionalities, such as the maintenance of the reliable multicast group membership and the packet loss recovery. This decomposition simplifies the reasoning and facilitates future modifications to the implementation. We show that RMI is safe, in the sense that it is a faithful implementation of RMS; that is, it may only deliver appropriate packets to each member of the reliable multicast group. We also show that, under certain constraints, RMI is live, in the sense that it guarantees the timely delivery of the appropriate packets to the appropriate members of the group.

The rest of the paper is organized as follows. Section 2 presents our modeling framework. Section 3 presents the abstract view of the physical system that we adopt in our work. Section 4 presents RMS and its eventual and timely reliability properties. Section 5 presents RMI, derives constraints on RMI's packet loss recovery parameters, and analyzes RMI's safety and liveness with respect to RMS. Finally, Section 6 presents the paper's contributions and future work directions.

2 Modeling Framework and Notation

In this paper, we use the *timed input/output (I/O) automaton* (TIOA) modeling framework (introduced as the *general timed automaton* model in Ref. 6); a framework for modeling timed systems. A timed I/O automaton A is a state-machine in which transitions are labeled by *actions*. A's actions ($acts(A)$) are partitioned into *input* ($in(A)$), *output* ($out(A)$), *internal* ($int(A)$), and *time-passage* sets. Time-passage actions model the passage of time. The input and output actions of A are collectively referred to as *external*; denoted $ext(A)$. Input, output, and time-passage actions are collectively referred to as *visible*; denoted $vis(A)$. A timed I/O automaton A is defined by its *signature* (input, output, internal, and time-passage actions), states ($states(A)$), start states ($start(A)$), and state-transition relation ($trans(A)$). The state-transition relation of A is a cross product of states, actions, and states that dictates A's allowable transitions; that is, $trans(A) \subseteq states(A) \times acts(A) \times states(A)$ and a transition of A from s to s' through action π is denoted by the tuple (s, π, s').

A *timed execution fragment* α of A is a finite or infinite alternating sequence, $\alpha = s_0\pi_1 s_1\pi_2 s_2 \ldots$, of states and actions consistent with A's state-transition relation; that is, $s_k \in states(A)$, $\pi_{k+1} \in acts(A)$, and $(s_k, \pi_{k+1}, s_{k+1}) \in trans(A)$, for all $k \in \mathbb{N}$. For any two timed execution fragments α and α' of A, we use the notation $\alpha \leq \alpha'$ to denote that α is a prefix of α'. A timed execution fragment of A is *admissible* if an infinite amount of time elapses within the particular fragment. An admissible timed execution fragment α of A is *fair* when no action is enabled in every state of a suffix of α without appearing in the given suffix. The time of occurrence of an action π_k, for $k \in \mathbb{N}^+$, within a timed execution fragment α of A is the time elapsing within α prior to the occurrence of π_k. Letting $s, s' \in \mathit{states}(A)$ be any two states occurring in a timed execution fragment α of A, we use the notation $s \leq_\alpha s'$ ($s <_\alpha s'$) to denote that the particular occurrence of s appears no later than (prior to, respectively) the particular occurrence of s' in α.

The *timed trace* β of a timed execution fragment α of A is the sequence of visible actions in α, each paired with its time of occurrence. For any two timed traces β and β' of A, we use the notation $\beta \leq \beta'$ to denote that β is a prefix of β'.

A *timed execution* of A is a timed execution fragment of A that begins in one of A's start states. We let $\mathit{aexecs}(A)$ denote the set of all admissible timed executions of A, $\mathit{attraces}(A)$ denote the timed traces of all executions in $\mathit{aexecs}(A)$, $\mathit{fair\text{-}aexecs}(A)$ denote the set of all fair admissible timed executions of A, and $\mathit{fair\text{-}attraces}(A)$ denote the timed traces of all executions in $\mathit{fair\text{-}aexecs}(A)$.

Two timed I/O automata A_1 and A_2 are *compatible* if $\mathit{int}(A_i) \cap \mathit{acts}(A_j) = \emptyset$ and $\mathit{out}(A_i) \cap \mathit{out}(A_j) = \emptyset$, for $i, j \in \{1, 2\}, i \neq j$. The composition of compatible timed I/O automata yields a timed I/O automaton. The *hiding* operation reclassifies output actions of a timed I/O automaton as internal. Letting A, B be timed I/O automata with the same external interface, B *implements* A, denoted $B \leq A$, when its external behavior is allowed by A; that is, when $\mathit{attraces}(B) \subseteq \mathit{attraces}(A)$. The implementation relation among two timed I/O automata is often shown by defining a *timed simulation relation*; that is, relating states of B to states of A and showing that for any step of B there is a timed execution fragment of A with the same timed trace as the step of B that preserves the state relation.

We use a *precondition-effect* style notation to define the state-transition relations of timed I/O automata. Moreover, we use the notation $S_1 \cup= S_2$, $S_1 \setminus= S_2$, and $s :\in S$ as shorthand for $S_1 := S_1 \cup S_2$, $S_1 := S_1 \setminus S_2$, and the assignment of an arbitrary element of S to the variable s.

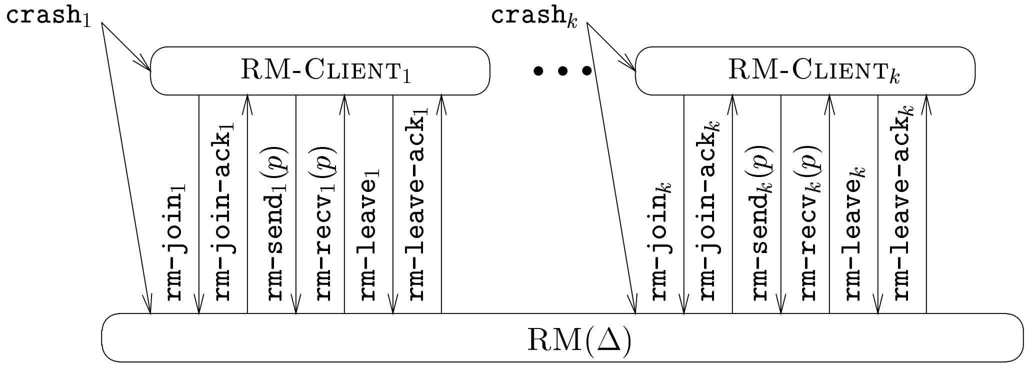
3 The Physical System

We assume that the physical system is comprised of an infinite set of hosts that interact through an underlying network. This network involves a set of interconnected routers. Each host is connected to a particular router of the underlying network; for each host, we refer to this particular router as the *gateway router* of the particular host. Hosts and routers are connected among themselves through bi-directional communication links.

We assume that all hosts are of comparable processing power and storage resources. Resident on each host are a set of processes. We assume that hosts are symmetric in the sense that the same set of processes reside on each host. The set of processes on each host consists of a single application process and several additional communication service processes. Henceforth, we refer to the application process at each host as the *client* at the given host. The communication service processes, either individually or collectively, provide the communication services required by the client. For instance, the IP unicast service may be modeled as a set of processes, one such process for each host. Clients may thus exchange IP unicast packets through their respective IP unicast processes; these may in turn interact with the hosts' gateway routers.

In terms of system faults, we consider only host crashes and packet drops on the communication links. Once a host crashes it remains crashed thereafter. A host is said to be *operational* prior to crashing and to have *crashed* thereafter. All the processes on each host are *fate-sharing*; that is, if a host crashes, then all of its processes crash. Router failures and network partitions are assumed to be ephemeral. Such failures are modeled as numerous consecutive packet drops.

Figure 1 Reliable Multicast Specification Component Interaction



Since crashes are assumed to be permanent, we model host restarts implicitly. We think of the restarting of a host as its reincarnation as a completely new host; that is, after crashing, a host may assume the identity of another host that has up to that point in time been idle. This modeling simplification is equivalent to explicitly modeling host restarts and having hosts choose a unique host identifier each time they restart. Such an identifier could involve, for instance, the processor identifier and an infinite reincarnation counter that is stable across crashes.

4 Reliable Multicast Specification (RMS)

We abstractly model the reliable multicast service as a single component that interacts with all client processes. Thus, the reliable multicast service encapsulates the behavior of all communication service processes at all hosts and the underlying network. For simplicity, we assume that there is a single reliable multicast group. Since we assume a single client per host and a single reliable multicast group, we do not distinguish among the client process and the host when considering reliable multicast group membership. In fact, we often use the terms client and host interchangeably.

Throughout our treatment of reliable multicast, we adopt the packet naming scheme used by Floyd *et al.* [1]. In this scheme, clients (applications) assign unique sequence numbers to each packet they multicast. These sequence numbers are assigned in a continuous fashion as hosts join, leave, and rejoin the reliable multicast group; that is, consecutive packets sent by each host are assigned consecutive sequence numbers. Thus, packets are uniquely and persistently identified by a pair involving their source host and their sequence number. Since the clients (applications) are responsible for naming packets, packets are referred to as *application data units* (ADUs).

4.1 Formal Model

We formally specify the reliable multicast service and each of the client processes using timed I/O automata. The automaton RM(Δ), for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, models the reliable multicast service. RM(Δ) defines what it means to be a member of the reliable multicast group and specifies precisely which packets are guaranteed delivery to each member of the reliable multicast group. The parameter Δ specifies an upper bound on the amount of time required by the reliable multicast service to reliably deliver each packet. The automaton RM-Client$_h$ models the client at the host h. We let RM-Clients denote the composition of all client automata and RM$_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, denote the composition of the reliable multicast service and all client automata; that is, RM$_S(\Delta)$ = RM(Δ) $\times$ RM-Clients. Figure 1 depicts the interaction of the RM(Δ) and RM-Client$_h$, for $h \in H$, automata.

We proceed by presenting some preliminary definitions and, subsequently, defining the RM(Δ) and

Figure 2 Reliable Multicast Specification Definitions

H Set of all hosts.

$Status = \{\texttt{idle}, \texttt{joining}, \texttt{leaving}, \texttt{member}, \texttt{crashed}\}$

$P_{\text{RM-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{RM-CLIENT}}$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$suffix(p) = \{\langle s,i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \leq i\}$

RM-CLIENT$_h$ automata.

4.1.1 Preliminary Definitions

Figure 2 includes several set definitions pertaining to our reliable multicast service specification. H is the set of all hosts that could potentially participate in the reliable multicast communication.

The set *Status* consists of all possible valuations of the reliable multicast membership status of a host. The value `idle` indicates that the host is *idle* with respect to the reliable multicast group; that is, it is neither a member, nor in the process of joining or leaving the reliable multicast group. The value `joining` indicates that the host is in the process of joining the reliable multicast group; that is, the client has issued a request to join the reliable multicast group and is awaiting an acknowledgment of this join request from the reliable multicast service. The value `leaving` indicates that the client is in the process of leaving the reliable multicast group; that is, the client has issued a request to leave the reliable multicast group and is awaiting an acknowledgment of this leave request from the reliable multicast service. The value `member` indicates that the client is a member of the reliable multicast group. The value `crashed` indicates that the host has crashed.

The set $P_{\text{RM-CLIENT}}$ represents the set of packets that may be transmitted by the client processes using the reliable multicast service. According to the ADU naming scheme described above, data segments are identified by their original source and a sequence number. Thus, for any packet $p \in P_{\text{RM-CLIENT}}$ the operations $source(p)$, $seqno(p)$, and $data(p)$ extract the source, sequence number, and data segment corresponding to the packet p. The operation $id(p)$ extracts the source and sequence number pair corresponding to the packet p. Such pairs comprise unique packet identifiers. We also define the $suffix(p)$ to be the subset of $P_{\text{RM-CLIENT}}$ comprised of all packets whose source is that of p and whose sequence number is greater than or equal to that of p.

4.1.2 The RM(Δ) Automaton

Figure 3 presents the signature, the variables, and the discrete transitions of RM(Δ). The RM(Δ) automaton maintains the set of members of the reliable multicast group. Hosts initiate the process of joining and leaving the reliable multicast group by issuing join and leave requests to the reliable multicast service. A request to join the reliable multicast group is effective only when the host is *idle* with respect to the reliable multicast group; that is, it is operational and neither a member of nor in the process of joining or leaving the reliable multicast group. A host becomes a member of the reliable multicast group upon the acknowledgment of an earlier join request. Hosts may only send and receive packets through the reliable multicast service while they are both operational and members of the reliable multicast group. Once a host issues a request to leave the reliable multicast group, it ceases to be a member of the reliable multicast group and, thus, relinquishes its right to receive any more reliable multicast packets. Leave requests overrule join requests in the sense that if the client is already in the process of joining the group while it issues a leave request, then the process of joining is aborted and the process of leaving is initiated. Once a host leaves the reliable multicast group, it may later rejoin the reliable multicast group by re-issuing a join request. Hosts may crash at any point in time. Once a host has crashed, the reliable multicast service ignores all events pertaining to the crashed host. Recall that host restarts are treated implicitly by thinking of host restarts as host reincarnations.

We say that a member h of the reliable multicast group has *delivered* the packet p if it has either sent or received the packet p. We say that a member h of the reliable multicast group is *aware of* a packet p, or is *expecting* p, if it has delivered either p or an earlier packet p' from the source of p. Moreover, we say that a packet p is *active* if at least one member of the reliable multicast group that has become aware of p since last joining the reliable multicast group, has also delivered it since last joining the reliable multicast group.

Once a host joins the reliable multicast group, the issue of catching up on any of the packets multicast earlier is orthogonal to the transmission of future packets using the reliable multicast service. Thus, once a host joins the reliable multicast group, the first packet it receives from a particular source dictates the set of packets that are guaranteed delivery to the given host. In particular, none of the earlier packets and any of the later packets that remain active after being sent are guaranteed delivery, provided the host remains a member of the reliable multicast group. The host may catch up on earlier packets from the given source through a separate service. For example, earlier packets may be requested directly from the source through a unicast communication channel. The rationale behind this modeling choice is that the recovery of a large number of earlier packets may strain the reliable multicast service and wastefully expose the recovery of earlier packets to all or a subset of the reliable multicast group.

If $\Delta = \infty$, then RM(Δ) guarantees that if a packet p remains active forever after its transmission then any member that becomes aware of p and remains a member of the reliable multicast group thereafter, delivers p. Equivalently, if two members become aware of a packet p, remain members forever thereafter, and one member delivers p, then the other member delivers p also. It is important to note that a host is not required to remain a member of the reliable multicast group indefinitely in order for the packets it multicasts to be received by hosts that become aware of them; the eventual reception of packets is guaranteed to all hosts that become aware of them provided the packets remain active forever after they are sent.

If $\Delta \in \mathbb{R}^{\geq 0}$, then RM($\Delta$) guarantees that if a packet remains active for Δ time units past its transmission, then it is delivered to all hosts that become aware of it within these Δ time units and, subsequently, remain members of the reliable multicast group for the remaining duration of these Δ time units elapse.

Parameters The RM automaton is parameterized by a time bound, $\Delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, which specifies the maximum delay in delivering each packet sent to the appropriate members of the reliable multicast group. The value ∞ corresponds to the case in which the reliable multicast service guarantees the eventual delivery of all packets to the appropriate members of the reliable multicast group. An instance of the RM automaton is denoted by RM(Δ).

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of RM. Each variable $status(h) \in Status$, for $h \in H$, denotes the status of the host h. Each of its valuations is described in the definition of the set *Status*. We say that the host h is *operational* if it has not crashed. After a host h crashes, none of the input actions pertaining to h affect the state of RM and none of the locally controlled actions pertaining to h are enabled.

Each variable $trans\text{-}time(p) \in \mathbb{R}^{\geq 0} \cup \bot$, for $p \in P_{\text{RM-CLIENT}}$, denotes the transmission time of the packet p; that is, the time the packet p was sent by its source. Prior to the transmission of p,

Figure 3 The RM(Δ) Automaton

Parameters:

$\Delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$

Actions:

Input:
- $\texttt{crash}_h$, for $h \in H$
- $\texttt{rm-join}_h$, for $h \in H$
- $\texttt{rm-leave}_h$, for $h \in H$
- $\texttt{rm-send}_h(p)$, for $h \in H, p \in P_{\text{RM-CLIENT}}$

Output:
- $\texttt{rm-join-ack}_h$, for $h \in H$
- $\texttt{rm-leave-ack}_h$, for $h \in H$
- $\texttt{rm-recv}_h(p)$, for $h \in H, p \in P_{\text{RM-CLIENT}}$

Time Passage:
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in Status$, for all $h \in H$, initially $status(h) = \texttt{idle}$, for all $h \in H$
$trans\text{-}time(p) \in \mathbb{R}^{\geq 0} \cup \perp$, for all $p \in P_{\text{RM-CLIENT}}$, initially $trans\text{-}time(p) =\perp$, for all $p \in P_{\text{RM-CLIENT}}$
$expected(h, h') \subseteq H \times \mathbb{N}$, for all $h, h' \in H$, initially $expected(h, h') = \emptyset$, for all $h, h' \in H$
$delivered(h, h') \subseteq H \times \mathbb{N}$, for all $h, h' \in H$, initially $delivered(h, h') = \emptyset$, for all $h, h' \in H$

Derived Variables:

$idle = \{h \in H \mid status(h) = \texttt{idle}\}$
$joining = \{h \in H \mid status(h) = \texttt{joining}\}$
$leaving = \{h \in H \mid status(h) = \texttt{leaving}\}$
$members = \{h \in H \mid status(h) = \texttt{member}\}$
$intended(p) = \{h \in H \mid id(p) \in expected(h, source(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$completed(p) = \{h \in H \mid id(p) \in delivered(h, source(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$sent\text{-}pkts = \{p \in P_{\text{RM-CLIENT}} \mid trans\text{-}time(p) \neq \perp\}$
$active\text{-}pkts = \{p \in P_{\text{RM-CLIENT}} \mid p \in sent\text{-}pkts \wedge intended(p) \cap completed(p) \neq \emptyset\}$

Discrete Transitions:

```
input crash_h
eff  status(h) := crashed
     foreach h' ∈ H do:
        expected(h, h') := ∅
        delivered(h, h') := ∅

input rm-join_h
eff  if h ∈ idle then
        status(h) := joining

input rm-leave_h
eff  if h ∈ joining ∪ members then
        status(h) := leaving
        foreach h' ∈ H do:
           expected(h, h') := ∅
           delivered(h, h') := ∅

input rm-send_h(p)
eff  if h ∈ members ∩ {source(p)} then
        if expected(h, h) = ∅ then
           expected(h, h) := suffix(p)
        if id(p) ∈ expected(h, h) then
           trans-time(p) := now
           delivered(h, h) ∪= {id(p)}

output rm-join-ack_h
pre  h ∈ joining
eff  status(h) := member

output rm-leave-ack_h
pre  h ∈ leaving
eff  status(h) := idle

output rm-recv_h(p)
pre  h ∈ members \ {source(p)}
     ∧p ∈ sent-pkts
     ∧(expected(h, source(p)) = ∅
        ⇒ now ≤ trans-time(p) + Δ)
     ∧(expected(h, source(p)) ≠ ∅
        ⇒ id(p) ∈ expected(h, source(p)))
eff  if expected(h, source(p)) = ∅ then
        expected(h, source(p)) := suffix(p)
     delivered(h, source(p)) ∪= {id(p)}

time-passage ν(t)
pre  ∀ p ∈ active-pkts,
        now + t ≤ trans-time(p) + Δ
        ∨intended(p) ⊆ completed(p)
eff  now := now + t
```

$trans\text{-}time(p)$ is equal to $\perp$. Each variable $expected(h, h') \subseteq H \times \mathbb{N}$, for $h, h' \in H$, is the set comprised of the identifiers of the packets from h' that the host h is aware of since it last joined the reliable multicast group and, consequently, expects to deliver. Each variable $delivered(h, h') \subseteq H \times \mathbb{N}$, for $h, h' \in H$, is the set comprised of the identifiers of the packets from h' that the host h has delivered.

Derived Variables The derived variable $idle \subseteq H$ is a set of hosts that is comprised of all the hosts that are idle with respect to the reliable multicast group. The derived variable $joining \subseteq H$ is a set of hosts that are in the process of joining the reliable multicast group. The derived variable $leaving \subseteq H$ is a set of hosts that are in the process of leaving the reliable multicast group. The derived variable $members \subseteq H$ is a set of hosts that are members of the reliable multicast group.

The derived variable *intended*(p), for each $p \in P_{\text{RM-CLIENT}}$, is the set of hosts that are expecting the delivery of the packet p. We henceforth refer to the set *intended*(p) as the intended delivery set of p. The derived variable *completed*(p), for each $p \in P_{\text{RM-CLIENT}}$, is the set of hosts that have delivered the packet p. Recall that we say that a host has delivered a packet p if it has either sent or received p. We henceforth refer to the set *completed*(p) as the completed delivery set of p. The derived variable *sent-pkts* is the set of packets that have been sent since the beginning of the given execution of the RM(Δ) automaton. The derived variable *active-pkts* is the set comprised of the sent packets that have been delivered by at least one of the hosts in their respective intended delivery sets.

Input Actions Each input action $\texttt{crash}_h$, for $h \in H$, models the crashing of the host h. The effects of $\texttt{crash}_h$ are to record that the host h has crashed by setting the variable *status*(h) to the value `crashed`. Furthermore, the $\texttt{crash}_h$ action resets the set of packets that the host h is expecting from each source and the set of packets it has delivered from each source. Thus, the RM automaton is released of the obligation to deliver any of the active packets to the host h.

The input action $\texttt{rm-join}_h$ models the client's request at the host h to join the reliable multicast group. The $\texttt{rm-join}_h$ action is effective only while the host h is idle with respect to the reliable multicast group. When effective, the $\texttt{rm-join}_h$ action sets the *status*(h) variable to `joining` so as to record that the host h has initiated the process of joining the reliable multicast group. If the client is either a member of or in the process of joining the reliable multicast group, then the $\texttt{rm-join}_h$ action is superfluous. If the client is already in the process of leaving the group, then the $\texttt{rm-join}_h$ action is discarded so as to allow the process of leaving the reliable multicast group to complete.

The input action $\texttt{rm-leave}_h$ models the client's request at the host h to leave the reliable multicast group. The $\texttt{rm-leave}_h$ action is effective only while the host h is a member of or in the process of joining the reliable multicast group. When effective, the $\texttt{rm-leave}_h$ action sets the *status*(h) variable to `leaving` so as to record that the host h has initiated the process of leaving the reliable multicast group. Moreover, the $\texttt{rm-leave}_h$ action initializes the set of packets that the host h is expecting from each source and the set of packets it has delivered from each source. Thus, the RM automaton is released of the obligation to deliver any of the active packets to the host h. Leave requests overrule join requests; that is, when a $\texttt{rm-leave}_h$ action is performed while the host h is in the process of joining the reliable multicast group, its effects are to abort the process of joining and to initiate the process of leaving the reliable multicast group. If the client is either idle or already in the process of leaving the reliable multicast group, then the $\texttt{rm-leave}_h$ action is superfluous.

The client at h sends the packet p using the reliable multicast service through the input action $\texttt{rm-send}_h(p)$. The $\texttt{rm-send}_h(p)$ action is effective only when the host h is both a member of the reliable multicast group and the source of the packet p. If p is the first packet sent by the host h, then the $\texttt{rm-send}_h(p)$ action initializes the set of packets expected by h from h to the set *suffix*(p); that is, all packets whose source is h and whose sequence number is greater or equal to that of p. Then, if p is in the expected set of packets of h from h, the $\texttt{rm-send}_h(p)$ records the transmission time of p by setting the variable *trans-time*(p) to *now* and adds the packet p to the set of packets from the host h that the host h has delivered.

Output Actions The output action $\texttt{rm-join-ack}_h$ acknowledges the join request of the client at h. The action $\texttt{rm-join-ack}_h$ is enabled when the host h is in the process of joining the reliable multicast group. Its effects are to set the *status*(h) variable to `member` so as to indicate that the client at h has become a member of the reliable multicast group.

The output action $\texttt{rm-leave-ack}_h$ acknowledges the leave request of the client at h. The action `rm-leave-ack`$_h$ is enabled when the host h is in the process of leaving the reliable multicast group. Its effects are to set the *status*(h) variable to `idle` so as to indicate that the client at h has become idle with respect to the reliable multicast group.

The output action `rm-recv`$_h(p)$ models the delivery of the packet p to the client at h. The `rm-recv`$_h(p)$ action is enabled when the host h is a member of the reliable multicast group, the host h is not the source of p, and p is an active packet. Moreover, if the expected deliver set of h with respect to the source of p is undefined, then the delivery deadline *trans-time*$(p) + \Delta$ of p must not have expired; that is, the first packet from any source to be delivered to any client must be delivered prior to its delivery deadline. If the expected deliver set of h with respect to the source of p has already been defined, then p must be expected by h. The effects of the `rm-recv`$_h(p)$ action are: i) to define the expected delivery set of h with respect to the source of p to the set *suffix*(p), unless already defined, and ii) to add the host h to the completed delivery set of p.

Time Passage The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing past the delivery deadline of any active packet that has yet to be delivered to all the hosts in its intended delivery set. Thus, prior to allowing time to elapse past the delivery deadline of an active packet, all the hosts in its intended delivery set must either send or receive the packet, leave the reliable multicast group, or crash.

4.1.3 The RM-Client$_h$ Automata

Figure 4 presents the signature, the variables, and the discrete transitions of RM-Client$_h$. The RM-Client$_h$ automaton models a *well-behaved* client; that is, a client that: i) transmits packets only when it is a member of the reliable multicast group, ii) transmits packets in ascending and contiguous sequence number order, iii) issues join requests only when it is idle with respect to the reliable multicast group, and iv) issues leave requests only when it is a member of the reliable multicast group.

Variables The variable *now* $\in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of RM-Client$_h$. The variable *status* $\in$ *Status* denotes the membership status of the host h. It takes on one of the following values: `idle`, `joining`, `leaving`, `member`, and `crashed`. These values indicate whether the host h either is idle, joining, leaving, a member of the reliable multicast group, or has crashed, respectively. We say that a host h is *operational* if it has not crashed. After a host h crashes, none of the input actions affect the state of RM-Client$_h$ and none of the locally controlled actions, except the time passage action, are enabled. The variable *seqno* $\in \mathbb{N} \cup \perp$ indicates the sequence number of the last packet to have been transmitted by RM-Client$_h$ — the value $\perp$ indicates that RM-Client$_h$ has yet to transmit a packet using the reliable multicast service. The *seqno* variable is initialized to $\perp$.

Input Actions The input action `crash`$_h$ models the crashing of the host h. The effects of `crash`$_h$ are to record that the host h has crashed by setting the *status* variable to `crashed`.

The input action `rm-join-ack`$_h$ acknowledges the client's join request at h. If the client is in the process of joining the reliable multicast group, *i.e.*, *status* $=$ `joining`, then the `rm-join-ack`$_h$ action sets the *status* variable to `member` so as to indicate that the client at h has become a member of the reliable multicast group.

The input action `rm-leave-ack`$_h$ acknowledges the client's leave request at h. If the client is in the process of leaving the reliable multicast group, *i.e.*, *status* $=$ `leaving`, then the `rm-leave-ack`$_h$

Figure 4 The RM-CLIENT$_h$ Automaton

Parameters:

$h \in H$

Actions:

Input:
`crash`$_h$
`rm-join-ack`$_h$
`rm-leave-ack`$_h$
`rm-recv`$_h(p)$, for all $p \in P_{\text{RM-CLIENT}}$

Output:
`rm-join`$_h$
`rm-leave`$_h$
`rm-send`$_h(p)$, for all $p \in P_{\text{RM-CLIENT}}$

Time Passage:
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in Status$, initially $status =$ `idle`
$seqno \in \mathbb{N} \cup \perp$, initially $seqno = \perp$

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status =$ `joining` **then**
$status :=$ `member`

input `rm-leave-ack`$_h$
eff **if** $status =$ `leaving` **then**
$status :=$ `idle`

input `rm-recv`$_h(p)$
eff None

output `rm-join`$_h$
pre $status =$ `idle`
eff $status :=$ `joining`

output `rm-leave`$_h$
pre $status =$ `member`
eff $status :=$ `leaving`

output `rm-send`$_h(p)$
pre $status =$ `member` $\wedge\, source(p) = h$
$\wedge(seqno = \perp \vee seqno(p) = seqno + 1)$
eff $seqno := seqno(p)$

time-passage $\nu(t)$
pre None
eff $now := now + t$

action sets the *status* variable to `idle` so as to indicate that the client at h has become idle with respect to the reliable multicast group.

The input action `rm-recv`$_h(p)$ models the delivery of the packet p to the client at h. This action has no effects.

Output Actions The output action `rm-join`$_h$ is performed by the client to initiate the process of joining the reliable multicast group. This action is enabled only while the client is idle with respect to the reliable multicast group. Its effects are to set the *status* variable to `joining` so as to indicate that the client at h has initiated the process of joining the reliable multicast group.

The output action `rm-leave`$_h$ is performed by the client so as to initiate the process of leaving the reliable multicast group. This action is enabled only while the client is a member of the reliable multicast group. Thus, the client waits for join requests to complete prior to issuing leave requests. Its effects are to set the *status* variable to `leaving` so as to indicate that the client at h has initiated the process of leaving the reliable multicast group.

The output action `rm-send`$_h(p)$ models the client's transmission of the packet p using the reliable multicast service. The `rm-send`$_h(p)$ action is enabled when the client is a member of the reliable multicast group and the packet p is either the first or the next packet in the sequence of packets to be transmitted by the client at h; that is, $status =$ `member`, $source(p) = h$, and either $seqno = \perp$ or $seqno(p) = seqno + 1$. The effects of the `rm-send`$_h(p)$ action are to set $seqno$ to $seqno(p)$ (or, equivalently, to increment $seqno$), thus recording the transmission of the packet p.

Time Passage The action $\nu(t)$ models the passage of t time units. It is enabled at any point in time and increments the variable now by t time units.

4.2 Preliminary Properties and Definitions

The automaton RM-CLIENT$_h$, for any $h \in H$, satisfies *transmission correctness*, *transmission uniqueness*, and *in order transmission*. Transmission correctness is the property that clients only transmit packets for which they are actually the source. Transmission uniqueness is the property that no two packets transmitted by a client share the same identifier. Finally, in order transmission is the property that each client transmits packets through the reliable multicast group in ascending sequence number order.

Lemma 4.1 (Transmission Correctness) *Let β be any timed trace of* RM-CLIENT$_h$*, for any $h \in H$. If β contains the action* $\texttt{rm-send}_h(p)$*, for some $p \in P_{\text{RM-CLIENT}}$, then the host h is the source of p; that is, $h = source(p)$.*

Proof: Follows directly from the precondition of the action $\texttt{rm-send}_h(p)$. ■

Lemma 4.2 (Transmission Uniqueness) *Let β be any timed trace of* RM-CLIENT$_h$*, for any $h \in H$. For any packet identifier $\langle s, i\rangle \in H \times \mathbb{N}$, at most one packet $p \in P_{\text{RM-CLIENT}}$ is transmitted within β; that is, β contains at most one action* $\texttt{rm-send}_h(p)$*, for $p \in P_{\text{RM-CLIENT}}$, such that $id(p) = \langle s, i\rangle$.*

Proof: Let α be any timed execution of RM-CLIENT$_h$ such that $\beta = ttrace(\alpha)$. Within α each action $\texttt{rm-send}_h(p')$, for $p' \in P_{\text{RM-CLIENT}}$ such that $source(p') = h$, transmits the packet p' whose sequence number is equal to *seqno* and increments the variable *seqno*. Since no other actions affect the variable *seqno* it follows that *seqno* monotonically increases each time a packet is transmitted. Thus, β does not contain the transmission of more than one packets sharing the same sequence number. ■

Lemma 4.3 (In Order Transmission) *Let β be any timed trace of* RM-CLIENT$_h$*, for $h \in H$, that contains the actions* $\texttt{rm-send}_h(p)$ *and* $\texttt{rm-send}_h(p')$*, for $p, p' \in P_{\text{RM-CLIENT}}$, such that $h = source(p) = source(p')$ and $seqno(p) < seqno(p')$. Then, the action* $\texttt{rm-send}_h(p)$ *precedes the action* $\texttt{rm-send}_h(p')$ *in β.*

Proof: The effects of any $\texttt{rm-send}_h(p'')$, for $p'' \in P_{\text{RM-CLIENT}}$, are to increment the variable RM-CLIENT$_h$.*seqno*. Moreover, no other action affects the variable RM-CLIENT$_h$.*seqno*. Thus is, the variable RM-CLIENT$_h$.*seqno* is monotonically non-decreasing in any execution of RM-CLIENT$_h$. The actions $\texttt{rm-send}_h(p)$ and $\texttt{rm-send}_h(p')$ are enabled only when $seqno(p) =$ RM-CLIENT$_h$.*seqno* and $seqno(p') =$ RM-CLIENT$_h$.*seqno*, respectively. It follows that $\texttt{rm-send}_h(p)$ precedes the action $\texttt{rm-send}_h(p')$ in any timed execution of RM-CLIENT$_h$ such that $\beta = ttrace(\alpha)$. ■

The automaton RM$_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$ satisfies *transmission integrity*. Transmission integrity it the property that, within a timed trace of RM$_S(\Delta)$, the reception of a packet must be preceded by the particular packet's transmission.

Lemma 4.4 (Transmission Integrity) *Let β be any timed trace of* RM$_S(\Delta)$*, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. For $h, h' \in H$ and $p \in P_{\text{RM-CLIENT}}$, such that $h \neq h'$ and $h = source(p)$, it is the case that any* $\texttt{rm-recv}_{h'}(p)$ *action is preceded in β by a* $\texttt{rm-send}_h(p)$ *action.*

Proof: Let α be any timed execution of RM$_S(\Delta)$ such that $\beta = ttrace(\alpha)$. It suffices to show that any $\texttt{rm-recv}_{h'}(p)$ action is preceded by a $\texttt{rm-send}_h(p)$ action within α. This follows directly from the precondition of the action $\texttt{rm-recv}_{h'}(p)$. In particular, the precondition of the action $\texttt{rm-recv}_{h'}(p)$ requires that there is a tuple in *pkts* corresponding to the packet p. However, such a tuple may be added to *pkts* only by the occurrence of the action $\texttt{rm-send}_h(p)$. Thus, the occurrence of any action $\texttt{rm-recv}_{h'}(p)$ within α is preceded by the occurrence of the action $\texttt{rm-send}_h(p)$. ■

We proceed by defining the set of *members* of the reliable multicast group following a finite timed trace of $\mathrm{RM}_S(\Delta)$.

Definition 4.1 (Membership) *Let β be any timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. We define the **members of** β, denoted members(β), to be the set of all hosts $h \in H$ such that β contains a $\texttt{rm-join-ack}_h$ action that is not succeeded by either an $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action. If a host $h \in H$ is in the set members(β), then we say that h is a reliable multicast group* member *of β.*

The following lemma relates the set *members*(β) of Definition 4.1 to the derived variable *members* of the automaton RM.

Lemma 4.5 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$ and α be any finite timed execution of $\mathrm{RM}_S(\Delta)$. Letting s be the last state in α and β be the timed trace of α, it is the case that $s.members = members(\beta)$.*

Proof: Follows directly from the definitions of $s.members$ and $members(\beta)$. ■

Lemma 4.6 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $h \in H$, and α be any timed execution of $\mathrm{RM}_S(\Delta)$ such that $h \in members(ttrace(\alpha))$. Letting s be any state following the last occurrence of the $\texttt{rm-join-ack}_h$ action in α, it is the case that $h \in s.members$.*

Proof: Let α', α'' be the execution fragments of $\mathrm{RM}_S(\Delta)$ such that $\alpha'\alpha'' = \alpha$ and the last action in α' is the last occurrence of the $\texttt{rm-join-ack}_h$ action in α. Letting $s' = \alpha'.lstate$, the effects of the $\texttt{rm-join-ack}_h$ action imply that $s'.status(h) = \texttt{member}$. By the definition of $members(ttrace(\alpha))$, it follows that α'' contains neither a $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action.

The rest of the proof involves showing that for any prefix α_n of α'' of length $n \in \mathbb{N}$, such that $s_n = \alpha_n.lstate$, it is the case that $h \in s_n.members$. This follows by a simple induction on the length n of α_n. For the base case, consider α_0. Since $\alpha_0 = s'$ and $s'.status(h) = \texttt{member}$, it follows that $s_0.status(h) = \texttt{member}$, as required. For the inductive step, consider α_{k+1}. Let $s_{k+1} = \alpha_{k+1}.lstate$, let α_k be the prefix of α_{k+1} involving its first k steps, and $s_k = \alpha_k.lstate$. The induction hypothesis is the assertion that $s_k.status(h) = \texttt{member}$. Since α'' contains neither a $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action, the $k+1$-st step of α_{k+1} is neither an $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action. Moreover, since $s_k.status(h) = \texttt{member}$, the $k+1$-st step of α_{k+1} is neither an $\texttt{rm-join}_h$, $\texttt{rm-join-ack}_h$, nor $\texttt{rm-leave-ack}_h$ action. The remaining actions do not affect the $status(h)$ variable. Thus, it follows that $s_{k+1}.status(h) = \texttt{member}$, as required. ■

We proceed by defining the *intended and completed delivery* sets of a packet within a timed trace of $\mathrm{RM}_S(\Delta)$.

Definition 4.2 (Intended Delivery Set) *Let β be any timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. We define the **intended delivery set of** p **within** β, denoted intended(p, β), to be the members of β that have delivered either the packet p or an earlier packet from the source of p since they last joined the reliable multicast group; that is, $h \in intended(p, \beta)$ if and only if $h \in members(\beta)$ and the last $\texttt{rm-join-ack}_h$ action in β is succeeded by either a rm-send$_h(p')$ or a rm-recv$_h(p')$ action, where $source(p') = source(p)$ and $seqno(p') \leq seqno(p)$.*

Lemma 4.7 *Let β be any finite timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. Then, it is the case that $intended(p, \beta) \subseteq members(\beta)$.*

Proof: Follows directly from Definition 4.2. ∎

The following lemma relates the intended delivery set of a packet p within a timed trace β defined in Definition 4.2 to the derived variable *intended*(p) of the RM automaton.

Lemma 4.8 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $p \in P_{\text{RM-CLIENT}}$, and α be any finite timed execution of $\mathrm{RM}_S(\Delta)$ that contains the transmission of p. Letting $s = \alpha.lstate$ and $\beta = ttrace(\alpha)$, it is the case that $s.intended(p) = intended(p, \beta)$.*

Proof: Follows directly from the definition of the derived variable *intended*(p) and Definition 4.2. ∎

Definition 4.3 (Completed Delivery Set) *Let β be any timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. We define the* ***completed delivery set of*** *p* ***within*** *β, denoted $completed(p, \beta)$, to be the members of β that have delivered the packet p since they last joined the reliable multicast group; that is, $h \in completed(p, \beta)$ if and only if $h \in members(\beta)$ and the last* `rm-join-ack`$_h$ *action in β is succeeded by either a rm-send$_h(p)$ or a rm-recv$_h(p)$ action.*

The following lemma relates the completed delivery set of a packet p within a timed trace β defined in Definition 4.3 to the derived variable *completed*(p) of the RM automaton.

Lemma 4.9 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $p \in P_{\text{RM-CLIENT}}$, and α be any finite timed execution of $\mathrm{RM}(\Delta) \times$ RMCLIENTS that contains the transmission of p. Letting $s = \alpha.lstate$ and $\beta = ttrace(\alpha)$, it is the case that $s.completed(p) = completed(p, \beta)$.*

Proof: Follows directly from the definition of the derived variable *completed*(p) and Definition 4.3. ∎

We continue by defining the set of active packets within a timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. This set is comprised of the packets whose intended and completed delivery sets within the given timed trace overlap; that is, the packets for which there is at least one host that was and has remained a member of the reliable multicast group following the packet's transmission and, moreover, has either sent or received the packet.

Definition 4.4 (Active Packets) *Let β be any timed trace of $\mathrm{RM}(\Delta) \times$ RMCLIENTS, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. We define the set of* ***active packets within*** *β, denoted active-pkts(β), to be the set of all packets $p \in P_{\text{RM-CLIENT}}$ such that $intended(p, \beta) \cap completed(p, \beta) \neq \emptyset$. If a packet $p \in P_{\text{RM-CLIENT}}$ is in the set active-pkts(β), then we say that p is active within β.*

The following lemma relates the set of active packets defined in Definition 4.4 to the derived variable *active-pkts* of the RM automaton.

Lemma 4.10 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $p \in P_{\text{RM-CLIENT}}$, and α be any finite timed execution of $\mathrm{RM}(\Delta) \times$ RMCLIENTS that contains the transmission of p. Letting $s = \alpha.lstate$ and $\beta = ttrace(\alpha)$, it is the case that $s.active\text{-}pkts = active\text{-}pkts(\beta)$.*

Proof: Follows directly from Lemmas 4.8 and 4.9, Definition 4.4, and the definition of the derived variable *active-pkts* of the RM automaton. ■

Lemma 4.11 *Let β, β' be timed traces of* $\mathrm{RM}(\Delta) \times \textsc{rmClients}$*, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\textsc{RM-Client}}$ such that $\beta' \leq \beta$. Then, it is the case that if $p \in \textit{active-pkts}(\beta)$ then $p \in \textit{active-pkts}(\beta')$.*

Proof: We prove the above claim by contradiction. Suppose that it is the case that $p \notin \textit{active-pkts}(\beta')$ and $p \in \textit{active-pkts}(\beta)$. Thus, there must be some action π following β' such that $p \notin \textit{active-pkts}(\beta_\pi)$ and $p \in \textit{active-pkts}(\beta_\pi \cdot \pi)$, where β_π, β'_π are the trace fragments of β such that $\beta_\pi \cdot \pi \cdot \beta'_\pi = \beta$.

Let α be any timed execution of $\mathrm{RM}(\Delta) \times \textsc{rmClients}$ such that $\beta = \textit{ttrace}(\alpha)$ and s_π and s'_π be the pre- and post-states of π within α. We proceed by considering the possibility of π being any of the actions of the $\mathrm{RM}_S(\Delta)$ automaton that affect the valuation of the derived variable *active-pkts*. Since $p \notin \textit{active-pkts}(\beta_\pi)$, Lemma 4.10 implies that $p \notin s_\pi.\textit{active-pkts}$. Thus, none of the $\texttt{rm-recv}_h(p)$, for $h \in H$, are enabled. Lemma 4.1 implies that none of the actions $\texttt{rm-send}_h(p)$, for $h \in H$, except for $h = \textit{source}(p)$ are enabled. Moreover, since p has already been sent within β_π, Lemma 4.2 implies that $\texttt{rm-send}_h(p)$, for $h = \textit{source}(p)$, is not enabled in s_π. The only other actions that affect the variable *active-pkts* are the $\texttt{crash}_h$ and $\texttt{rm-leave}_h$ actions, for $h \in H$. The effects of these actions are to remove the host h from both the $\textit{intended}(p)$ and $\textit{completed}(p)$ sets. Clearly, if $\textit{intended}(p) \cap \textit{completed}(p) = \emptyset$ in the state s_π, then the same holds for s'_π. Thus, it follows that $p \notin s'_\pi.\textit{active-pkts}$. Lemma 4.10 implies that $p \notin \textit{active-pkts}(\beta_\pi \cdot \pi)$, which contradicts our original supposition. ■

Lemma 4.12 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $h \in H$, $p \in P_{\textsc{RM-Client}}$, and α be any timed execution of $\mathrm{RM}_S(\Delta)$ that ends with the discrete transition (s, π, s'), for $\pi = \textit{rm-send}_h(p)$. Then, it is the case that $p \in s'.\textit{sent-pkts}$.*

Proof: From the precondition of $\textit{rm-send}_h(p)$, it follows that $s.\textit{status} = \texttt{member}$ and $\textit{source}(p) = h$. Thus, the effects of the $\textit{rm-send}_h(p)$ are to set the variable $\textit{trans-time}(p)$ to the value of *now*. By the definition of the derived variable *sent-pkts* of $\mathrm{RM}(\Delta)$, it follows that $p \in s'.\textit{sent-pkts}$, as required. ■

Lemma 4.13 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $p \in P_{\textsc{RM-Client}}$, $s \in \textit{states}(\mathrm{RM}(\Delta))$ be any reachable state of $\mathrm{RM}(\Delta)$ such that $p \in s.\textit{sent-pkts}$, and α be any timed execution fragment of $\mathrm{RM}(\Delta)$ such that $s = \alpha.\textit{fstate}$. For any $s' \in \textit{states}(\mathrm{RM}(\Delta))$ in α, it is the case that $p \in s'.\textit{sent-pkts}$.*

Proof: Follows from a simple induction on the length of the prefix of α leading to s' and the fact that none of the actions of $\mathrm{RM}(\Delta)$ reset the variable $\textit{trans-time}(p)$ to $\perp$. ■

Lemma 4.14 *Let $h \in H$, $p \in P_{\textsc{RM-Client}}$, $s \in \textit{states}(\mathrm{RM}(\Delta))$, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, and α be any timed execution fragment of $\mathrm{RM}(\Delta)$, such that $s = \alpha.\textit{fstate}$, $h \in s.\textit{intended}(p)$ (or, equivalently, $\textit{id}(p) \in s.\textit{expected}(h, \textit{source}(p))$), and α contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Then, for any state $s' \in \textit{states}(\mathrm{RM}(\Delta))$ in α, it is the case that $h \in s'.\textit{intended}(p)$ (or, equivalently, $\textit{id}(p) \in s'.\textit{expected}(h, \textit{source}(p))$).*

Proof: Follows from a simple induction on the length of the prefix of α leading to s' and the facts that: i) the variable $expected(h, source(p))$ may only be set to a non-empty set if it is empty, and ii) the variable $expected(h, source(p))$ is reset to the empty set only by the actions $\texttt{crash}_h$ and $\texttt{rm-leave}_h$. ■

Invariant 4.1 *For $h \in H$ and any reachable state s of* $\mathrm{RM}(\Delta) \times \textsc{rmClients}$*, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, it is the case that* $s[\textsc{RM-Client}_h].status = s[\mathrm{RM}(\Delta)].status(h)$.

Proof: Follows by a simple induction on the length of any timed execution of $\mathrm{RM}_S(\Delta)$ leading to s. ■

Invariant 4.2 *Let $h, h' \in H$ and s be any reachable state of* $\mathrm{RM}_S(\Delta)$*, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. If* $s[\mathrm{RM}(\Delta)].status(h) \neq \texttt{member}$*, then it is the case that* $s[\mathrm{RM}(\Delta)].expected(h, h') = \emptyset$ *and* $s[\mathrm{RM}(\Delta)].delivered(h, h') = \emptyset$.

Proof: Follows from a simple induction on the length of any execution of $\mathrm{RM}_S(\Delta)$ leading to s and the facts that: i) the actions that set the variable $\mathrm{RM}(\Delta).expected(h, h')$ are only enabled when $\mathrm{RM}(\Delta).status(h) = \texttt{member}$, ii) the actions that add elements to the variable $\mathrm{RM}(\Delta).delivered(h, h')$ are only enabled when $\mathrm{RM}(\Delta).status(h) = \texttt{member}$, and iii) the actions that reset the variables $\mathrm{RM}(\Delta).expected(h, h')$ and $\mathrm{RM}(\Delta).delivered(h, h')$ also set the variable $\mathrm{RM}(\Delta).status(h)$ to a value other than $\texttt{member}$. ■

Letting $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, the following invariant states that, for any active packet in any reachable state of $\mathrm{RM}(\Delta) \times \textsc{rmClients}$, either Δ time units have yet to elapse past the packet's transmission time, or the packet has been delivered to all members that are aware of it. Thus, Δ bounds the delivery latency of any active packet.

Invariant 4.3 *Let s be any reachable state of the timed automaton* $\mathrm{RM}_S(\Delta)$*, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. Then, for any active packet $p \in P_{\textsc{RM-Client}}$ in s, i.e., $p \in s.active\text{-}pkts$, it is the case that either $s.now \leq s.trans\text{-}time(p) + \Delta$ or $s.intended(p) \subseteq s.completed(p)$.*

Proof: The proof is by induction of the number of steps $n \in \mathbb{N}$ of a timed execution α of $\mathrm{RM}_S(\Delta)$ leading to the state s. For the base case, consider a timed execution with no steps; that is, $n = 0$ and $\alpha = s$ for some $s \in start(\mathrm{RM}_S(\Delta))$. Since $s.active\text{-}pkts = \emptyset$, the invariant assertion is trivially satisfied.

For the inductive step, consider a timed execution α with $k + 1$ steps. Let α' be the prefix of α containing the first k steps of α and s' be the last state of α'. The induction hypothesis is that for any active packet $p' \in P_{\textsc{RM-Client}}$ in s', *i.e.*, $p' \in s'.active\text{-}pkts$, it is the case that either $s'.now \leq s'.trans\text{-}time(p') + \Delta$ or $s'.intended(p') \subseteq s'.completed(p')$. For the inductive step, we show that for any active packet $p \in P_{\textsc{RM-Client}}$ in s, *i.e.*, $p \in s.active\text{-}pkts$, it is the case that either $s.now \leq s.trans\text{-}tims(p) + \Delta$ or $s.intended(p) \subseteq s.completed(p)$.

Suppose that $p \in s.active\text{-}pkts$ and consider two cases depending on whether $p \in s'.active\text{-}pkts$. First, consider the case in which $p \notin s'.active\text{-}pkts$. Lemma 4.11 implies that the step from s' to s involves the action $\texttt{rm-send}_h(p)$, for $h = source(p)$. Its effects are to set the variable $trans\text{-}time(p)$ to now. It follows that $s.now \leq s.trans\text{-}time(p) + \Delta$. Thus, the invariant assertion is satisfied in s.

Second, consider the case in which $p \in s'.active\text{-}pkts$. Then, the induction hypothesis implies that either $s'.now \leq s'.trans\text{-}time(p) + \Delta$ or $s'.intended(p) \subseteq s'.completed(p)$. We proceed by considering the effects of each of the actions that affect any of the variables present in the invariant assertion:

- ❒ $\texttt{crash}_h$, for $h \in H$: the effects of this action are to remove the host h from the intended and completed delivery sets of p. Thus, the induction hypothesis implies that either $s.now \leq s.trans\text{-}time(p) + \Delta$ or $s.intended(p) \subseteq s.completed(p)$.
- ❒ $\texttt{rm-leave}_h$, for $h \in H$: the reasoning for this action is similar to that of the $\texttt{crash}_h$ action.
- ❒ $\texttt{rm-send}_h(p)$, for $h = source(p)$: since $p \in s'.active\text{-}pkts$ it follows that p has been sent prior to state s' within α. Thus, Lemma 4.2 implies that the $\texttt{rm-send}_h(p)$ action is not enabled in s'.
- ❒ $\texttt{rm-recv}_h(p)$, for $h \in H$: we consider two cases depending on whether $s'.expected(h, source(p))$ is empty. First, if $s'.expected(h, source(p)) = \emptyset$, the precondition of $\texttt{rm-recv}_h(p)$ implies that $s'.now \leq s'.trans\text{-}time(p) + \Delta$. Since the $\texttt{rm-recv}_h(p)$ action affects neither the now nor the $trans\text{-}time(p)$ variables, it follows that $s.now \leq s.trans\text{-}time(p) + \Delta$. Thus, the invariant assertion is satisfied in s. Second, if $s'.expected(h, source(p)) \neq \emptyset$, the precondition of $\texttt{rm-recv}_h(p)$ implies that $id(p) \in s'.expected(h, source(p))$. The effects of $\texttt{rm-recv}_h(p)$ are to add the element $id(p)$ to the set $delivered(h, source(p))$. Thus, the induction hypothesis implies that either $s.now \leq s.trans\text{-}time(p) + \Delta$ or $s.intended(p) \subseteq s.completed(p)$.
- ❒ $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$: the effects of the time-passage action are to allow t time units to elapse. However, the precondition of the action $\nu(t)$ implies that the invariant assertion is satisfied in s.

■

4.3 Reliability Properties

The $\text{RM}_S(\Delta)$ automaton, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, satisfies the *eventual delivery* and, equivalently, *pairwise eventual delivery*, properties. Eventual delivery is the property that if a host h is a member of the reliable multicast group, becomes aware of a packet p, remains a member of the group thereafter, and p remains active thereafter, then h delivers p since last joining the reliable multicast group. Its pairwise counterpart is the property that if two hosts are members of the reliable multicast group, become aware of the packet p, remain members of the group thereafter, and one of them delivers p since last joining the reliable multicast group, then so does the other. The eventual and pairwise eventual delivery properties are equivalent.

Theorem 4.15 (Eventual Delivery) *Let β be any fair admissible timed trace of $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. If $p \in active\text{-}pkts(\beta)$, then p is delivered by each host in the intended delivery set of p within β since each such host last joined the reliable multicast group; that is, $intended(p, \beta) \subseteq completed(p, \beta)$.*

Proof: Let α be any fair admissible timed execution of $\text{RM}_S(\Delta)$, such that $\beta = ttrace(\alpha)$. Suppose that $p \in active\text{-}pkts(\beta)$ and let $h \in intended(p, \beta)$. It suffices to show that $h \in completed(p, \beta)$.

First, we consider the case where h is the source of p. Since $h \in intended(p, \beta)$, Definition 4.2 implies that the last $\texttt{rm-join-ack}_h$ action in β is succeeded by a $rm\text{-}send_h(p')$ action, where $source(p') = source(p)$ and $seqno(p') \leq seqno(p)$. If $seqno(p') = seqno(p)$ and, consequently, $p' = p$, then it is the case that the last $\texttt{rm-join-ack}_h$ action in β is succeeded by a $rm\text{-}send_h(p)$ action. By Definition 4.3, it follows that $h \in completed(p, \beta)$, as needed. If $seqno(p') < seqno(p)$, then Lemma 4.3 implies that the transmission of p in β succeeds the transmission of p' in β. Since the $rm\text{-}send_h(p')$ action succeeds the last $\texttt{rm-join-ack}_h$ action in β, so does the $rm\text{-}send_h(p)$ action. By Definition 4.3, it follows that $h \in completed(p, \beta)$, as needed.

Second, consider the case where h is not the source of p. Since $h \in intended(p, \beta)$, Definition 4.2 implies that the last $\texttt{rm-join-ack}_h$ action in β is succeeded by a $rm\text{-}recv_h(p')$ action, where $source(p') = source(p)$ and $seqno(p') \leq seqno(p)$. If $seqno(p') = seqno(p)$ and, consequently,

$p' = p$, then it is the case that the last $\texttt{rm-join-ack}_h$ action in β is succeeded by a $\mathit{rm\text{-}recv}_h(p)$ action. By Definition 4.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed.

Now, consider the case where $\mathit{seqno}(p') < \mathit{seqno}(p)$. Let (s'_-, π, s'_+) be the discrete transition in α corresponding to the particular occurrence of the $\mathit{rm\text{-}recv}_h(p')$ action in β and α' be the suffix of α that starts in the post-state s'_+ of (s'_-, π, s'_+). Moreover, let $s_{\alpha'}$ be any state in α'. Since $h \in \mathit{intended}(p, \beta)$, Lemma 4.7 implies that $h \in \mathit{members}(\beta)$. Since α' succeeds the last $\texttt{rm-join-ack}_h$ action in α, Lemma 4.6 implies that $h \in s_{\alpha'}.\mathit{members}$. Since $h \neq \mathit{source}(p)$, it follows that $h \in s_{\alpha'}.\mathit{members} \backslash \{\mathit{source}(p)\}$. The precondition and the effects of the $\mathit{rm\text{-}recv}_h(p')$ action imply that $\mathit{id}(p) \in s'_+.\mathit{expected}(h, \mathit{source}(p))$. Moreover, Lemma 4.14 implies that $\mathit{id}(p) \in s_{\alpha'}.\mathit{expected}(h, \mathit{source}(p))$.

Moreover, let (s''_-, π, s''_+) be the discrete transition in α corresponding to the occurrence of the $\mathit{rm\text{-}send}_{h'}(p)$ action in β, for $h' = \mathit{source}(p)$, and α'' be the suffix of α that starts in the post-state s''_+ of (s''_-, π, s''_+). Moreover, let $s_{\alpha''}$ be any state in α''. Lemma 4.12 implies that $p \in s''_+.\mathit{sent\text{-}pkts}$ and Lemma 4.13 implies that $p \in s_{\alpha''}.\mathit{sent\text{-}pkts}$.

Now, let α^* be any timed execution fragment that is a common suffix of α' and α'' and let s^* be any state in α^*. Since $h \in s_{\alpha'}.\mathit{members} \backslash \{\mathit{source}(p)\}$, $p \in s_{\alpha''}.\mathit{sent\text{-}pkts}$, and $\mathit{id}(p) \in s_{\alpha'}.\mathit{expected}(h, \mathit{source}(p))$, it is the case that $h \in s^*.\mathit{members} \backslash \{\mathit{source}(p)\}$, $p \in s^*.\mathit{sent\text{-}pkts}$, and $\mathit{id}(p) \in s^*.\mathit{expected}(h, \mathit{source}(p))$. Thus, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in s^*; that is, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in any state in α^*.

Since α^* is a suffix of α and α is an admissible timed execution of $\mathrm{RM}_S(\Delta)$, it is the case that α^* is infinite. Since the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in any state of α^*, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled infinitely often in α^*. Since α is fair, the $\texttt{rm-recv}_h(p)$ action occurs in α^*. Thus, the $\texttt{rm-recv}_h(p)$ action succeeds the last $\texttt{rm-join-ack}_h$ action in α. By Definition 4.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed. ■

The following theorem defines the *pairwise eventual delivery* property of $\mathrm{RM}_S(\Delta)$. It states that if two hosts are members of the reliable multicast group, become aware of the packet p, remain members of the group thereafter, and one of them delivers p, then so does the other. The pairwise eventual delivery is equivalent to the eventual delivery property defined in Theorem 4.15.

Corollary 4.16 (Pairwise Eventual Delivery) *Let β be any fair admissible timed trace of the $\mathrm{RM}_S(\Delta)$ automaton, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$ and the hosts $h, h' \in H, h \neq h'$ be any two distinct hosts in the intended delivery set of p within β. Then, if h delivers p within β, then so does h'.*

Proof: Since h is in the intended delivery set of p within β and it delivers p within β, it follows that p is active within β; that is, $p \in \mathit{active\text{-}pkts}(\beta)$. Since h' is in the intended delivery set of p within β, Theorem 4.15 implies that h' delivers p within β. ■

The following theorem defines the notion of *time-bounded delivery*; that is, the property that any packet that remains active for at least $\Delta \in \mathbb{R}^{\geq 0}$ time units past its transmission is delivered within these Δ time units to all hosts that become aware of it within these Δ time units.

Theorem 4.17 (Time-Bounded Delivery) *Let β be any admissible timed trace of $\mathrm{RM}(\Delta) \times$ RMCLIENTS, for any $\Delta \in \mathbb{R}^{\geq 0}$, that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. Let β' be the finite prefix of β ending with the transmission of p; that is, the last action contained in β' is the action $\texttt{rm-send}_h(p)$, for $h \in H, h = \mathit{source}(p)$. Let β'' be any finite prefix of β, such that $\beta' \leq \beta'' \leq \beta$ and $t' + \Delta < t''$, with $t', t'' \in \mathbb{R}^{\geq 0}$ being the time of occurrence of the last actions of β' and β'', respectively. Suppose that the host h' is in the intended delivery set of p within β''*

and that the packet p is active within β''. Then, the host h delivers the packet p within β''; that is, $h' \in completed(p, \beta'')$.

Proof: Let α be any admissible execution of $\mathrm{RM}(\Delta) \times$ rmClients such that $\beta = ttrace(\alpha)$. Moreover, let α' and α'' be finite prefixes of α such that $\alpha' \leq \alpha'' \leq \alpha$, $\beta' = ttrace(\alpha')$, $\beta'' = ttrace(\alpha'')$, and the last actions in α' and α'' are the last actions in β' and β'', respectively. Finally, let s' and s'' be the last states of α' and α'', respectively.

Since $t' + \Delta < t''$, it follows that $s''.trans\text{-}time(p) + \Delta < s''.now$. Since $p \in active\text{-}pkts(\beta'')$, Lemma 4.10 implies that $p \in s''.active\text{-}pkts$. Since $p \in s''.active\text{-}pkts$ and $s''.trans\text{-}time(p) + \Delta < s''.now$, Invariant 4.3 implies that $s''.intended(p) \subseteq s''.completed(p)$. Lemmas 4.8 and 4.9, imply that $intended(p, \beta'') \subseteq completed(p, \beta'')$. Finally, since $h' \in intended(p, \beta'')$, it follows that $h' \in completed(p, \beta'')$; that is, the host h' delivers the packet p within β''. ■

5 Reliable Multicast Implementation (RMI)

In this section, we present RMI — a formal model of the Scalable Reliable Multicast (SRM) protocol [1]. RMI precisely specifies the behavior of the basic version of SRM — more sophisticated versions involve adaptive and local recovery schemes [1, 5].

5.1 Overview of RMI's Functionality

RMI consists of two distinct functional components: i) *packet loss recovery*, and ii) *session message exchange*. We proceed by describing each of these components.

Packet Loss Recovery Receivers detect packet losses by identifying sequence number gaps in the stream of packets received from each source. Upon detecting the loss of a packet p, a host h initiates a new recovery round for p by scheduling a retransmission *request* for p. This request is scheduled for transmission at a point in time in the future that is uniformly chosen within the interval $[C_1\hat{d}_{hs}, (C_1 + C_2)\hat{d}_{hs}]$, where $C_1, C_2 \in \mathbb{R}^{\geq 0}$ are request scheduling parameters and $\hat{d}_{hs}$ is half of h's round-trip-time (RTT) estimate to the source s of the packet p.

Upon either the transmission of a request for p or the reception of a request for p while a request for p is pending transmission, the host h initiates a new recovery round for p by rescheduling the request for p for transmission at a point in time in the future that is uniformly chosen within the interval $2^{k-1}[C_1\hat{d}_{hs}, (C_1 + C_2)\hat{d}_{hs}]$, where $k \in \mathbb{N}^+$ is the number of recovery rounds for p that h has already initiated. In effect, the request for p is rescheduled by performing an exponential back-off. If h receives p while a request for p is pending transmission, then the request for p is canceled.

Once h reschedules its request for p, it observes a *back-off abstinence period*. During this period, it refrains from backing-off its request for p. Any requests for p received during this period are considered to pertain to prior recovery rounds and are discarded. Thus, back-off abstinence periods prevent requests from being backed-off multiple times by requests pertaining to the same recovery round. The back-off abstinence period for p expires at the point in time that is $2^{k-1}C_3\hat{d}_{hs}$ time units in the future, where $k \in \mathbb{N}^+$ is the number of recovery rounds for p that h has already initiated and $C_3 \in \mathbb{R}^{\geq 0}$ is the back-off abstinence parameter.

Our modeling of back-off abstinence periods departs slightly from SRM. Floyd *et al.* [1] propose two schemes for ensuring that requests are backed off only once per recovery round. The first scheme involves back-off abstinence periods that expire once half the time to the transmission time of the respective request has elapsed. Our use of a parameter for specifying how long to abstain from backing off allows more tuning freedom. Moreover, having back-off abstinence periods expire once half the time to the transmission time of the respective request has elapsed allows for the back-off abstinence period to overlap the interval within which requests are scheduled. This seems to go against the intention of the abstinence period. Requests received within the interval within which the current request was scheduled, should be considered to be requests of the current round and, thus, should result in the rescheduling of the current request. The second scheme annotates requests with their recovery round and backs off requests only upon receiving a request pertaining to the same or, presumably, a later round.

If a host h' receives a request for the packet p from the host h and it has already either sent or received p, then it schedules a *reply* for (retransmission of) p. This reply is scheduled for transmission at a point in time in the future that is uniformly chosen within the interval $[D_1\hat{d}_{h'h}, (D_1 + D_2)\hat{d}_{h'h}]$, where $D_1, D_2 \in \mathbb{R}^{\geq 0}$ are reply scheduling parameters and $\hat{d}_{h'h}$ is half of h''s RTT estimate to h (the requestor of p). If h' receives a reply for p while its own reply for p is pending transmission, then h' cancels its own reply for p.

Once h' either receives a reply for p or retransmits p itself, it observes a *reply abstinence period*; a period during which it refrains from scheduling replies to requests for p. The reply abstinence period for p expires at the point in time that is $D_3\hat{d}_{hh'}$ time units in the future, where $D_3 \in \mathbb{R}^{\geq 0}$ is the reply abstinence parameter. The reply abstinence period prevents multiple requests pertaining to a given recovery round from generating multiple replies.

Session Message Exchange The reliable multicast group members periodically exchange session messages. These messages carry transmission state and timing information that allow the prompt detection of packet losses and the calculation of inter-host distance estimates; within SRM, inter-host distances are quantified by the one-way transmission latency between hosts. For simplicity, we assume that hosts transmit session messages with a fixed period. In practice however, so as to limit the overhead associated with the exchange of session messages, the frequency of session message transmission is reduced as the size of the reliable multicast group grows.

Receivers detect packet losses by detecting sequence number gaps in the stream of packets received from each source. However, this approach presumes either that later packets within the sequence of transmitted packets are received, or that receivers get informed of the transmission progress of each source through a separate service. Unfortunately, relying solely on the reception of later packets may result in long recovery latencies. This is evident when the total number of packets within a sequence is unknown *a priori* and either long transmission pauses, or long loss bursts are considered. Session messages mitigate this problem by allowing reliable multicast group members to exchange transmission progress state, in terms of ADU sequence numbers that they have observed with respect to each source. Discrepancies in the observed transmission progress for each source by each host reveal whether and which packets a particular host is missing.

In addition to contributing to packet loss detection, session messages are used to calculate inter-host distance estimates. Hosts estimate the one-way transmission latencies between them by exchanging timing information through their session messages. For the purposes of illustration, we demonstrate how a host h calculates its distance estimate to a host h'. This calculation is initiated when the host h transmits a session message, p. This session message includes a field containing its transmission time t_s. Let t'_r denote the time the host h' receives p. Upon receiving p, h' records the times at which p was transmitted and received, *i.e.*, it records a tuple of the form $\langle t_s, t'_r \rangle$. Subsequently, the host h' includes the tuple $\langle t_s, t'_d \rangle$ within its next session message, p', where t'_d corresponds to the time elapsed since the host h' received p and the time h' transmits p'. Finally, letting t_r denote the point in time that h receives p', h estimates its distance $\hat{d}_{hh'}$ to h' as $(t_r - t'_d - t_s)/2$ time units.

Although the above scheme for calculating inter-host transmission latencies is simple, it presumes that inter-host transmission latencies are symmetric — the one way inter-host transmission latency is estimated as half the *round-trip-time* (RTT) between hosts. Another drawback of this scheme is the dependence of its accuracy on the frequency of session message transmission. The frequency of calculating inter-host distance estimates is dictated by the frequency of session message transmission. Thus, if the frequency of session message transmission were adjusted based on the size of the reliable multicast group, then as the group would increase in size the accuracy of the inter-host distance estimates would drop.

5.2 Formal Model of RMI

Presuming the abstract view of the physical system introduced in Section 3, RMI involves the interaction of a set of client processes, one process per host, a set of reliable multicast processes, one process per host, and an IP multicast service component. The client processes are identical to those presented in Section 4. The reliable multicast processes execute the SRM protocol. The IP multicast service component encapsulates the behavior of all communication processes at all hosts and the underlying network and provides the best-effort multicast primitive.

We model each reliable multicast process as four interacting components, each with distinct functionalities. The *membership component* manages the reliable multicast group membership of the host. It handles the join and leave requests of the client process and issues join and leave requests to the underlying IP multicast service. The *IP buffer component* buffers all packets either received from or to be transmitted using the underlying IP multicast service. The *recovery component* incorporates all the functionality pertaining to the detection and recovery of missing packets. Finally, the *reporting component* incorporates all the functionality pertaining to the exchange of session messages among the members of the reliable multicast group. Session messages are used to exchange transmission state and inter-host round-trip-time (RTT) information. This information aids the detection of losses, in particular during transmission gaps, and the calculation of inter-host round-trip-time estimates, which are required by the recovery component.

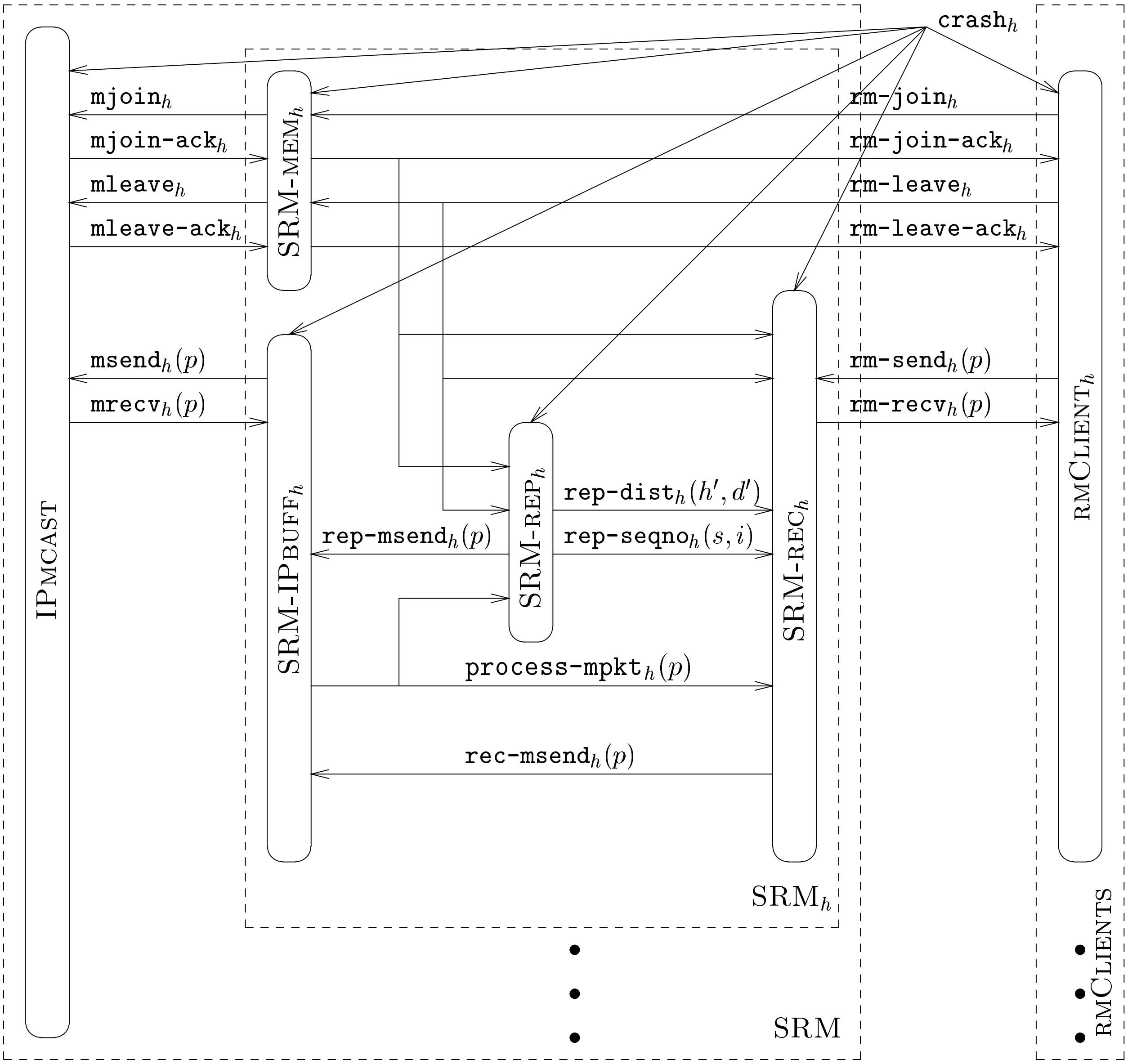
Figure 5 depicts the interaction of the various components of RMI. The reliable multicast process SRM_h at each host h is the composition of the automata $\mathrm{SRM\text{-}MEM}_h$, $\mathrm{SRM\text{-}IPBUFF}_h$, $\mathrm{SRM\text{-}REC}_h$, and $\mathrm{SRM\text{-}REP}_h$. The reliable multicast implementation as a whole, denoted SRM, is the composition of the SRM processes and the underlying IP multicast service after hiding all output actions that are not output actions of the specification $\mathrm{RM}(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$; that is, $\mathrm{SRM} = hide_\Phi(\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IPMCAST})$, with $\Phi = out(\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IPMCAST}) \setminus out(\mathrm{RM}(\Delta))$. Finally, we define RM_I to be the composition of the reliable multicast implementation with all the client automata; that is, $\mathrm{RM}_I = \mathrm{SRM} \times \mathrm{RM\text{-}CLIENTS}$.

5.2.1 Preliminary Definitions

Figure 6 contains a list of set definitions that specify the format of the various types of packets used throughout the following sections. The set $P_{\mathrm{RM\text{-}CLIENT}}$ represents the set of packets that may be transmitted by the client processes using the reliable multicast service. As defined in Section 4, for any packet $p \in P_{\mathrm{RM\text{-}CLIENT}}$ the operations $source(p)$, $seqno(p)$, and $data(p)$ extract the source, sequence number, and data segment corresponding to the packet p. For shorthand, we use the operation $id(p)$ to extract the identifier of p; that is, its source and sequence number pair.

The set P_{SRM} is comprised of all packets whose format is that used by the reliable multicast process. The format of each packet $p \in P_{\mathrm{SRM}}$ depends on its type. The type of the packet p, $type(p)$, is one of the following: `DATA`, `RQST`, `REPL`, and `SESS`. The type of p denotes whether the packet is an original transmission, a repair request, a repair reply, or a session packet, respectively. Depending

Figure 5 Reliable Multicast Implementation Component Interaction



on its type, the packet p supports a different set of operations.

When the packet p is an original transmission, that is, when $type(p)$ = `DATA`, p supports the operations $sender(p)$, $source(p)$, $seqno(p)$, $data(p)$, and $strip(p)$. These operations extract the sender, source, sequence number, data segment, and ADU corresponding to p. In the case of original transmissions, it is the case that $sender(p) = source(p)$. When p is a repair request, that is, when $type(p)$ = `RQST`, p supports the operations $sender(p)$, $source(p)$, and $seqno(p)$. These operations extract the sender, source, and sequence number corresponding to the packet p. When p is a repair reply, that is, when $type(p)$ = `REPL`, p supports the operations $sender(p)$, $source(p)$, $seqno(p)$, $data(p)$, and $strip(p)$. These operations extract the sender, source, sequence number, data segment, and ADU packet corresponding to p. For `DATA`, `RQST`, and `REPL` packets, we also use the operation $id(p)$ to extract the identifier of p; that is, its source and sequence number pair.

When the packet p is a session packet, that is, when $type(p)$ = `SESS`, p supports the operations $sender(p)$, $time\text{-}sent(p)$, $dist\text{-}rprt?(p)$, $dist\text{-}rprt(p, h)$, and $seqno\text{-}rprts(p)$. The operation $sender(p)$ extracts the sender of the session packet. The operation $time\text{-}sent(p)$ extracts the time the session packet p was sent. The operation *dist-rprt?*(p) extracts the set of hosts for which the session packet is distance reporting. The operation *dist-rprt*(p, h) extracts the distance report for h within p; that is, *dist-rprt*(p, h) corresponds to a tuple comprised of two elements: the time the most recently observed session packet sent by h was received by the sender of p and the time that elapsed between the reception of h's session packet by the sender of p and the transmission of p. The operation *seqno-rprts*(p) extracts the state reports included in p; that is, *seqno-rprts*(p) corresponds to a set of tuples, each of which is comprised of two elements: the source and the maximum sequence number observed by the sender of p to have been transmitted by this source.

The set $P_{\text{IPMCAST-CLIENT}}$ represents the set of packets that may be transmitted by the clients of the IP multicast service. For any packet $p \in P_{\text{IPMCAST-CLIENT}}$ the operations *source*(p), *seqno*(p), and *strip*(p) extract the source, the sequence number, and the data packet encapsulated in p.

The set P_{IPMCAST} is comprised of tuples, each of which describes the transmission progress of a particular packet transmitted using the IP multicast service. We refer to the tuples comprising P_{IPMCAST} as IP multicast progress packets or transmission progress tuples. For any element *pkt* of P_{IPMCAST}, the operations *strip*(*pkt*), *intended*(*pkt*), *completed*(*pkt*), *dropped*(*pkt*) extract the packet, the *intended delivery set*, the *completed delivery set*, and the *dropped set* corresponding to *pkt*. Letting $p =$ *strip*(*pkt*), the *intended delivery set* of *pkt* is the set of hosts that were and have remained members of the IP multicast group following the transmission of p. The *completed delivery set* of *pkt* is the set of hosts to which p has already been delivered. The *dropped set* of *pkt* is the set of hosts to which the IP multicast service can no longer deliver the packet p due to packet drops.

Figure 7 contains a list of set definitions used throughout the following sections.

5.2.2 The Membership Component — SRM-MEM$_h$

The SRM-MEM$_h$ timed I/O automaton specifies the membership component of the reliable multicast process. Figures 8 and 9 present the signature, the variables, and the discrete transitions of SRM-MEM$_h$.

Variables The variable *now* $\in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-MEM$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `join-rqst-pending`, `join-pending`, `join-ack-pending`, `leave-rqst-pending`, `leave-pending`, `leave-ack-pending`, `member`, and `crashed`.

The value `idle` indicates that the host h is *idle* with respect to the reliable multicast group; that is, it is neither a member, nor in the process of joining or leaving the reliable multicast group. The value `join-rqst-pending` indicates that SRM-MEM$_h$ has received a join request from the client but has yet to issue a join request to the underlying IP multicast service. The value `join-pending` indicates that SRM-MEM$_h$ has issued a join request to the underlying IP multicast service and is awaiting a join acknowledgment. The value `join-ack-pending` indicates that SRM-MEM$_h$ has successfully joined the underlying IP multicast service but has yet to issue a join acknowledgment to the client. The value `member` indicates that the host h is a member of the reliable multicast group. The value `leave-rqst-pending` indicates that SRM-MEM$_h$ has received a leave request from the client but has yet to issue a leave request to the underlying IP multicast service. The value `leave-pending` indicates that SRM-MEM$_h$ has issued a leave request to the underlying IP multicast service and is awaiting a leave acknowledgment. The value `leave-ack-pending` indicates that SRM-MEM$_h$ has successfully left the underlying IP multicast service but has yet to issue a leave acknowledgment to the client. The value `crashed` indicates that the host h has crashed. While the host h has not crashed, we say that it is *operational*. Once the host h crashes, none

Figure 6 SRM Packet Definitions

$P_{\text{RM-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{RM-CLIENT}}$

$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$suffix(p) = \{\langle s,i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \leq i\}$

$P_{\text{RM-CLIENT}}[h] = \{p \in P_{\text{RM-CLIENT}} \mid source(p) = h\}$

P_{SRM} = Set of packets such that $\forall\, p \in P_{\text{SRM}}$

$type(p) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}, \texttt{SESS}\}$

`DATA` :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$strip(p) \in P_{\text{RM-CLIENT}}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$

`RQST` :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$

`REPL` :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$strip(p) \in P_{\text{RM-CLIENT}}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$

`SESS` :
$sender(p) \in H$
$time\text{-}sent(p) \in \mathbb{R}^{\geq 0}$
$dist\text{-}rprt?(p) \subseteq H$
$dist\text{-}rprt(p,h) \in \{\langle t,t'\rangle \mid t,t' \in \mathbb{R}^{\geq 0}\}$, for all $h \in H$
$seqno\text{-}rprts(p) \subseteq \{\langle s,i\rangle \mid s \in H, i \in \mathbb{N}\}$

$P_{\text{IPMCAST-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{IPMCAST-CLIENT}}$:

$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$strip(p) \in \{0,1\}^*$

P_{IPMCAST} = Set of packets such that $\forall\, pkt \in P_{\text{IPMCAST}}$:

$strip(pkt) \in P_{\text{IPMCAST-CLIENT}}$
$intended(pkt) \subseteq H$
$completed(pkt) \subseteq H$
$dropped(pkt) \subseteq H$

Figure 7 SRM Set Definitions

$Pending\text{-}Rqsts = \{\langle s,i,t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$Scheduled\text{-}Rqsts = \{\langle s,i,t,k\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$Pending\text{-}Repls = \{\langle s,i,t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$Scheduled\text{-}Repls = \{\langle s,i,t,r\rangle \mid s,r \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$

$SRM\text{-}Status = \{\texttt{idle}, \texttt{member}, \texttt{crashed}\}$
$Joining = \{\texttt{join-rqst-pending}, \texttt{join-pending}, \texttt{join-ack-pending}\}$
$Leaving = \{\texttt{leave-rqst-pending}, \texttt{leave-pending}, \texttt{leave-ack-pending}\}$
$SRM\text{-}Mem\text{-}Status = SRM\text{-}Status \cup Joining \cup Leaving$
$Action\text{-}Pending = \{\texttt{join-rqst-pending}, \texttt{join-ack-pending}, \texttt{leave-rqst-pending}, \texttt{leave-ack-pending}\}$

$IPmcast\text{-}Status = \{\texttt{idle}, \texttt{joining}, \texttt{leaving}, \texttt{member}, \texttt{crashed}\}$

Figure 8 The SRM-MEM$_h$ Automaton — Signature

Parameters:

$h \in H$

Actions:

input
crash$_h$
rm-join$_h$
rm-leave$_h$
mjoin-ack$_h$
mleave-ack$_h$

output
mjoin$_h$
mleave$_h$
rm-join-ack$_h$
rm-leave-ack$_h$

time-passage
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Figure 9 The SRM-MEM$_h$ Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in$ *SRM-Mem-Status*, initially $status =$ `idle`

Discrete Transitions:

input crash$_h$
eff $status :=$ `crashed`

input rm-join$_h$
eff **if** $status =$ `idle` **then**
$status :=$ `join-rqst-pending`

input rm-leave$_h$
eff **if** $status \in Joining \cup \{$`member`$\}$ **then**
$status :=$ `leave-rqst-pending`

input mjoin-ack$_h$
eff **if** $status \in Joining$ **then**
$status :=$ `join-ack-pending`

input mleave-ack$_h$
eff **if** $status \in Leaving$ **then**
$status :=$ `leave-ack-pending`

output mjoin$_h$
pre $status =$ `join-rqst-pending`
eff $status :=$ `join-pending`

output mleave$_h$
pre $status =$ `leave-rqst-pending`
eff $status :=$ `leave-pending`

output rm-join-ack$_h$
pre $status =$ `join-ack-pending`
eff $status :=$ `member`

output rm-leave-ack$_h$
pre $status =$ `leave-ack-pending`
eff $status :=$ `idle`

time-passage $\nu(t)$
pre $status \notin$ *Action-Pending*
eff $now := now + t$

of the input actions of SRM-MEM$_h$ affect the state of SRM-MEM$_h$ and none of the internal and output actions of SRM-MEM$_h$, except the time passage action, are enabled.

Input Actions The input action `crash`$_h$ models the crashing of SRM-MEM$_h$. The effects of `crash`$_h$ are to set the u variable to `False`, denoting that SRM-MEM$_h$ has crashed.

The input action `rm-join`$_h$ models the client's request to join the reliable multicast group. It is effective only when the host h is idle with respect to the reliable multicast group. If the client h is already either a member of, or in the process of joining, the reliable multicast group (that is, $status \in Joining \cup \{$`member`$\}$), then the scheduling of `rm-join`$_h$ is superfluous. If the client h is already in the process of leaving the reliable multicast group (that is, $status \in Leaving$), then `rm-join`$_h$ is ignored so as to allow the ongoing process of leaving the reliable multicast group to complete. When effective, `rm-join`$_h$ initiates the process of joining the reliable multicast group by setting the *status* variable to `join-rqst-pending`.

The input action `rm-leave`$_h$ models the client's request to leave the reliable multicast group. It is effective only when the host h is either a member of, or in the process of joining, the reliable multicast group. If the host h is either already in the process of leaving, or idle with respect to the reliable multicast group, then the `rm-leave`$_h$ action is superfluous. When effective, `rm-leave`$_h$ initiates the process of leaving the reliable multicast group by setting the *status* variable to `leave-rqst-pending`.

The input action $\texttt{mjoin-ack}_h$ acknowledges that the host h has successfully joined the underlying IP multicast group. It is effective only when the host h is in the process of joining the reliable multicast group; that is, when $status \in Joining$. When effective, $\texttt{mjoin-ack}_h$ enables the I/O component to acknowledge the client's join request by setting the *status* variable to `join-ack-pending`.

The input action $\texttt{mleave-ack}_h$ acknowledges that the host h has successfully left the underlying IP multicast group. It is effective only when the host h is in the process of leaving the reliable multicast group; that is, when $status \in Leaving$. When effective, $\texttt{mleave-ack}_h$ sets the *status* variable to `leave-ack-pending`. Thus, it enables the I/O component to acknowledge the client's leave request.

Output Actions SRM-MEM$_h$ initiates the process of joining of the underlying IP multicast group by scheduling the output action $\texttt{mjoin}_h$. This action is enabled whenever the client has effectively requested to join the reliable multicast group; that is, when $status =$ `join-rqst-pending`. Its effects are to record the fact that SRM-MEM$_h$ has requested to join the IP multicast group; that is, it sets the *status* variable to `join-pending`. Joining the underlying IP multicast group is not always immediate. In order for the IP multicast service to forward packets to the host h, it may have to extend the IP multicast tree to include the host h. The time involved in extending the IP multicast tree to include the host h heavily depends on the location of the host h and the reach of the current IP multicast tree.

SRM-MEM$_h$ initiates the process of leaving of the underlying IP multicast group by scheduling the output action $\texttt{mleave}_h$. This action is enabled whenever the client has effectively requested to leave the reliable multicast group; that is, $status =$ `leave-rqst-pending`. Its effects are to record the fact that SRM-MEM$_h$ has requested to leave the IP multicast group; that is, it sets the *status* variable to `leave-pending`.

SRM-MEM$_h$ acknowledges the client's request to join the reliable multicast group by scheduling the $\texttt{rm-join-ack}_h$ output action. This action is enabled whenever the join acknowledgment is pending; that is, $status =$ `join-ack-pending`. Time is not allowed to elapse while a join acknowledgment is pending. Thus, a join acknowledgement is sent immediately after SRM-MEM$_h$ determines that it has successfully joined the IP multicast group.

SRM-MEM$_h$ acknowledges the client's request to leave the reliable multicast group by scheduling the $\texttt{rm-leave-ack}_h$ output action. This action is enabled whenever the leave acknowledgment is pending; that is, $status =$ `leave-ack-pending`. Time is not allowed to elapse while a leave acknowledgment is pending. Thus, a leave acknowledgement is sent immediately after SRM-MEM$_h$ determines that it has successfully left the IP multicast group.

Time Passage The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing while there are pending actions — either pending requests to join or leave the underlying IP multicast group, or pending acknowledgments that the client has successfully joined or left the reliable multicast group. The effects of the $\nu(t)$ action are to increment the variable *now* by t time units.

5.2.3 The IP Buffer Component — SRM-IPBUFF$_h$

The SRM-IPBUFF$_h$ timed I/O automaton specifies the IP buffer component of the reliable multicast process. Figures 10 and 11 present the signature, the variables, and the discrete transitions of SRM-IPBUFF$_h$.

Figure 10 The SRM-IPBUFF$_h$ Automaton — Signature

Parameters:

$h \in H$

Actions:

input
`crash`$_h$
`rm-join-ack`$_h$
`rm-leave`$_h$
`mrecv`$_h(p)$, for $p \in P_{\text{IPMCAST-CLIENT}}$
`rep-msend`$_h(p)$, for $p \in P_{\text{SRM}}$
`rec-msend`$_h(p)$, for $p \in P_{\text{SRM}}$

output
`process-mpkt`$_h(p)$, for $p \in P_{\text{SRM}}$
`msend`$_h(p)$, for $p \in P_{\text{IPMCAST-CLIENT}}$

time-passage
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Figure 11 The SRM-IPBUFF$_h$ Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in$ *SRM-Status*, initially $status =$ `idle`
$seqno \in \mathbb{N}$, initially $seqno = 0$
$msend\text{-}buff \subseteq P_{\text{IPMCAST-CLIENT}}$, initially $mrecv\text{-}buff = \emptyset$
$mrecv\text{-}buff \subseteq P_{\text{IPMCAST-CLIENT}}$, initially $mrecv\text{-}buff = \emptyset$

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status \neq$ `crashed` **then** $status :=$ `member`

input `rm-leave`$_h$
eff **if** $status \neq$ `crashed` **then**
Reinitialize all variables except *now* and *seqno*.

input `mrecv`$_h(p)$
eff **if** $status =$ `member` **then** $mrecv\text{-}buff \cup= \{p\}$

input `rep-msend`$_h(p)$
eff **if** $status =$ `member` **then**
$msend\text{-}buff \cup= \{comp\text{-}IPmcast\text{-}pkt(h, seqno, p)\}$
$seqno := seqno + 1$

input `rec-msend`$_h(p)$
eff **if** $status =$ `member` **then**
$msend\text{-}buff \cup= \{comp\text{-}IPmcast\text{-}pkt(h, seqno, p)\}$
$seqno := seqno + 1$

output `process-mpkt`$_h(p)$
choose $pkt \in P_{\text{IPMCAST-CLIENT}}$
pre $status =$ `member` $\wedge\ pkt \in mrecv\text{-}buff \wedge p = strip(pkt)$
eff $mrecv\text{-}buff \setminus= \{pkt\}$

output `msend`$_h(p)$
pre $status =$ `member` $\wedge\ p \in msend\text{-}buff$
eff $msend\text{-}buff \setminus= \{p\}$

time-passage $\nu(t)$
pre $status =$ `crashed` $\vee\ (msend\text{-}buff = \emptyset \wedge mrecv\text{-}buff = \emptyset)$
eff $now := now + t$

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-IPBUFF$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `member`, and `crashed`. While the host h has not crashed, we say that it is *operational*. Once the host h has crashed, none of the input actions of SRM-IPBUFF$_h$ affect the state of SRM-IPBUFF$_h$ and none of the internal and output actions of SRM-IPBUFF$_h$, except the time passage action, are enabled. The variable $seqno \in \mathbb{N}$ is a counter of the number of packets transmitted by SRM-IPBUFF$_h$ using the underlying IP multicast service.

The sets *msend-buff* and *mrecv-buff* are used to buffer all packets to be sent by and received from, respectively, the underlying IP multicast service.

Input Actions The input action `crash`$_h$ models the crashing of SRM-IPBUFF$_h$. The effects of `crash`$_h$ are to set the *status* variable to `crashed`, denoting that the host h has crashed. After the host h has crashed, the SRM-IPBUFF$_h$ automaton does not restrict time from elapsing.

The input action `rm-join-ack`$_h$ informs the SRM-IPBUFF$_h$ automaton that the host h has joined the reliable multicast group. If the host h is operational, then the action `rm-join-ack`$_h$ records the fact that the host h has joined the reliable multicast group by setting the variable *status* to `member`.

The input action `rm-leave`$_h$ informs the SRM-IPBUFF$_h$ automaton that the host h has left the reliable multicast group. If the host h is operational, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-IP$\textsc{buff}_h$ except the variables *now* and *seqno*.

The input action $\texttt{mrecv}_h(p)$ models the reception of the packet p from the underlying IP multicast service. If the host h is a member of the reliable multicast group, then the $\texttt{mrecv}_h(p)$ action adds the packet p to the *mrecv-buff* buffer. Thus, the contents of the packet p may subsequently be processed by the reliable multicast service and, when appropriate, delivered to the client.

The input actions $\texttt{rep-msend}_h(p)$ and $\texttt{rec-msend}_h(p)$ are performed by the reporting and recovery components, respectively, so as to transmit the packet p using the underlying IP multicast service. In the case of the $\texttt{rep-msend}_h(p)$ action, the packet p is a session packet. In the case of a $\texttt{rec-msend}_h(p)$ action, the packet p is either a data, a request, or a reply packet.

If the host h is a member of the reliable multicast group, then SRM-IP$\textsc{buff}_h$ encapsulates h, *seqno*, and p into a packet *pkt*, buffers *pkt* in *msend-buff* for transmission using the underlying IP multicast service, and increments *seqno*. In effect, the encapsulation of p annotates it with the host h and the value of *seqno*. Since the variable *seqno* is persistent across host joins and leaves, packets transmitted by the SRM-IP$\textsc{buff}_h$ automata, for $h \in H$, are unique.

Output Actions The output action $\texttt{process-mpkt}_h(p)$ models the processing of the packet p by the reporting and recovery components. It is enabled when the host h is a member of the reliable multicast group and there is a packet *pkt* in the *mrecv-buff* buffer, such that $strip(pkt) = p$. Its effects are to remove the element *pkt* from the *mrecv-buff* buffer.

The output action $\texttt{msend}_h(p)$ models the transmission of the packet p using the underlying IP multicast service. It is enabled when the host h is a member of the group and the packet p is in the *msend-buff* buffer. Its effects are to remove the packet p from the *msend-buff* buffer.

Time Passage The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing while the host h is operational and either of the buffers *msend-buff* and *mrecv-buff* is non-empty. The effects of the $\nu(t)$ action are to increment the variable *now* by t time units.

5.2.4 The Recovery Component — SRM-$\textsc{rec}_h$

The SRM-$\textsc{rec}_h$ timed I/O automaton specifies the recovery component of the reliable multicast service. Figure 12 presents the signature of SRM-$\textsc{rec}_h$, that is, its parameters, and actions. Figure 13 presents the variables of SRM-$\textsc{rec}_h$. Figures 14 and 15 present the discrete transitions of SRM-$\textsc{rec}_h$. In order to provide the appropriate context, the description of each of the parameters of SRM-$\textsc{rec}_h$ is deferred to appropriate places within the description of its variables and actions.

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-$\textsc{rec}_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `member`, and `crashed`. While the host h has not crashed, we say that it is *operational*. Each of the $dist(h') \in \mathbb{R}^{\geq 0}$ variables, for $h' \in H, h' \neq h$, denotes the host h's distance estimate to the host h'. Each of the $dist(h')$ variables are initialized to the parameter `DFLT-DIST`. Each of the $min\text{-}seqno(h') \in \mathbb{N}$ and $max\text{-}seqno(h') \in \mathbb{N}$ variables, for $h' \in H$, denotes the minimum and maximum ADU sequence numbers observed to have been transmitted by the host h'. The variable $archived\text{-}pkts \subseteq P_{\text{RM-Client}} \times \mathbb{R}^{\geq 0}$ is comprised of pairs involving the ADUs that have either been sent by or buffered for delivery to the client at h and the first point in time at which each ADU has either been sent by or buffered for delivery to the client at h. The variable $to\text{-}be\text{-}requested \subseteq H \times \mathbb{N}$ denotes the set of ADU packets that have been identified as missing and for which a request has yet to be scheduled. The elements of *to-be-requested* are tuples of the form $\langle s, i \rangle$, with $s \in H$ and $i \in \mathbb{N}$ denoting the source s and the sequence number i of the missing ADU.

The set *pending-rqsts* $\subseteq$ *Pending-Rqsts* is comprised of tuples that correspond to packets for which a request is pending; that is, a request for the particular packet has recently either been sent or received and a reply is being awaited. The tuples of *pending-rqsts* are of the form $\langle s, i, t \rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i represent the source and sequence number of the packet whose request is pending and t represents the back-off abstinence deadline; that is, the time before which the request timeout timer for the given packet may not be backed off. A pending request *expires* when time elapses past its back-off abstinence timeout. Prior to its expiration, a pending request is said to be *active*.

The set *scheduled-rqsts* $\subseteq$ *Scheduled-Rqsts* is comprised of tuples that correspond to packets for which a request has been scheduled and is awaiting transmission. The tuples of *scheduled-rqsts* are of the form $\langle s, i, t, k \rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$; s and i correspond to the source and sequence number of the packet to be requested, t is the time for which the request is scheduled for transmission, and k is the number of times a request for the given packet has already been scheduled.

The set *pending-repls* $\subseteq$ *Pending-Repls* is comprised of tuples that correspond to packets for which a reply has recently been either sent or received. The tuples of *pending-repls* are of the form $\langle s, i, t \rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i correspond to the source and sequence number of the packet for which a reply has already been either sent or received and t is the abstinence timeout of the reply; that is, a deadline before which replies for the given packet may not be scheduled by the host h. A pending reply *expires* when time elapses past its abstinence timeout. Prior to its expiration, a pending reply is said to be *active*.

The set *scheduled-repls* $\subseteq$ *Scheduled-Repls* is comprised of tuples that correspond to packets for which a reply has been scheduled and is awaiting transmission. The tuples comprising the set *scheduled-repls* are of the form $\langle s, i, t, r \rangle$, with $s, r \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i correspond to the source and sequence number of the packet to be retransmitted, t is the time for which the reply is scheduled for transmission, and r is the host whose request induced the scheduling of the particular reply.

The set *to-be-delivered* $\subseteq P_{\text{RM-CLIENT}}$ is used to buffer the packets that are to be subsequently delivered to the client. The set *msend-buff* $\subseteq P_{\text{SRM}}$ is used to buffer the packets that are to be subsequently multicast using the underlying IP multicast service; that is, it contains the data packets of the client and the requests and replies of the recovery component to be transmitted by the host h.

Derived Variables The derived variable *proper?*(h'), for $h' \in H$, is the set comprised of the identifiers of the packets from h' whose sequence numbers are no less than *min-seqno*(h'). The derived variable *window?*(h'), for $h' \in H$, is the set comprised of the identifiers of the packets from h' whose sequence numbers are no less than *min-seqno*(h') and no greater than *max-seqno*(h').

The derived variable *archived-pkts?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding tuple in the set *archived-pkts*. The derived variable *archived-pkts?*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' for which there is a corresponding tuple in the set *archived-pkts*.

The derived variable *to-be-requested*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' that are in the set *to-be-requested*. The derived variable *to-be-delivered?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding tuple in the set *to-be-delivered*. The derived variable *to-be-delivered?*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' that are in the set

Figure 12 The SRM-REC$_h$ Automaton — Signature

Parameters:
$h \in H, C_1, C_2, C_3, D_1, D_2, D_3 \in \mathbb{R}^{\geq 0}, \texttt{DFLT-DIST} \in \mathbb{R}^{\geq 0}$

Actions:

input
- $\texttt{crash}_h$
- $\texttt{rm-join-ack}_h$
- $\texttt{rm-leave}_h$
- $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{rep-dist}_h(h', d')$, for $h' \in H, h' \neq h, d' \in \mathbb{R}^{\geq 0}$
- $\texttt{rep-seqno}_h(s, i)$, for $s \in H, s \neq h, i \in \mathbb{N}$
- $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

internal
- $\texttt{schdl-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-repl}_h(s, i)$, for $s \in H, i \in \mathbb{N}$

output
- $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{rec-msend}_h(p)$, for $p \in P_{\text{SRM}}$

Figure 13 The SRM-REC$_h$ Automaton — Variables

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in \textit{SRM-Status}$, initially $status = \texttt{idle}$
$dist(h') \in \mathbb{R}^{\geq 0}$, for all $h' \in H, h' \neq h$, initially $dist(h') = \texttt{DFLT-DIST}$, for all $h' \in H, h' \neq h$
$\textit{min-seqno}(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $\textit{min-seqno}(h') = \perp$, for all $h' \in H$
$\textit{max-seqno}(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $\textit{max-seqno}(h') = \perp$, for all $h' \in H$
$\textit{archived-pkts} \subseteq P_{\text{RM-CLIENT}} \times \mathbb{R}^{\geq 0}$, initially $\textit{archived-pkts} = \emptyset$
$\textit{to-be-requested} \subseteq H \times \mathbb{N}$, initially $\textit{to-be-requested} = \emptyset$
$\textit{pending-rqsts} \subseteq \textit{Pending-Rqsts}$, initially $\textit{pending-rqsts} = \emptyset$
$\textit{scheduled-rqsts} \subseteq \textit{Scheduled-Rqsts}$, initially $\textit{scheduled-rqsts} = \emptyset$
$\textit{pending-repls} \subseteq \textit{Pending-Repls}$, initially $\textit{pending-repls} = \emptyset$
$\textit{scheduled-repls} \subseteq \textit{Scheduled-Repls}$, initially $\textit{scheduled-repls} = \emptyset$
$\textit{to-be-delivered} \subseteq P_{\text{RM-CLIENT}}$, initially $\textit{to-be-delivered} = \emptyset$
$\textit{msend-buff} \subseteq P_{\text{SRM}}$, initially $\textit{msend-buff} = \emptyset$

Derived Variables:

for all $h' \in H$, $\textit{proper?}(h') = \begin{cases} \emptyset & \text{if } \textit{min-seqno}(h') = \perp \\ \{\langle s, i\rangle \in H \times \mathbb{N} \mid s = h', \textit{min-seqno}(h') \leq i\} & \text{otherwise} \end{cases}$

for all $h' \in H$, $\textit{window?}(h') = \begin{cases} \emptyset & \text{if } \textit{min-seqno}(h') = \perp \\ \{\langle s, i\rangle \in H \times \mathbb{N} \mid s = h', \textit{min-seqno}(h') \leq i \leq \textit{max-seqno}(h')\} & \text{otherwise} \end{cases}$

$\textit{archived-pkts?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0} : \langle p, t\rangle \in \textit{archived-pkts} \wedge id(p) = \langle s, i\rangle\}$
$\textit{archived-pkts?}(h') = \{\langle s, i\rangle \in \textit{archived-pkts?} \mid s = h'\}$, for all $h' \in H$
$\textit{to-be-requested}(h') = \{\langle s, i\rangle \in \textit{to-be-requested} \mid s = h'\}$, for all $h' \in H$
$\textit{to-be-delivered?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, p \in \textit{to-be-delivered} : \langle s, i\rangle = id(p)\}$
$\textit{to-be-delivered?}(h') = \{\langle s, i\rangle \in \textit{to-be-delivered?} \mid s = h'\}$, for all $h' \in H$
$\textit{scheduled-rqsts?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N} : \langle s, i, t, k\rangle \in \textit{scheduled-rqsts}\}$
$\textit{scheduled-rqsts?}(h') = \{\langle s, i\rangle \in \textit{scheduled-rqsts?} \mid s = h'\}$
$\textit{scheduled-repls?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, r \in H : \langle s, i, t, r\rangle \in \textit{scheduled-repls}\}$
$\textit{pending-rqsts?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s, i, t\rangle \in \textit{pending-rqsts}\}$
$\textit{pending-repls?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s, i, t\rangle \in \textit{pending-repls}\}$

to-be-delivered?.

The derived variable *scheduled-rqsts?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding scheduled request tuple in the set *scheduled-rqsts*. The derived variable *scheduled-rqsts?*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' whose identifiers are in the set *scheduled-rqsts?*. The derived variable *scheduled-repls?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding scheduled reply tuple in the set *scheduled-repls*.

The derived variable *pending-rqsts?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is an active pending request; that is, there is a corresponding tuple in the set *pending-rqsts* whose back-off abstinence timeout has not yet expired. The derived variable *pending-repls?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is an active pending reply; that is, there is a corresponding tuple in the set *pending-repls* whose abstinence timeout has not yet expired.

Input Actions The input action $\texttt{crash}_h$ models the crashing of the host h. The effects of $\texttt{crash}_h$ are to set the *status* variable to `crashed`. Once the host h has crashed, none of the input actions of SRM-$\textsc{rec}_h$ affect its state, none of the internal and output actions of SRM-$\textsc{rec}_h$ are enabled, and time is not restricted from elapsing.

The input action $\texttt{rm-join-ack}_h$ informs the SRM-$\textsc{rec}_h$ automaton that the host h has joined the reliable multicast group. If the host h is operational, then the $\texttt{rm-join-ack}_h$ action records the fact that the host h has joined the reliable multicast group by setting the variable *status* to `member`. Subsequently, SRM-$\textsc{rec}_h$ may transmit, process, and deliver packets and schedule packet requests and replies.

The input action $\texttt{rm-leave}_h$ informs the SRM-$\textsc{rec}_h$ automaton that the host h has left the reliable multicast group. If the host h is operational, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-$\textsc{rec}_h$ except the variable *now*. Subsequently, SRM-$\textsc{rec}_h$ automaton ceases transmitting, processing, and delivering packets and scheduling packet requests and replies.

The input action $\texttt{rm-send}_h(p)$ models the transmission of the packet p by the client at h using the reliable multicast service. $\texttt{rm-send}_h(p)$ is effective only when the host h is a member of the reliable multicast group and the host h is the source of the packet p. If p is the first packet to be transmitted by the client since it last joined the reliable multicast group, the $\texttt{rm-send}_h(p)$ action sets the *min-seqno*(h) variable to the sequence number of p. Otherwise, SRM-$\textsc{rec}_h$ ensures that p corresponds to the next packet awaited; that is, the packet whose sequence number is one larger than the sequence number of the latest packet transmitted by h. If so, SRM-$\textsc{rec}_h$ updates *max-seqno*(h), archives p, and generates a `DATA` packet to subsequently be transmitted to the other members of the reliable multicast group through the underlying IP multicast service. The operation *comp-data-pkt*(p) composes a `DATA` packet corresponding to the client packet p.

Each input action $\texttt{rep-dist}_h(h', d')$, for $h' \in H, h' \neq h, d' \in \mathbb{R}^{\geq 0}$, reports to SRM-$\textsc{rec}_h$ an updated distance estimate d' to h'. If the host h is a member of the reliable multicast group, then the $\texttt{rep-dist}_h(h', d')$ action sets the variable *dist*(h') to the value d'.

Each input action $\texttt{rep-seqno}_h(s, i)$, for $s \in H, s \neq h, i \in \mathbb{N}$, reports to SRM-$\textsc{rec}_h$ the latest observed sequence number i for the source s. If the host h is a member of the reliable multicast group, $\langle s, i\rangle$ corresponds to a proper packet, and i is greater than *max-seqno*(s), then the $\texttt{rep-seqno}_h(s, i)$ action adds the packets from s with sequence numbers ranging from *max-seqno*$(s) + 1$ to i to the set *to-be-requested* and sets *max-seqno*(s) to i.

The input action $\texttt{process-mpkt}_h(p)$ models the processing of the packet p by SRM-$\textsc{rec}_h$. The packet p is processed only when the host h is a member of the reliable multicast group. We proceed by describing the effects of $\texttt{process-mpkt}_h(p)$ depending on the type of the packet p. When p is either a `DATA`, `RQST`, or `REPL` packet, we let $s_p \in H$ and $i_p \in \mathbb{N}$ denote the source and the sequence number pertaining to the packet p.

First, consider the case where p is a `DATA` packet. If h is not the source of p and p is the first packet from s_p to be received by h, then the variables *min-seqno*(s_p) and *max-seqno*(s_p) are set to i_p. Following this initial assignment of *min-seqno*(s_p) to i_p, all `DATA`, `RQST`, and `REPL` packets pertaining to ADUs from s_p with sequence numbers less than i_p are considered *improper* and are discarded. Conversely, all `DATA`, `RQST`, and `REPL` packets pertaining to ADUs from s_p with sequence numbers equal to or greater than i_p are considered *proper* and are processed.

The processing of packet p proceeds only while it is considered a proper packet. Unless either h is the source of p or p is already archived, p is archived by adding the tuple $\{\langle strip(p), now\rangle\}$ to *archived-pkts*. Unless h is the source of p, the ADU contained in p is buffered in *to-be-delivered* so that it may subsequently be delivered to the client. Thus, the reliable multicast process does not deliver packets sent by a client to itself. Moreover, the reliable multicast service may also deliver

Figure 14 The SRM-REC_h Automaton — Discrete Transitions

input $\texttt{crash}_h$
eff $\mathit{status} := \texttt{crashed}$

input $\texttt{rm-join-ack}_h$
eff **if** $\mathit{status} \neq \texttt{crashed}$ **then** $\mathit{status} := \texttt{member}$

input $\texttt{rm-leave}_h$
eff **if** $\mathit{status} \neq \texttt{crashed}$ **then**
Reinitialize all variables except now.

input $\texttt{rm-send}_h(p)$
eff **if** $\mathit{status} = \texttt{member} \wedge h = \mathit{source}(p)$ **then**
$\langle s_p, i_p\rangle = \mathit{id}(p)$
\\ `Record foremost DATA packet`
if $\mathit{min\text{-}seqno}(s_p) = \perp$ **then** $\mathit{min\text{-}seqno}(s_p) := i_p$
\\ `Only consider next packet`
if $\mathit{max\text{-}seqno}(s_p) = \perp$
$\vee i_p = \mathit{max\text{-}seqno}(s_p) + 1$
then
$\mathit{max\text{-}seqno}(s_p) := i_p$
\\ `Archive packet`
$\mathit{archived\text{-}pkts} \cup= \{\langle p, \mathit{now}\rangle\}$
\\ `Compose data packet`
$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}data\text{-}pkt}(p)\}$

input $\texttt{rep-dist}_h(h', d')$
eff **if** $\mathit{status} = \texttt{member}$ **then**
$\mathit{dist}(h') := d'$

input $\texttt{rep-seqno}_h(s, i)$
eff **if** $\mathit{status} = \texttt{member}$
$\wedge \mathit{min\text{-}seqno}(s) \neq \perp \wedge \mathit{max\text{-}seqno}(s) < i$
then
$\mathit{to\text{-}be\text{-}requested} \cup=$
$\{\langle s, i'\rangle \mid i' \in \mathbb{N}, \mathit{max\text{-}seqno}(s) < i' \leq i\}$
$\mathit{max\text{-}seqno}(s) := i$

internal $\texttt{schdl-rqst}_h(s, i)$
pre $\mathit{status} = \texttt{member} \wedge \langle s, i\rangle \in \mathit{to\text{-}be\text{-}requested}$
eff \\ `Schedule new request`
$k_r := 1$; $d_r := \mathit{dist}(s)$
$t_r :\in \mathit{now} + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$\mathit{scheduled\text{-}rqsts} \cup= \{\langle s, i, t_r, k_r\rangle\}$
\\ `Pkt request has been scheduled`
$\mathit{to\text{-}be\text{-}requested} \setminus= \{\langle s, i\rangle\}$

internal $\texttt{send-rqst}_h(s, i)$
choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$
pre $\mathit{status} = \texttt{member}$
$\wedge t = \mathit{now} \wedge \langle s, i, t, k\rangle \in \mathit{scheduled\text{-}rqsts}$
eff \\ `Compose request packet`
$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}rqst\text{-}pkt}(h, \langle s, i\rangle)\}$
\\ `Back-off scheduled request`
$\mathit{scheduled\text{-}rqsts} \setminus= \{\langle s, i, t, k\rangle\}$
$k_r := k + 1$; $d_r := \mathit{dist}(s)$
$t_r :\in \mathit{now} + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$\mathit{scheduled\text{-}rqsts} \cup= \{\langle s, i, t_r, k_r\rangle\}$
\\ `A request becomes pending`
$\mathit{pending\text{-}rqsts} \setminus= \{\langle s, i, t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_r := \mathit{now} + 2^{k_r - 1} C_3 d_r$
$\mathit{pending\text{-}rqsts} \cup= \{\langle s, i, t_r\rangle\}$

internal $\texttt{send-repl}_h(s, i)$
choose $t \in \mathbb{R}^{\geq 0}, r \in H$
pre $\mathit{status} = \texttt{member}$
$\wedge t = \mathit{now} \wedge \langle s, i, t, r\rangle \in \mathit{scheduled\text{-}repls}$
eff \\ `Compose reply packet`
choose $p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0}$
where $\langle p, t\rangle \in \mathit{archived\text{-}pkts} \wedge \mathit{id}(p) = \langle s, i\rangle$
$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}repl\text{-}pkt}(h, p)\}$
\\ `A reply becomes pending`
$\mathit{pending\text{-}repls} \setminus= \{\langle s, i, t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_{repl} := \mathit{now} + D_3\, \mathit{dist}(r)$
$\mathit{pending\text{-}repls} \cup= \{\langle s, i, t_{repl}\rangle\}$
\\ `Cancel scheduled reply`
$\mathit{scheduled\text{-}repls} \setminus= \{\langle s, i, t, r\rangle\}$

output $\texttt{rm-recv}_h(p)$
pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{to\text{-}be\text{-}delivered}$
$\wedge(\nexists\, p' \in \mathit{to\text{-}be\text{-}delivered}:$
$\mathit{source}(p') = \mathit{source}(p) \wedge \mathit{seqno}(p') < \mathit{seqno}(p))$
eff $\mathit{to\text{-}be\text{-}delivered} \setminus= \{p\}$

output $\texttt{rec-msend}_h(p)$
pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{msend\text{-}buff}$
eff $\mathit{msend\text{-}buff} \setminus= \{p\}$

time-passage $\nu(t)$
pre $\mathit{status} = \texttt{crashed}$
$\vee(\mathit{to\text{-}be\text{-}requested} = \emptyset \wedge \mathit{to\text{-}be\text{-}delivered} = \emptyset$
$\wedge \mathit{msend\text{-}buff} = \emptyset$
$\wedge$ no requests scheduled earlier than $\mathit{now} + t$
$\wedge$ no replies scheduled earlier than $\mathit{now} + t)$
eff $\mathit{now} := \mathit{now} + t$

the same ADU to the client multiple times. The identifier of the ADU pertaining to p is removed from the *to-be-requested* set and any scheduled requests and replies for the ADU pertaining to p are canceled. Finally, unless h is the source of p, SRM-REC_h adds any trailing missing packets to the set *to-be-requested*, so that a request for each of them may subsequently be scheduled.

Second, consider the case where p is a `REPL` packet. The processing of a a `REPL` packet is similar to that of a `DATA` packet. The differences are that p is processed only if it pertains to a proper ADU and that in addition to the effects of processing a `DATA` packet, a reply for the given ADU becomes pending. While this pending reply is active, SRM-REC_h does not schedule replies for the ADU pertaining to p.

Third, consider the case where p is a `RQST` packet. Once again, p is processed only if it pertains to a proper ADU. If p pertains to an ADU that has been archived and for which a reply is neither scheduled, nor pending, then SRM-REC_h schedules a retransmission of the requested ADU. This retransmission is scheduled for a point it time in the future that is chosen uniformly within the interval $\mathit{now} + [D_1 d_{repl}, (D_1 + D_2) d_{repl}]$, with $d_{repl} = \mathit{dist}(\mathit{sender}(p))$. If p pertains to an ADU that has not been archived, then the effects of $\texttt{process-mpkt}_h(p)$ depend on whether there is a request for the given ADU already scheduled. If h is not the source of p and there is no request for the ADU of p already scheduled, then a request for the given ADU is scheduled. This request is scheduled for a point it time in the future that is chosen uniformly within the interval $now + 2[C_1 d_r, (C_1 + C_2)d_r]$, with $d_r = dist(s_p)$; that is, the request is scheduled as if a first round request is being backed off. If h is not the source of p, there is a request for the ADU of p already scheduled and there, are there are no pending requests for the ADU of p still active, then the request for the ADU of p that is already scheduled is exponentially backed off. When either a new request is scheduled or an existing request is backed-off, a request for the given ADU becomes pending with a back-off abstinence timeout equal to $now + 2^{k-1}C_3 d_r$, where k is the round of the rescheduled request and $d_r = dist(s_p)$. Finally, unless h is the source of p, SRM-REC$_h$ adds any trailing missing packets to the set *to-be-requested*, so that a request for each of them may subsequently be scheduled.

Finally, in the case where p is a `SESS` packet, the $\texttt{process-mpkt}_h(p)$ action does not affect the state of SRM-REC$_h$; `SESS` packets are in effect discarded by the SRM-REC$_h$ automaton.

Internal Actions Each internal action $\texttt{schdl-rqst}_h(s,i)$, for $s \in H, s \neq h, i \in \mathbb{N}$, schedules a request for the packet $\langle s,i \rangle$. The precondition of the $\texttt{schdl-rqst}_h(s,i)$ action is that the host h is a member of the reliable multicast group and the tuple $\langle s,i \rangle$ is in the set *to-be-requested*. The effects of the $\texttt{schdl-rqst}_h(s,i)$ action are to schedule a new request for a point in time in the future that is chosen uniformly within the interval $now + [C_1 d_r, (C_1 + C_2)d_r]$, with $d_r = dist(s)$, and to remove the tuple $\langle s,i \rangle$ from the set *to-be-requested*.

Each internal action $\texttt{send-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled request for the packet $\langle s,i \rangle$. The precondition of $\texttt{send-rqst}_h(s,i)$ is that the host h is a member of the reliable multicast group and a previously scheduled request for the packet $\langle s,i \rangle$ has expired; that is, there is a tuple $\langle s,i,t,k \rangle$ in *scheduled-rqsts* such that $t = now$. Let the tuple $\langle s,i,t,k \rangle$ be the element of *scheduled-rqsts* corresponding to the packet $\langle s,i \rangle$. $\texttt{send-rqst}_h(s,i)$ composes a request packet and adds it to the buffer *msend-buff*. The operation $comp\text{-}rqst\text{-}pkt(h, \langle s,i \rangle)$ composes a `RQST` packet from h for the packet $\langle s,i \rangle$.

Moreover, the request $\langle s,i,t,k \rangle$ is backed off and a request for the given ADU becomes pending. The timeout timer of the rescheduled request is set to a point it time in the future that is chosen uniformly within the interval $now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2)d_r]$ and the back-off abstinence timeout of the pending request is set to $now + 2^{k_r - 1}C_3 d_r$, with $k_r = k + 1$ and $d_r = dist(s)$.

Each internal action $\texttt{send-repl}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled reply for the packet $\langle s,i \rangle$. The precondition of $\texttt{send-repl}_h(s,i)$ is that the host h is a member of the reliable multicast group and a previously scheduled reply for the packet $\langle s,i \rangle$ has expired; that is, there is a tuple $\langle s,i,t,r \rangle$ in *scheduled-repls* such that $t = now$. Let the tuple $\langle s,i,t,r \rangle$ be the element of *scheduled-repls* corresponding to the packet $\langle s,i \rangle$. $\texttt{send-repl}_h(s,i)$ composes a reply packet and adds it to the buffer *msend-buff*. The operation $comp\text{-}repl\text{-}pkt(h,p)$ composes a `REPL` packet from h for the packet p.

Moreover, the tuple corresponding to $\langle s,i \rangle$ is removed from the set *scheduled-repls* and a tuple corresponding to $\langle s,i \rangle$ is added to the set *pending-repls*. The reply abstinence timeout of this pending reply is set to $now + D_3\, dist(r)$. This pending reply prevents the scheduling of replies for the given ADU for $D_3\, dist(r)$ time units.

Output Actions Each output action $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, models the delivery of the packet p to the client. It is enabled when the host h is a member of the reliable multicast group and the packet p is the packet in the *to-be-delivered* buffer with the smallest sequence number.

Figure 15 The SRM-$\textsc{rec}_h$ Automaton — Discrete Transitions (Cnt'd)

```
input process-mpkt_h(p)
where type(p) = DATA
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Record foremost DATA packet
       if h ≠ s_p ∧ min-seqno(s_p) = ⊥ then
         min-seqno(s_p) := i_p; max-seqno(s_p) := i_p
       \\ Only consider proper packets
       if min-seqno(s_p) ≠ ⊥ ∧ min-seqno(s_p) ≤ i_p then
         \\ Archive and deliver packet
         if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ archived-pkts? then
           archived-pkts ∪= {⟨strip(p), now⟩}
         if h ≠ s_p then to-be-delivered ∪= {strip(p)}
         \\ Pkt need not be requested
         to-be-requested \= {⟨s_p, i_p⟩}
         \\ Cancel any scheduled requests and replies
         scheduled-rqsts \= {⟨s_p, i_p, t, k⟩ | t ∈ ℝ^{≥0}, k ∈ ℕ}
         scheduled-repls \= {⟨s_p, i_p, t, r⟩ | t ∈ ℝ^{≥0}, r ∈ H}
         \\ Cancel any pending requests
         pending-rqsts \= {⟨s_p, i_p, t⟩ | t ∈ ℝ^{≥0}}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           to-be-requested ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i < i_p}
           max-seqno(s_p) := i_p

input process-mpkt_h(p)
where type(p) = REPL
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Only consider proper packets
       if min-seqno(s_p) ≠ ⊥ ∧ min-seqno(s_p) ≤ i_p then
         \\ A reply becomes pending
         pending-repls \= {⟨s_p, i_p, t_*⟩ | t_* ∈ ℝ^{≥0}}
         t_repl := now + D_3 dist(s_p)
         pending-repls ∪= {⟨s_p, i_p, t_repl⟩}
         \\ Archive and deliver packet
         if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ archived-pkts? then
           archived-pkts ∪= {⟨strip(p), now⟩}
         if h ≠ s_p then to-be-delivered ∪= {strip(p)}
         \\ Pkt need not be requested
         to-be-requested \= {⟨s_p, i_p⟩}
         \\ Cancel any scheduled requests and replies
         scheduled-rqsts \= {⟨s_p, i_p, t, k⟩ | t ∈ ℝ^{≥0}, k ∈ ℕ}
         scheduled-repls \= {⟨s_p, i_p, t, r⟩ | t ∈ ℝ^{≥0}, r ∈ H}
         \\ Cancel any pending requests
         pending-rqsts \= {⟨s_p, i_p, t⟩ | t ∈ ℝ^{≥0}}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           to-be-requested ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i < i_p}
           max-seqno(s_p) := i_p

input process-mpkt_h(p)
where type(p) = RQST
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Only consider proper packets
       if min-seqno(s_p) ≠ ⊥ ∧ min-seqno(s_p) ≤ i_p then
         if h ≠ s_p then
           if ⟨s_p, i_p⟩ ∈ archived-pkts? then
             if ⟨s_p, i_p⟩ ∉ scheduled-repls?
               ∧ ⟨s_p, i_p⟩ ∉ pending-repls?
             then
               \\ Schedule a new reply
               d_repl := dist(sender(p))
               t_repl :∈ now + [D_1 d_repl, (D_1 + D_2) d_repl]
               r_repl := sender(p)
               scheduled-repls ∪= {⟨s_p, i_p, t_repl, r_repl⟩}
           else
             if ⟨s_p, i_p⟩ ∉ scheduled-rqsts? then
               \\ Schedule a backed-off request
               k_r := 2; d_r := dist(s_p)
               t_r :∈ now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]
               scheduled-rqsts ∪= {⟨s_p, i_p, t_r, k_r⟩}
               \\ Pkt request has been scheduled
               to-be-requested \= {⟨s_p, i_p⟩}
               \\ A request becomes pending
               pending-rqsts \= {⟨s_p, i_p, t_*⟩ | t_* ∈ ℝ^{≥0}}
               t_r := now + 2^{k_r - 1} C_3 d_r
               pending-rqsts ∪= {⟨s_p, i_p, t_r⟩}
             else
               if ⟨s_p, i_p⟩ ∉ pending-rqsts? then
                 \\ Backoff scheduled request
                 choose t ∈ ℝ^{≥0}, k ∈ ℕ
                   where ⟨s_p, i_p, t, k⟩ ∈ scheduled-rqsts
                 scheduled-rqsts \= {⟨s_p, i_p, t, k⟩}
                 k_r := k + 1; d_r := dist(s_p)
                 t_r :∈ now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]
                 scheduled-rqsts ∪= {⟨s_p, i_p, t_r, k_r⟩}
                 \\ A request becomes pending
                 pending-rqsts \= {⟨s_p, i_p, t_*⟩ | t_* ∈ ℝ^{≥0}}
                 t_r := now + 2^{k_r - 1} C_3 d_r
                 pending-rqsts ∪= {⟨s_p, i_p, t_r⟩}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           to-be-requested ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i < i_p}
           max-seqno(s_p) := i_p

input process-mpkt_h(p)
where type(p) = SESS
eff  None
```

This ordering constraint ensures that the foremost packet received of any source is delivered to the client prior to any other packet from the particular source. Its effects are to remove the packet p from the *rm-recv-buff* buffer.

Each output action $\texttt{rec-msend}_h(p)$, for $p \in P_{\mathrm{SRM}}$, hands off the packet p from SRM-$\textsc{rec}_h$ to SRM-IP$\textsc{buff}_h$ so that it may subsequently be multicast by SRM-IP$\textsc{buff}_h$ using the underlying IP multicast service. The precondition of the $\texttt{rec-msend}_h(p)$ action is that the host h is a member of the reliable multicast group and p is in the *msend-buff* buffer. Its effects are to remove p from the *msend-buff* buffer.

Time Passage The action $\nu(t)$ models the passage of t time units. If the host h has crashed, then time is allowed to elapse. Otherwise, time is prevented from elapsing while either there are packets in the delivery and IP multicast transmission buffers or there are packets which have been declared missing but for which a request has yet to be scheduled; that is, while either the buffer *to-be-delivered*, the buffer *msend-buff*, or the set *to-be-requested* is non-empty. Furthermore, time is prevented from elapsing past the transmission deadline of any scheduled requests or replies.

5.2.5 The Reporting Component — SRM-REP$_h$

The SRM-REP$_h$ timed I/O automaton specifies the reporting component of the reliable multicast process at each host $h \in H$. Figures 16, 17, and 18 present the signature, the variables, and the discrete transitions of SRM-REP$_h$, respectively.

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-REP$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `member`, and `crashed`. While the host h has not crashed, we say that it is *operational*. The variable $\mathit{rep\text{-}deadline} \in \mathbb{R}^{\geq 0} \cup \perp$ denotes the point in time at which the next session packet is scheduled for transmission. The variable *rep-deadline* is equal to $\perp$ when undefined.

The variable $\mathit{dist\text{-}rprt}(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \cup \perp$, for each $h' \in H, h' \neq h$, records the transmission and the reception times of the most recent session packet of h' to be received by the host h. That is, for each $h' \in H$, the variable $\mathit{dist\text{-}rprt}(h')$ is a tuple of the form $\langle t_{sent}, t_{rcvd} \rangle$, where t_{sent} is the transmission time of the most recent session packet of h' to be received by h and t_{rcvd} is the reception time of this session packet by h. If the host h has not received a session packet from the host h' since joining the reliable multicast group, then the variable $\mathit{dist\text{-}rprt}(h')$ is undefined; that is, $\mathit{dist\text{-}rprt}(h') = \perp$.

The variable $\mathit{dist}(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, for each $h' \in H, h' \neq h$, records the most up-to-date estimate of the distance from h to the host h'. Such distance estimates are ordered by the transmission time of the session packet of h that initiated their calculation; that is, a distance estimate calculated as a result of the transmission of a more recent session packet of h is considered more up-to-date. If two calculations are initiated by the same session packet of h, then the later calculation is considered more up-to-date. Thus, for each $h' \in H$, the variable $\mathit{dist}(h')$ is a tuple of the form $\langle t_{rprt}, t_{dist} \rangle$, where t_{rprt} is the transmission time of the session packet of h that initiated the particular distance estimate calculation and t_{dist} is the distance estimate obtained as a result of the particular calculation.

The variable $\mathit{max\text{-}seqno}(h') \in \mathbb{N} \cup \perp$, for each $h' \in H, h' \neq h$, records the latest sequence number of h' to have been observed by h. Recall that h may observe the transmission progress of other hosts by examining any type of packet. If the host h has not yet observed the transmission of any packets from the host h', then the variable $\mathit{max\text{-}seqno}(h')$ is undefined; that is, $\mathit{max\text{-}seqno}(h') = \perp$.

The variable $\mathit{dist\text{-}buff} \subseteq H$ contains the hosts whose distance estimates have recently been updated but have not yet been reported to the SRM-REC$_h$ automaton. Similarly, the variable *seqno-buff* contains the hosts whose maximum observed sequence numbers have recently been updated but have not yet been reported to the SRM-REC$_h$ automaton.

Derived Variables The derived variable *dist-rprt* records the transmission and the reception times of the most recent session packet of all other hosts. *dist-rprt* is the set of tuples of the form $\langle h', t_s, t_r \rangle$, with $\langle t_s, t_r \rangle = \mathit{dist\text{-}rprt}(h')$, for $h' \in H, h' \neq h$, and $\mathit{dist\text{-}rprt}(h') \neq \perp$. In effect, *dist-rprt* summarizes the information recorded by the $\mathit{dist\text{-}rprt}(h')$ variables, for all $h' \in H, h' \neq h$.

Figure 16 The SRM-REP_h Automaton — Signature

Parameters:

$h \in H$, $\texttt{DFLT-DIST} \in \mathbb{R}^{\geq 0}$, $\texttt{SESS-PERIOD} \in \mathbb{R}^{+}$

Actions:

input
$\texttt{crash}_h$
$\texttt{rm-join-ack}_h$
$\texttt{rm-leave}_h$
$\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$

time-passage
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

output
$\texttt{rep-msend}_h(p)$, for $p \in P_{\text{SRM}}$
$\texttt{rep-dist}_h(h', d')$, for $h' \in H, h' \neq h, d \in \mathbb{R}^{\geq 0}$
$\texttt{rep-seqno}_h(s, i)$, for $s \in H, s \neq h, i \in \mathbb{N}$

Figure 17 The SRM-REP_h Automaton — Variables

Variables:

$\mathit{now} \in \mathbb{R}^{\geq 0}$, initially $\mathit{now} = 0$
$\mathit{status} \in \mathit{SRM\text{-}Status}$, initially $\mathit{status} = \texttt{idle}$
$\mathit{rep\text{-}deadline} \in \mathbb{R}^{\geq 0} \cup \perp$, initially $\mathit{rep\text{-}deadline} = \perp$
$\mathit{dist\text{-}rprt}(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \cup \perp$, for all $h' \in H, h' \neq h$, initially $\mathit{dist\text{-}rprt}(h') = \perp$
$\mathit{dist}(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, for all $h' \in H, h' \neq h$, initially $\mathit{dist}(h') = \langle 0, \texttt{DFLT-DIST} \rangle$
$\mathit{max\text{-}seqno}(h') \in \mathbb{N} \cup \perp$, for all $h' \in H, h' \neq h$, initially $\mathit{max\text{-}seqno}(h') = \perp$
$\mathit{dist\text{-}buff} \subseteq H$, initially $\mathit{dist\text{-}buff} = \emptyset$
$\mathit{seqno\text{-}buff} \subseteq H$, initially $\mathit{seqno\text{-}buff} = \emptyset$

Derived Variables:

$\mathit{dist\text{-}rprt} = \cup_{h' \in H, h' \neq h, \mathit{dist\text{-}rprt}(h') \neq \perp} \{\langle h', t_{sent}, t_{rcvd} \rangle \mid \mathit{dist\text{-}rprt}(h') = \langle t_{sent}, t_{rcvd} \rangle\}$
$\mathit{max\text{-}seqno} = \cup_{h' \in H, h' \neq h, \mathit{max\text{-}seqno}(h') \neq \perp} \{\langle h', \mathit{max\text{-}seqno}(h') \rangle\}$

The derived variable *max-seqno* records the transmission progress of all other hosts. *max-seqno* is the set of tuples of the form $\langle h', \mathit{max\text{-}seqno}(h') \rangle$, for $h' \in H, h' \neq h$, and $\mathit{max\text{-}seqno}(h') \neq \perp$. In effect, *max-seqno* summarizes the information recorded by the $\mathit{max\text{-}seqno}(h')$ variables, for all $h' \in H, h' \neq h$.

Input Actions As in the case of the SRM-IPBUFF_h and SRM-REC_h automata, the input action $\texttt{crash}_h$ models the crashing of the host h. The effects of the action $\texttt{crash}_h$ are to set the *status* variable to `crashed`, denoting that the host h has crashed. Once the host h has crashed, none of the input actions affect the state of SRM-REP_h, none of the internal and output actions are enabled, and time is not restricted from elapsing.

The input action $\texttt{rm-join-ack}_h$ informs the SRM-REP_h automaton that the host h has joined the reliable multicast group. If the host h is operational, then the $\texttt{rm-join-ack}_h$ action records the fact that the host h has joined the reliable multicast group by setting the variable *status* to `member`. Moreover, it schedules the transmission of a session packet no later than `SESS-PERIOD` time units in the future by setting the *rep-deadline* variable to a value that is uniformly chosen within the interval $\mathit{now} + (0, \texttt{SESS-PERIOD}]$.

The input action $\texttt{rm-leave}_h$ informs the SRM-REP_h automaton that the host h has left the reliable multicast group. If the host h is operational, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REP_h except the variable *now*.

The input action $\texttt{process-mpkt}_h(p)$ processes the packet p. Recall that the functionality of the reporting component includes tracking the transmission progress of all sources and estimating the distance estimates from the host h to all other reliable multicast group members. Provided the host h is a member of the reliable multicast group, the packet p is processed according to its packet type.

We first consider the case where p is a `SESS` packet. Letting s_p denote the sender of p, SRM-REP_h checks whether p is either the first or the most recent session packet of s_p to be received by h. If so, the variable $\mathit{dist\text{-}rprt}(s_p)$ is set to $\langle \mathit{time\text{-}sent}(p), \mathit{now} \rangle$ to record the reception of a more recent session packet from the host s_p.

Then, if p is distance reporting for h and the session packet that initiated this report is at least as recent as the session packet that initiated the calculation of the current distance estimate to s_p, then a new distance estimate to s_p is calculated. If the calculation of the current distance estimate was initiated by the same session packet as the new calculation, then the new distance estimate is considered more recent since the latency observed from s_p to h is more recent. SRM-REP_h records the new distance estimate to s_p by reassigning the tuple $dist(s_p)$. Furthermore, s_p is added to the *dist-buff* buffer so that SRM-REP_h may subsequently report to SRM-REC_h the new distance estimate to s_p.

Finally, SRM-REP_h goes through the transmission state reports contained in p to determine whether s_p has observed further progress in the transmission of any of the sources; that is, whether s_p has observed the transmission of later ADU packets by any of the sources. For each state report indicating further transmission progress, the corresponding *max-seqno* variable is updated. Moreover, the respective source is added to the *seqno-buff* buffer so that SRM-REP_h may subsequently report this transmission progress of the respective source to SRM-REC_h.

We now consider the case where p is either a `DATA`, `RQST`, or `REPL` packet. Let s_p and i_p denote the source and sequence number of the ADU packet contained in p. If the packet p is a `DATA` packet and is the first data packet to be received from s_p, that is, if $max\text{-}seqno(s_p) = \perp$, then $max\text{-}seqno(s_p)$ is set to i_p. If the packet p is either a `DATA`, `RQST`, or `REPL` packet and i_p is greater than $max\text{-}seqno(s_p)$, then $max\text{-}seqno(s_p)$ is set to i_p.

Output Actions The output action $\texttt{rep-msend}_h(p)$, for $p \in P_{\text{SRM}}$, hands off the packet p to SRM-IPBUFF_h so that it may subsequently be multicast by SRM-IPBUFF_h using the underlying IP multicast service. The precondition of the $\texttt{rep-msend}_h(p)$ action is that the host h is a member of the reliable multicast group, the variable *now* equals the session packet deadline *rep-deadline*, and the packet p corresponds to a session packet pertaining to the current state of the SRM-REP_h automaton. The operation $comp\text{-}sess\text{-}pkt(h, now, dist\text{-}rprt, seqno)$ composes the session packet p. $\texttt{rep-msend}_h(p)$ schedules the transmission of the next session packet by setting the *rep-deadline* to `SESS-PERIOD` time units in the future. The parameter `SESS-PERIOD` of the SRM-REP_h automaton specifies the period with which the host h transmits session packets.

The output action $\texttt{rep-dist}_h(h', d')$ reports to SRM-REC_h the most recent distance estimate d' to the host h'. The action $\texttt{rep-dist}_h(h', d')$ is enabled when the host h is a member of the reliable multicast group, the distance estimate to h' has recently been updated but has yet to be reported to SRM-REC_h, that is, $h' \in dist\text{-}buff$, and the distance d' is the most recent distance estimate to h', that is, it is the distance component of the tuple $dist(h')$. The effects of $\texttt{rep-dist}_h(h', d')$ are to remove the host h' from the *dist-buff* buffer.

The output action $\texttt{rep-seqno}_h(s, i)$ reports to SRM-REC_h the most recent maximum sequence number observed for the source s. The action $\texttt{rep-seqno}_h(s, i)$ is enabled when the host h is a member of the reliable multicast group, the maximum sequence number for the source s has recently been updated but has yet to be reported to SRM-REC_h, that is, $s \in seqno\text{-}buff$, and i is the most recently recorded maximum sequence number for the source s, that is, $i = max\text{-}seqno(s)$. The effects of $\texttt{rep-seqno}_h(s, i)$ are to remove the source s from the *seqno-buff* buffer.

Time Passage The time passage action $\nu(t)$ models the passage of t time units of time. If the host h has crashed, then time is allowed to elapse. Otherwise, time is allowed to elapse neither past the transmission of the next session packet, *rep-deadline*, nor while there are pending reports; that is, the reporting buffers *dist-buff* and *seqno-buff* are non-empty.

Figure 18 The SRM-REP$_h$ Automaton — Discrete Transitions

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status \neq$ `crashed` **then**
  $status :=$ `member`
  $rep\text{-}deadline :\in now + (0, \texttt{SESS-PERIOD}]$

input `rm-leave`$_h$
eff **if** $status \neq$ `crashed` **then**
  Reinitialize all variables except now.

input `process-mpkt`$_h(p)$
where $type(p) = \texttt{SESS}$
eff **if** $status =$ `member` **then**
  $s_p := sender(p)$
  if $dist\text{-}rprt(s_p) = \perp$ **then**
    $dist\text{-}rprt(s_p) := \langle time\text{-}sent(p), now \rangle$
  else
    $\langle t_{sent}, t_{rcvd} \rangle := dist\text{-}rprt(s_p)$
    if $t_{sent} \leq time\text{-}sent(p)$ **then**
      $dist\text{-}rprt(s_p) := \langle time\text{-}sent(p), now \rangle$
  if $h \in dist\text{-}rprt?(p)$ **then**
    $\langle t_{sent}, t_{delayed} \rangle := dist\text{-}rprt(p, h)$
    $\langle t_{rprt}, t_{dist} \rangle := dist(s_p)$
    if $t_{rprt} \leq t_{sent}$ **then**
      $t'_{dist} := (now - t_{delayed} - t_{sent})/2$
      $dist(s_p) := \langle t_{sent}, t'_{dist} \rangle$
      $dist\text{-}buff \cup= \{s_p\}$
  foreach $\langle h'', i'' \rangle \in seqno\text{-}rprts(p)$ **do:**
    if $max\text{-}seqno(h'') < i''$ **then**
      $max\text{-}seqno(h'') := i''$
      $seqno\text{-}buff \cup= \{h''\}$

input `process-mpkt`$_h(p)$
where $type(p) \neq \texttt{SESS}$
eff **if** $status =$ `member` **then**
  $\langle s_p, i_p \rangle := id(p)$
  if $max\text{-}seqno(s_p) = \perp$
    $\wedge type(p) = \texttt{DATA}$
  then
    $max\text{-}seqno(s_p) := i_p$
  if $max\text{-}seqno(s_p) \neq \perp$
    $\wedge max\text{-}seqno(s_p) < i_p$
  then
    $max\text{-}seqno(s_p) := i_p$

output `rep-msend`$_h(p)$
pre $status = \texttt{member} \wedge now = rep\text{-}deadline$
  $\wedge p = comp\text{-}sess\text{-}pkt(h, now, dist\text{-}rprt, seqno)$
eff $rep\text{-}deadline := now + \texttt{SESS-PERIOD}$

output `rep-dist`$_h(h', d')$
choose $t' \in \mathbb{R}^{\geq 0}$
pre $status = \texttt{member} \wedge h' \in dist\text{-}buff \wedge \langle t', d' \rangle = dist(h')$
eff $dist\text{-}buff \setminus= \{h'\}$

output `rep-seqno`$_h(s, i)$
pre $status = \texttt{member} \wedge s \in seqno\text{-}buff \wedge i = max\text{-}seqno(s)$
eff $seqno\text{-}buff \setminus= \{s\}$

time-passage $\nu(t)$
pre $status = \texttt{crashed}$
  $\vee(dist\text{-}buff = \emptyset \wedge seqno\text{-}buff = \emptyset$
    $\wedge(rep\text{-}deadline = \perp \vee now + t \leq rep\text{-}deadline))$
eff $now := now + t$

Figure 19 The IPMCAST Automaton — Signature

Actions:

input
- `crash`$_h$, for $h \in H$
- `mjoin`$_h$, for $h \in H$
- `mleave`$_h$, for $h \in H$
- `msend`$_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$

internal
- `mgrbg-coll`(pkt), for $pkt \in P_{\text{IPMCAST}}$

output
- `mjoin-ack`$_h$, for $h \in H$
- `mleave-ack`$_h$, for $h \in H$
- `mrecv`$_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$
- `mdrop`(p, H_d), for $p \in P_{\text{IPMCAST-CLIENT}}, H_d \subseteq H$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

5.2.6 The IP Multicast Component — IPMCAST

In this section, we give an abstract specification of the IP multicast service; the IP primitive that provides best-effort point to multi-point communication. In order to simplify the presentation, we assume that only a single multicast group exists. Furthermore, we abstract away the specifics of the underlying protocols that collectively provide the IP multicast service. In our model, hosts join, leave, and send data packets to the IP multicast group by issuing join and leave requests and by multicasting data packets, respectively. Following the initial service model of IP multicast, a host need not be a member of the IP multicast group to send messages addressed to the group. However, a host must join the IP multicast group in order to receive packets addressed to the IP multicast group. The IP multicast service guarantees that only hosts who are members of the IP multicast group actually receive IP multicast packets.

Figures 19 and 20 present the signature, variables, and discrete transitions of the the IPMCAST timed I/O automaton; an abstract specification of the IP multicast service.

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of IPmcast. Each variable $status(h) \in IPmcast\text{-}Status$, for $h \in H$, denotes the IP multicast membership status of the host h. The value `idle` indicates that h is *idle* with respect to the IP multicast group; that is, it is neither a member, nor in the process of joining or leaving the IP multicast group. The value `joining` indicates that h is in the process of joining the IP multicast group; that is, the client has issued a request to join the IP multicast group and is awaiting an acknowledgment of this join request from the IP multicast service. The value `leaving` indicates that h is in the process of leaving the IP multicast group; that is, the client has issued a request to leave the IP multicast group and is awaiting an acknowledgment of this leave request from the IP multicast service. The value `member` indicates that h is a member of the IP multicast group. The value `crashed` indicates that h has crashed. When the host h has crashed, none of the input actions pertaining to h affect the state of IPmcast and none of the locally controlled actions pertaining to h are enabled. While the host h has not crashed, we say that it is *operational*.

The variable $mpkts \subseteq P_{\text{IPMCAST}}$ is comprised of the tuples that track the transmission progress of the packets transmitted during the particular execution of IPmcast. Of course, the size of the intended delivery set of each transmission progress tuple decreases monotonically as the hosts it consists of may leave the IP multicast group or crash.

Derived Variables The derived variable $up \subseteq H$ is the set of hosts that are operational; that is, the set of hosts that have not yet crashed. The derived variable $idle \subseteq H$ is a set of hosts that are idle with respect to the IP multicast group. The derived variable $joining \subseteq H$ is a set of hosts that are in the process of joining the IP multicast group. The derived variable $leaving \subseteq H$ is a set of hosts that are in the process of leaving the IP multicast group. The derived variable $members \subseteq H$ is a set of hosts that are members of the IP multicast group.

Input Actions Each input action $\texttt{crash}_h$, for $h \in H$, models the crashing of the host h. The $\texttt{crash}_h$ action records the fact that h has crashed by setting the $status(h)$ variable to `crashed`. Moreover, the $\texttt{crash}_h$ action removes the host h from the intended delivery set of any packet in the set of pending packets $mpkts$.

The input action $\texttt{mjoin}_h$ models the request of the client at h to join the IP multicast group. The $\texttt{mjoin}_h$ action is effective only while the host is idle with respect to the IP multicast group. When effective, the $\texttt{mjoin}_h$ action sets the $status(h)$ variable to `joining` so as to record that the host h has initiated the process of joining the IP multicast group. If the client is either a member of or in the process of joining the IP multicast group, then the $\texttt{mjoin}_h$ action is superfluous. If the client is already in the process of leaving the group, then the $\texttt{mjoin}_h$ action is discarded so as to allow the process of leaving the IP multicast group to complete.

The input action $\texttt{mleave}_h$ models the request of the client at h to leave the IP multicast group. The $\texttt{mleave}_h$ action is effective only while the host is either a member of or in the process of joining the IP multicast group. When effective, the $\texttt{mleave}_h$ action sets the $status(h)$ variable to `leaving` so as to record that the host h has initiated the process of leaving the IP multicast group. Moreover, the $\texttt{mleave}_h$ action removes the host h from the intended delivery set of any packet in the set of pending packets $mpkts$. Leave requests overrule join requests; that is, when a $\texttt{mleave}_h$ action is performed while the host h is in the process of joining the IP multicast group, its effects are to abort the process of joining and to initiate the process of leaving the IP multicast group. If the client is either idle with respect to or already in the process of leaving the IP multicast group, then the $\texttt{mleave}_h$ action is superfluous.

The input action $\texttt{msend}_h(p)$ models the transmission by the client at h of the packet p using the IP multicast service. The $\texttt{msend}_h(p)$ action is effective only if the client is operational; recall that a

Figure 20 The IPMCAST automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in \mathit{IPmcast\text{-}Status}$, for all $h \in H$,
initially $status(h) = \texttt{idle}$, for all $h \in H$
$mpkts \subseteq P_{\text{IPMCAST}}$, initially $mpkts = \emptyset$

Derived Variables:

$up = \{h \in H | status(h) \neq \texttt{crashed}\}$
$idle = \{h \in H | status(h) = \texttt{idle}\}$
$joining = \{h \in H | status(h) = \texttt{joining}\}$
$leaving = \{h \in H | status(h) = \texttt{leaving}\}$
$members = \{h \in H | status(h) = \texttt{member}\}$

Discrete Transitions:

input $\texttt{crash}_h$
eff **if** $h \in up$ **then**
$status(h) := \texttt{crashed}$
foreach $pkt \in mpkts$ **do:**
$intended(pkt) \setminus= \{h\}$

input $\texttt{mjoin}_h$
eff **if** $h \in idle$ **then**
$status(h) := \texttt{joining}$

input $\texttt{mleave}_h$
eff **if** $h \in joining \cup members$ **then**
$status(h) := \texttt{leaving}$
foreach $pkt \in mpkts$ **do:**
$intended(pkt) \setminus= \{h\}$

input $\texttt{msend}_h(p)$
eff **if** $h \in up$ **then**
$mpkts \cup= \{\langle p, members, \{h\}, \emptyset\rangle\}$

internal $\texttt{mgrbg-coll}(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
$\wedge intended(pkt) \subseteq (completed(pkt) \cup dropped(pkt))$
eff $mpkts \setminus= \{pkt\}$

output $\texttt{mjoin-ack}_h$
pre $h \in joining$
eff $status(h) := \texttt{member}$

output $\texttt{mleave-ack}_h$
pre $h \in leaving$
eff $status(h) := \texttt{idle}$

output $\texttt{mrecv}_h(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $h \in members \setminus dropped(pkt)$
$\wedge pkt \in mpkts \wedge p = strip(pkt)$
eff $completed(pkt) \cup= \{h\}$

output $\texttt{mdrop}(p, H_d)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
$\wedge H_d \subseteq members \setminus (completed(pkt) \cup dropped(pkt))$
eff $dropped(pkt) \cup= H_d$

time-passage $\nu(t)$
pre None
eff $now := now + t$

client need not be a member of the IP multicast group to multicast packets using the IP multicast service. The effects of the $\texttt{msend}_h(p)$ action are to add a tuple corresponding to the transmission of the packet p to $mpkts$. This tuple is initialized as follows: its intended delivery set is initialized to the current members of the IP multicast group, its completed delivery set is initialized to the host h as if the packet p has already been delivered to the client at the host h, and its dropped set is initialized to the empty set.

Output Actions The output action $\texttt{mjoin-ack}_h$ acknowledges the join request of the client at h. The $\texttt{mjoin-ack}_h$ action is enabled only when the host is in the process of joining the IP multicast group. Its effects are to set the $status(h)$ variable to $\texttt{member}$ so as to indicate that the client at h has become a member of the IP multicast group.

The output action $\texttt{mleave-ack}_h$ acknowledges the leave request of the client at h. The action $\texttt{mleave-ack}_h$ is enabled when the host is in the process of leaving the IP multicast group. Its effects are to set the $status(h)$ variable to $\texttt{idle}$ so as to indicate that the client at h has become idle with respect to the IP multicast group.

The output action $\texttt{mrecv}_h(p)$ models the delivery of the packet p to the client at h. The $\texttt{mrecv}_h(p)$ action is enabled when p is a pending packet, the host h is both a member of the IP multicast group and absent from the dropped set of the transmission progress tuple pkt in $mpkts$ pertaining to p. The effects of the $\texttt{mrecv}_h(p)$ action are to add the host h to the completed delivery set of p's transmission progress tuple pkt.

The output action $mdrop(p, H_d)$, for any $p \in P_{\text{IPMCAST-CLIENT}}$ and $H_d \subseteq H$, models the drop of the packet p on a link of the underlying IP multicast tree whose descendants are the hosts in the set H_d. The $mdrop(p, H_d)$ action is enabled when p is a pending packet and H_d is comprised of members of the IP multicast group for which the delivery of the packet p has neither completed, nor failed due to prior packet drops. The $mdrop(p, H_d)$ action adds the hosts comprising H_d to the dropped set of the transmission progress tuple *pkt* in *mpkts* pertaining to p.

Internal Actions The internal action $\texttt{mgrbg-coll}(p)$ models the garbage collection of the packet p. A packet p may only be garbage collected after all the hosts comprising its intended delivery set either receive the packet or suffer a loss that prevents the packet from being forwarded to them. The effects of the $\texttt{mgrbg-coll}(p)$ action are to remove the transmission progress tuple *pkt* pertaining to p from the set *mpkts*.

Time Passage The time-passage action $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$, models the passage of t time units. The action $\nu(t)$ is enabled at any point in time and increments the variable *now* by t time units.

Properties

Lemma 5.1 (Transmission Integrity) *For any timed trace β of* IPMCAST*, it is the case that any $\texttt{mrecv}_h(p)$ action, for $h \in H$, in β is preceded in β by a $\texttt{msend}_{h'}(p)$ action, for some $h' \in H$.*

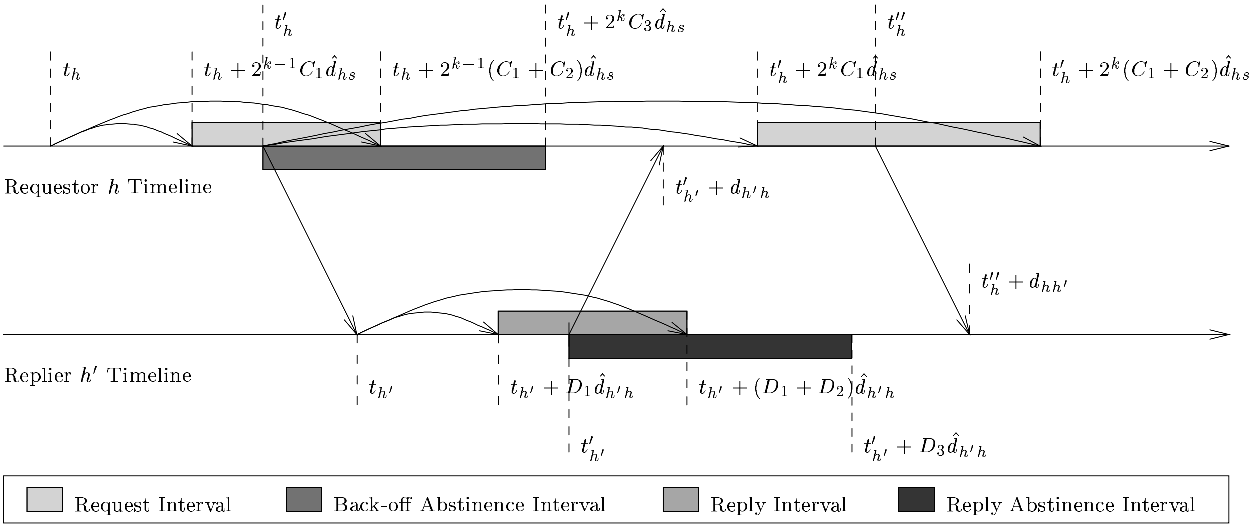
Proof: Let α be any timed execution of IPMCAST such that $\beta = ttrace(\alpha)$. Consider a particular occurrence of an action $\texttt{mrecv}_h(p)$ in α, for $h \in H$. Let $(u, \texttt{mrecv}_h(p), u') \in trans(\text{IPMCAST})$ be the discrete transition in α corresponding to the particular occurrence of the action $\texttt{mrecv}_h(p)$ in α. From the precondition of $\texttt{mrecv}_h(p)$, it is the case that there is a packet $pkt \in u.mpkts$, such that $p = strip(pkt)$. However, such a packet may be added to *mpkts* only by the occurrence of an action $\texttt{msend}_{h'}(p)$, for some $h \in H$. It follows that the occurrence of any action $\texttt{mrecv}_h(p)$ in α is preceded by the occurrence of an action $\texttt{rm-send}_{h'}(p)$, for some $h' \in H$. ■

5.3 Constraints on RMI's Parameters

Figure 21 illustrates the behavior of RMI's packet loss recovery scheme. In particular, for any $k \in \mathbb{N}^+$, it depicts the transmission of a k-th round request by h, the scheduling of a $k+1$-st round request by h, and the scheduling of a reply to h's k-th round request by a host h'. t_h is the point in time at which h schedules its k-th round request, t'_h is the point in time for which h schedules its k-th round request, $t_{h'}$ is the point in time h' receives h's k-th round request, and $t'_{h'}$ is the point in time for which h' schedules its reply to h's k-th round request. $\hat{d}_{hs}$ is half of h's RTT estimate to the source s of the packet being recovered, $d_{hh'}$ and $d_{h'h}$ are the actual transmission latencies between h and h', and $\hat{d}_{h'h}$ is half of the RTT estimate of h' to h.

RMI must ensure that the back-off abstinence intervals do not overlap with request intervals. From Figure 21, this requirement is enforced by imposing the parameter constraint $C_3 < C_1$. Moreover, RMI must ensure that requestors schedule their retransmission requests such that they succeed the reception of replies pertaining to prior recovery rounds. Prematurely transmitting requests would result in wasteful recovery traffic. From Figure 21, this requirement corresponds to the satisfaction of the inequalities $d_{hh'} + (D_1 + D_2)\hat{d}_{h'h} + d_{h'h} < 2^k C_1 \hat{d}_{hs}$, for $k \in \mathbb{N}^+$. Presuming that inter-host transmission latencies are fixed and symmetric and that RMI's inter-host RTT estimates are accurate, these inequalities are satisfied if $D_1 + D_2 + 2 < 2C_1$. Finally, RMI must also ensure that a particular round's requests are not discarded by potential repliers because they are received during the repliers' abstinence periods pertaining to the prior recovery round. From Figure 21, this requirement corresponds to the satisfaction of the inequalities $d_{hh'} + (D_1 + D_2)\hat{d}_{h'h} + D_3\hat{d}_{h'h} < 2^k C_1 \hat{d}_{hs} + d_{hh'}$, for $k \in \mathbb{N}^+$. Presuming that inter-host transmission

Figure 21 Timing Diagram of SRM's Loss Recovery Scheme



latencies are fixed and symmetric and that RMI's inter-host RTT estimates are accurate, these inequalities are satisfied if $D_1 + D_2 + D_3 < 2C_1$.

The following assumption summarizes the constraints on RMI's parameters.

Assumption 5.1 RM$_I$*'s parameters* C_1, C_2, C_3, D_1, D_2, *and* D_3 *satisfy the following constraints:* $C_3 < C_1$, $D_1 + D_2 + 2 < 2C_1$, *and* $D_1 + D_2 + D_3 < 2C_1$.

To our knowledge, these constraints on SRM's request/reply scheduling parameters, or even similar ones, have not been expressed to date. In fact, most analyses and simulations presume that no recovery packets are lost; that is, they presume that the initial recovery round is always successful. Our timing analysis illustrates that if the parameters are chosen arbitrarily it is possible to cause either superfluous requests and replies or the failure of a recovery round due to replier abstinence. Although in practice, due to inaccurate inter-host RTT estimates and varying and non-symmetric inter-host transmission latencies, superfluous traffic and/or recovery round failure may indeed be unavoidable, it is still important to realize their tie to SRM's parameters.

5.4 Safety and Liveness Analysis of RMI

We begin this section by defining some history variables that facilitate the proof that RMI implements RMS. We then define a relation between the states of RMI and RMS and prove that this relation is indeed a timed forward simulation relation. This proof establishes that RMI is safe with respect to RMS; that is, it may only deliver appropriate packets to each member of the reliable multicast group. We conclude by showing that, under certain constraints, RMI is live with respect to RMS; that is, under the given constraints, RMI guarantees the timely delivery of the appropriate packets to the appropriate members of the reliable multicast group, as formalized in Section 4.

5.4.1 History Variables

Figure 22 introduces history and derived history variables for the automata SRM-rec$_h$ and SRM, respectively.

The history variables of the SRM-rec$_h$ automata, for $h \in H$, are the variables *trans-time*(p), for all $p \in P_{\text{RM-Client}}[h]$, *expected*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, and *delivered*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$.

Figure 22 History and Derived History Variables

History Variables of SRM-REC_h**:**

$\textit{trans-time}(p) \in \mathbb{R}^{\geq 0} \cup \perp$, for all $p \in P_{\text{RM-CLIENT}}[h]$, initially $\textit{trans-time}(p) = \perp$, for all $p \in P_{\text{RM-CLIENT}}[h]$
$\textit{expected}(h') \subseteq H \times \mathbb{N}$, for all $h' \in H$, initially $\textit{expected}(h') = \emptyset$, for all $h' \in H$
$\textit{delivered}(h') \subseteq H \times \mathbb{N}$, for all $h' \in H$, initially $\textit{delivered}(h') = \emptyset$, for all $h' \in H$

Derived History Variables of SRM**:**

$\textit{sent-pkts} = \{p \in P_{\text{RM-CLIENT}} \mid \textit{trans-time}(p) \neq \perp\}$
$\textit{sent-pkts?} = \{\langle s, i\rangle \in H \times \mathbb{N} \mid \exists\, p \in \textit{sent-pkts} : id(p) = \langle s, i \rangle\}$
$\textit{intended}(p) = \{h \in H \mid id(p) \in \text{SRM-REC}_h.\textit{expected}(\textit{source}(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$\textit{completed}(p) = \{h \in H \mid id(p) \in \text{SRM-REC}_h.\textit{delivered}(\textit{source}(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$\textit{active-pkts} = \{p \in P_{\text{RM-CLIENT}} \mid p \in \textit{sent-pkts} \wedge \textit{intended}(p) \cap \textit{completed}(p) \neq \emptyset\}$

Figure 23 SRM-REC_h History Variable Assignments

```
input crash_h
eff ...
    foreach h' ∈ H do:
      expected(h') := ∅
      delivered(h') := ∅

input rm-leave_h
eff if status ≠ crashed then
      Reinitialize all variables except now.
      foreach h' ∈ H do:
        expected(h') := ∅
        delivered(h') := ∅

input rm-send_h(p)
eff ...
      \\ Record foremost DATA packet
      if min-seqno(s_p) =⊥ then
        ...
        expected(h) := suffix(p)
      ...
      if max-seqno(s_p) =⊥
        ∨i_p = max-seqno(s_p) + 1
      then
        ...
        trans-time(p) := now
        delivered(h) ∪= {id(p)}

output rm-recv_h(p)
pre ...
eff ...
    ⟨s_p, i_p⟩ := id(p)
    if expected(s_p) = ∅ then
      expected(s_p) := suffix(p)
    delivered(s_p) ∪= {id(p)}
```

Each $\textit{trans-time}(p)$ variable, for $p \in P_{\text{RM-CLIENT}}[h]$, records the transmission time of the packet p by the host h. Each $\textit{expected}(h')$ variable , for $h' \in H$, is comprised of the identifiers of the packets from h' that the host h expects to deliver since it last joined the reliable multicast group. Each $\textit{delivered}(h')$ variable, for $h' \in H$, is comprised of the identifiers of the packets from h' that the host h has already delivered since it last joined the reliable multicast group. Figure 23 specifies how the actions of SRM-REC_h affect these history variables.

The derived history variables of SRM are the set of identifiers of all packets sent since the beginning of the execution, *sent-pkts*, the intended delivery set of p, $\textit{intended}(p)$, for all $p \in P_{\text{RM-CLIENT}}$, the completed delivery set of p, $\textit{completed}(p)$, for all $p \in P_{\text{RM-CLIENT}}$, and the set of active packets, *active-pkts*.

5.4.2 Preliminary Invariants and Lemmas

In this section, we present several preliminary invariants and lemmas that are later used in the safety and liveness proofs of the RM_I automaton. We begin by presenting several invariants pertaining to the SRM-REC_h automaton, for $h \in H$.

Invariant 5.1 *For $h, h' \in H$ and any reachable state u of SRM-REC_h, if $u.status \neq$ `member`, then $u.\textit{expected}(h') = \emptyset$ and $u.\textit{delivered}(h') = \emptyset$.*

Proof: Let α be any finite timed execution of SRM-REC_h leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is,

$\alpha = u$. Since u is a start state of SRM-REC_h, it follows that $u.\mathit{status} = \texttt{idle}$, $u.\mathit{expected}(h') = \emptyset$, and $u.\mathit{delivered}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables *status*, $\mathit{expected}(h')$, and $\mathit{delivered}(h')$.

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets the variable *status* to `crashed` and the variables $\mathit{expected}(h')$ and $\mathit{delivered}(h')$ to $\emptyset$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-join-ack}_h$: if $u_k.\mathit{status} \neq \texttt{crashed}$, then the action $\texttt{rm-join-ack}_h$ sets the variable *status* to `member`. Thus, the invariant assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} = \texttt{crashed}$, then the action $\texttt{rm-join-ack}_h$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-leave}_h$: if $u_k.\mathit{status} \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variable *status* to `idle` and the $\mathit{expected}(h')$ and $\mathit{delivered}(h')$ variables to $\emptyset$. Thus, the invariant assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: first, consider the case where $\neg(u_k.\mathit{status} = \texttt{member} \wedge h = \mathit{source}(p))$. In this case, $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Second, consider the case where $u_k.\mathit{status} = \texttt{member}$ and $h = \mathit{source}(p)$. Since $u_k.\mathit{status} = \texttt{member}$ and the $\texttt{rm-send}_h(p)$ does not affect the *status* variable, it follows that $u.\mathit{status} = \texttt{member}$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: the precondition of the action $\texttt{rm-recv}_h(p)$ implies that $u_k.\mathit{status} = \texttt{member}$. Since the $\texttt{rm-recv}_h(p)$ does not affect the *status* variable, it follows that $u.\mathit{status} = \texttt{member}$. Thus, the invariant assertion is satisfied in u.

■

Invariant 5.2 *For $h, h' \in H$ and any reachable state u of* SRM-REC_h*, if $u.\mathit{min\text{-}seqno}(h') \neq \perp$, then $u.\mathit{min\text{-}seqno}(h') \leq u.\mathit{max\text{-}seqno}(h')$.*

Proof: Let α be any finite timed execution of SRM-REC_h leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC_h, it follows that $u.\mathit{min\text{-}seqno}(h) = \perp$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.\mathit{status} \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variables $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$ to $\perp$. Thus, the induction assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, consider the case where $\neg(u_k.\mathit{status} = \texttt{member} \wedge h = s_p)$. In this case, $\texttt{rm-send}_h(p)$ does not affect the variables $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\mathit{status} = \texttt{member}$ and $h = s_p$. Since $s_p = h'$, it follows that $h = h' = s_p$. If p is the foremost packet from s_p, that is, $u_k.\mathit{min\text{-}seqno}(s_p) = \perp$, then the $\texttt{rm-send}_h(p)$ action sets both *min-seqno*(h') and *max-seqno*(h') to i_p. It follows that $u.\textit{min-seqno}(h') \leq u.\textit{max-seqno}(h')$. Thus, the invariant assertion is satisfied in u.

If p is the next packet from s_p, then the action $\texttt{rm-send}_h(p)$ does not affect *min-seqno*(h') and sets *max-seqno*(h') to i_p; that is, $u.\textit{min-seqno}(h') = u_k.\textit{min-seqno}(h')$ and $u.\textit{max-seqno}(h') = u_k.\textit{max-seqno}(h') + 1$. Since $i_p = u_k.\textit{max-seqno}(h') + 1$, it follows that $u_k.\textit{max-seqno}(h') < u.\textit{max-seqno}(h')$. The induction hypothesis implies that $u_k.\textit{min-seqno}(h') \leq u_k.\textit{max-seqno}(h')$. Thus, since it follows that $u.\textit{min-seqno}(h') \leq u.\textit{max-seqno}(h')$, as needed.

If p is neither the foremost nor the next packet from s_p, then the action $\texttt{rm-send}_h(p)$ does not affect the variables *min-seqno*(h') and *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{rep-seqno}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: first, consider the case where $\neg(u_k.\textit{status} = \texttt{member} \wedge u_k.\textit{min-seqno}(s) \neq \perp \wedge u_k.\textit{max-seqno}(s) < i)$. In this case, the action $\texttt{rep-seqno}_h(s,i)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\textit{status} = \texttt{member} \wedge u_k.\textit{min-seqno}(s) \neq \perp \wedge u_k.\textit{max-seqno}(s) < i$. In this case, $\texttt{rep-seqno}_h(s,i)$ does not affect *min-seqno*(h') and sets *max-seqno*(h') to i; that is, $u.\textit{min-seqno}(h') = u_k.\textit{min-seqno}(h')$ and $u.\textit{max-seqno}(h') = i$. Since $u_k.\textit{max-seqno}(h') < i$ and $u.\textit{max-seqno}(h') = i$, it follows that $u_k.\textit{max-seqno}(h') < u.\textit{max-seqno}(h')$. From the induction hypothesis, it is the case that $u_k.\textit{min-seqno}(h') \leq u_k.\textit{max-seqno}(h')$. Thus, it follows that $u.\textit{min-seqno}(h') \leq u.\textit{max-seqno}(h')$, as needed.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $\textit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \textit{id}(p)$, we analyze the effects of $\texttt{process-mpkt}_h(p)$ by cases. First, if $u_k.\textit{status} \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\textit{status} = \texttt{member}$. If p is the foremost packet from s_p, that is, $\textit{type}(p) = \texttt{DATA}$, $h \neq s_p$, and $u_k.\textit{min-seqno}(s_p) = \perp$, then the action $\texttt{process-mpkt}_h(p)$ sets both *min-seqno*(h') and *max-seqno*(h') to i_p. It follows that $u.\textit{min-seqno}(h') \leq u.\textit{max-seqno}(h')$, as needed.

If p is not the foremost packet from s_p but is proper, that is, $u_k.\textit{min-seqno}(s_p) \neq \perp$ and $u_k.\textit{min-seqno}(s_p) \leq i_p$, then the action $\texttt{process-mpkt}_h(p)$ does not affect *min-seqno*(h') and may increase the value of *max-seqno*(h'). It follows that $u.\textit{min-seqno}(h') = u_k.\textit{min-seqno}(h')$ and $u_k.\textit{max-seqno}(h') \leq u.\textit{max-seqno}(h')$. From the induction hypothesis, it is the case that $u_k.\textit{min-seqno}(h') \leq u_k.\textit{max-seqno}(h')$. Thus, it follows that $u.\textit{min-seqno}(h') \leq u.\textit{max-seqno}(h')$, as needed.

Otherwise, if p is neither the foremost nor a proper packet from s_p, then $\texttt{process-mpkt}_h(p)$ does not affect the variables *min-seqno*(h') and *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.3 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, if $u.\textit{status} = \texttt{member}$, then it is the case that $u.\textit{archived-pkts?}(h') = u.\textit{delivered}(h') \cup u.\textit{to-be-delivered?}(h')$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\textit{status} = \texttt{idle}$. Thus, the invariant assertion holds in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u, we consider only the actions that affect the variables *archived-pkts*, *delivered*(h'), and *to-be-delivered?*(h').

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets the variable *status* to `crashed`. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variable *status* to `idle`. Thus, the invariant assertion holds in u.

Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. It follows that $u.status = \texttt{crashed}$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.status = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member} \wedge h = s_p$. If p is either the foremost or the next packet from h, then $\texttt{rm-send}_h(p)$ archives p and records it as having been delivered. Thus, the induction hypothesis and the fact that the packet p is both archived and recorded as having been delivered imply that the invariant assertion holds in u.

Otherwise, if p is neither the foremost nor the next packet from h, then the action $\texttt{rm-send}_h(p)$ does not affect the variables *archived-pkts?*(h'), *delivered*(h'), and *to-be-delivered?*(h'). Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: $\texttt{rm-send}_h(p)$ removes $id(p)$ from *to-be-delivered?*(h') and adds it to *delivered*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{process-mpkt}_h(p)$ by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. We begin by considering the case where $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$. In this case, consider the case where p is either the foremost or a proper packet from s_p and $h \neq s_p$. In this case, if p has not already been archived, then $\texttt{process-mpkt}_h(p)$ adds $id(p)$ to both *archived-pkts?*(h') and *to-be-delivered?*(h'). This fact and the induction hypothesis imply that the invariant assertion is satisfied in u. Otherwise, if p has already been archived, then $\texttt{process-mpkt}_h(p)$ adds $id(p)$ to *to-be-delivered?*(h') only. Since $id(p) \in u_k.archived\text{-}pkts?(h')$ and $\texttt{process-mpkt}_h(p)$ does not affect *archived-pkts*, it follows that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$ and, thus, $id(p) \in u.archived\text{-}pkts?(h')$. Moreover, since $\texttt{process-mpkt}_h(p)$ adds $id(p)$ to *to-be-delivered?*(h'), it follows that $u.to\text{-}be\text{-}delivered?(h') = u_k.to\text{-}be\text{-}delivered?(h') \cup \{id(p)\}$. From the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') = u_k.delivered(h') \cup u_k.to\text{-}be\text{-}delivered?(h')$. Since $\texttt{process-mpkt}_h(p)$ does not affect *delivered*(h'), it follows that the invariant assertion holds in u.

Otherwise, if either p is neither the foremost nor a proper packet from s_p or $h = s_p$, $\texttt{process-mpkt}_h(p)$ does not affect *archived-pkts?*(h'), *delivered*(h'), and *to-be-delivered?*(h'). Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

If $type(p) \in \{\texttt{RQST}, \texttt{SESS}\}$, then the action $\texttt{process-mpkt}_h(p)$ does not affect *archived-pkts?*(h'), *delivered*(h'), and *to-be-delivered?*(h'). Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

■

Invariant 5.4 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.min\text{-}seqno(h') = \perp$ and $u.archived\text{-}pkts?(h') = \emptyset$. Since $u.min\text{-}seqno(h') = \perp$, it is the case that $u.window?(h') = \emptyset$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $archived\text{-}pkts?(h')$.

- ❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. Thus, it is the case that $u.min\text{-}seqno(h') = \perp$ and $u.archived\text{-}pkts?(h') = \emptyset$. Since $u.min\text{-}seqno(h') = \perp$, it is the case that $u.window?(h') = \emptyset$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

  Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

- ❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, consider the case where $\neg(u_k.status = \texttt{member} \wedge h = s_p)$. In this case, $\texttt{rm-send}_h(p)$ does not affect the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.status = \texttt{member}$ and $h = s_p$. Since $s_p = h'$, it follows that $h = h' = s_p$. If p is the foremost packet from s_p, that is, $u_k.min\text{-}seqno(s_p) = \perp$, then the $\texttt{rm-send}_h(p)$ action sets both $min\text{-}seqno(s_p)$ and $max\text{-}seqno(s_p)$ to i_p and adds the element $\langle p, now\rangle$ to *archived-pkts*. Since $u_k.min\text{-}seqno(s_p) = \perp$, it is the case that $u_k.window?(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.archived\text{-}pkts?(h') = \emptyset$. It follows that $u.archived\text{-}pkts?(h') = \{id(p)\}$. Moreover, since $u.min\text{-}seqno(h') = u.max\text{-}seqno(h') = i_p$, it follows that $u_k.window?(h') = \{id(p)\}$. Thus, if follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

  If p is the next packet from s_p, that is, $u_k.min\text{-}seqno(s_p) \neq \perp$ and $i_p = u_k.max\text{-}seqno(s_p) + 1$, then $\texttt{rm-send}_h(p)$ sets $max\text{-}seqno(s_p)$ to i_p and adds the element $\langle p, now\rangle$ to *archived-pkts*. It follows that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h') \cup \{id(p)\}$ and $u.window?(h') = u_k.window?(h') \cup \{id(p)\}$. From the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') \subseteq u_k.window?(h')$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

- ❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $source(p) = h'$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of $\texttt{process-mpkt}_h(p)$ by cases.

  First, consider the case where p is the foremost packet from s_p; that is, $type(p) = \texttt{DATA}$, $h \neq s_p$, and $u_k.min\text{-}seqno(s_p) = \perp$. Since $u_k.min\text{-}seqno(s_p) = \perp$, it is the case that $u_k.window?(s_p) = \emptyset$. Thus, the induction hypothesis implies that $u_k.archived\text{-}pkts?(s_p) = \emptyset$. Since $\texttt{process-mpkt}_h(p)$ sets both variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$ to i_p and adds $\langle strip(p), now\rangle$ to *archived-pkts*, it follows that $u.archived\text{-}pkts?(h') = u.window?(s_p) = \{id(p)\}$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$.

  Second, consider the case where p is not the foremost packet from s_p but is proper; that is, $u_k.min\text{-}seqno(s_p) \neq \perp$ and $u_k.min\text{-}seqno(s_p) \leq i_p$. In this case, the $\texttt{process-mpkt}_h(p)$ action:

i) adds the element $\langle strip(p), now\rangle$ to *archived-pkts*, if $h \neq s_p \wedge \langle s_p, i_p\rangle \notin u_k.archived\text{-}pkts?$, and ii) sets $max\text{-}seqno(s_p)$ to i_p, if $u_k.max\text{-}seqno(s_p) < i_p$. It follows that $u.archived\text{-}pkts?(s_p) \subseteq u_k.archived\text{-}pkts?(s_p) \cup \{id(p)\}$ and $u_k.window?(s_p) \cup \{id(p)\} \subseteq u.window?(s_p)$. Moreover, from the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') \subseteq u_k.window?(h')$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

■

Invariant 5.5 *For $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and any reachable state u of* SRM-REC$_h$*, if $p \in u.to\text{-}be\text{-}delivered$, then $u.min\text{-}seqno(source(p)) \neq \perp$ and $u.min\text{-}seqno(source(p)) \leq seqno(p)$.*

Proof: From the effects of the $process\text{-}mpkt_h(p)$ action, for $h \in H$ and $p \in P_{\text{SRM}}$, such that $id(p) = \langle s_p, i_p\rangle$, it follows that a packet p may be added to *to-be-delivered* only if h is not the source of p and p is a proper packet; that is, $h \neq s_p$, $min\text{-}seqno(s_p) \neq \perp$, and $min\text{-}seqno(s_p) \leq i_p$. ■

Invariant 5.6 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that:*

1. $u.min\text{-}seqno(h') = \perp \Rightarrow u.expected(h') = \emptyset$,
2. $u.delivered(h') \subseteq u.expected(h')$,
3. $h = h' \wedge u.status \neq \texttt{crashed} \Rightarrow u.expected(h') = u.proper?(h')$, *and*
4. $u.expected(h') \neq \emptyset \Rightarrow u.expected(h') = u.proper?(h')$

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.min\text{-}seqno(h') = \perp$, $u.delivered(h') = \emptyset$, $u.expected(h') = \emptyset$, and $u.proper?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $min\text{-}seqno(h')$, $delivered(h')$, $expected(h')$, and $proper?(h')$.

❒ $\texttt{crash}_h$: the $\texttt{crash}_h$ action sets $delivered(h')$ and $expected(h')$ to $\emptyset$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now* and sets the variables $delivered(h')$ and $expected(h')$ to $\emptyset$. It follows that $u.min\text{-}seqno(h') = \perp$, $u.delivered(h') = \emptyset$, $u.expected(h') = \emptyset$, and $u.proper?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u.

  Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of $\texttt{rm-recv}_h(p)$ by cases. First, if $\neg(u_k.status = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Second, consider the case where $u_k.status = \texttt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by s_p; that is, $u_k.min\text{-}seqno(s_p) = \perp$, then $\texttt{rm-send}_h(p)$ sets $min\text{-}seqno(h')$ to i_p, sets $expected(h')$ to $suffix(p)$, and adds $id(p)$ to $delivered(h')$. The induction hypothesis and the fact that $u_k.min\text{-}seqno(s_p) = \perp$ imply that $u_k.expected(s_p) = \emptyset$. Moreover, from the induction hypothesis it is the case that $u_k.\mathit{delivered}(s_p) \subseteq u_k.\mathit{expected}(s_p)$. Since $u_k.\mathit{expected}(s_p) = \emptyset$, it follows that $u_k.\mathit{delivered}(s_p) = \emptyset$. Thus, from the effects of $\texttt{rm-send}_h(p)$, it follows that $u.\mathit{expected}(s_p) = \mathit{suffix}(p)$ and $u.\mathit{delivered}(s_p) = \{\mathit{id}(p)\}$. Since $\mathit{id}(p) \in \mathit{suffix}(p)$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$. Moreover, since $u.\mathit{proper?}(h') = \mathit{suffix}(p)$, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$. Since $u.\mathit{min\text{-}seqno}(s_p) = i_p$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

If p is the next packet from s_p, that is, $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$ and $i_p = u_k.\mathit{max\text{-}seqno}(s_p) + 1$, then $\texttt{rm-send}_h(p)$ does not affect $\mathit{min\text{-}seqno}(h')$, sets $\mathit{max\text{-}seqno}(h')$ to i_p, and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{min\text{-}seqno}(s_p) = u_k.\mathit{min\text{-}seqno}(s_p)$, $u.\mathit{max\text{-}seqno}(s_p) = i_p$, and $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$.

Since $h = h' \wedge u_k.\mathit{status} \neq \texttt{crashed}$, the induction hypothesis implies that $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$. Since $\texttt{rm-send}_h(p)$ affects neither $\mathit{min\text{-}seqno}(h')$ nor $\mathit{expected}(h')$, it follows that $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$ and $u.\mathit{expected}(h') = u_k.\mathit{expected}(h')$. Thus, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

From the induction hypothesis, it is the case that $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$. Since $i_p = u_k.\mathit{max\text{-}seqno}(s_p) + 1$ and $u.\mathit{max\text{-}seqno}(s_p) = i_p$, it is the case that $u_k.\mathit{max\text{-}seqno}(s_p) < u.\mathit{max\text{-}seqno}(s_p)$. Thus, Invariant 5.2 implies that $u_k.\mathit{min\text{-}seqno}(s_p) < i_p$. Since $u_k.\mathit{min\text{-}seqno}(s_p) < i_p$, it follows that $\mathit{id}(p) \in u_k.\mathit{proper?}(h')$. Since $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$, it follows that $\mathit{id}(p) \in u_k.\mathit{expected}(h')$. Since $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$, $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$, $\mathit{id}(p) \in u_k.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u_k.\mathit{expected}(h')$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$. Since $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{rm-recv}_h(p)$ by cases. First, consider the case where $u_k.\mathit{expected}(h') = \emptyset$. From the induction hypothesis, it is the case that $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$. Thus, it follows that $u_k.\mathit{delivered}(h') = \emptyset$. Since $u_k.\mathit{expected}(h') = \emptyset$, $\texttt{rm-recv}_h(p)$ sets $\mathit{expected}(h')$ to $\mathit{suffix}(p)$ and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{expected}(s_p) = \mathit{suffix}(p)$ and $u.\mathit{delivered}(s_p) = \{\mathit{id}(p)\}$. Since $\mathit{id}(p) \in \mathit{suffix}(p)$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, as needed.

Since $u_k.\mathit{delivered}(h') = \emptyset$, Invariant 5.3 implies that $u_k.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{to\text{-}be\text{-}delivered?}(h')$. From the precondition of $\texttt{rm-recv}_h(p)$, it follows that p is h's foremost packet from h'; that is, $i_p = u_k.\mathit{min\text{-}seqno}(h')$. Since $\mathit{suffix}(p) = \{\langle s, i \rangle \in H \times \mathbb{N} \mid s_p = s \wedge i_p \leq i\}$, it follows that $u.\mathit{proper?}(h') = \mathit{suffix}(p)$. Thus, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

Finally, since $p \in u_k.\mathit{to\text{-}be\text{-}delivered}$, Invariant 5.5 implies that $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since $\texttt{rm-recv}_h(p)$ does not affect $\mathit{min\text{-}seqno}(s_p)$, it follows that $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.\mathit{expected}(h') \neq \emptyset$. In this case, $\texttt{rm-recv}_h(p)$ does not affect $\mathit{min\text{-}seqno}(s_p)$, does not affect $\mathit{expected}(h')$, and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$, $u.\mathit{expected}(s_p) = u_k.\mathit{expected}(s_p)$, and $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$. Since $u_k.\mathit{expected}(h') \neq \emptyset$, the induction hypothesis implies that $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$. Since $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$, $u.\mathit{expected}(s_p) = u_k.\mathit{expected}(s_p)$, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

Since $p \in u_k.\mathit{to\text{-}be\text{-}delivered}$, Invariant 5.3 implies that $\mathit{id}(p) \in u_k.\mathit{archived\text{-}pkts?}(h')$. Thus, Invariant 5.4 implies that $\mathit{id}(p) \in u_k.\mathit{window?}(h')$. By definition it follows that $\mathit{window?}(h') \subseteq \mathit{proper?}(h')$. Thus, it is the case that $\mathit{id}(p) \in u_k.\mathit{proper?}(h')$ and, since $u.\mathit{proper?}(h') =$

$u_k.proper?(h')$, $id(p) \in u.proper?(h')$. Thus, it follows that $u.delivered(s_p) \subseteq u.expected(s_p)$, as needed.

Finally, since $p \in u_k.to\text{-}be\text{-}delivered$, Invariant 5.5 implies that $u_k.min\text{-}seqno(s_p) \neq \perp$. Since $\texttt{rm-recv}_h(p)$ does not affect $min\text{-}seqno(s_p)$, it follows that $u.min\text{-}seqno(s_p) \neq \perp$. Since it is the case that $u.min\text{-}seqno(s_p) \neq \perp$, $u.delivered(h') \subseteq u.expected(h')$, and $u.expected(h') = u.proper?(h')$, it follows that the invariant assertion is satisfied in u.

- ❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{process-mpkt}_h(p)$ by cases.

  First, if $type(p) = \texttt{DATA}$, $u_k.status = \texttt{member}$, $h \neq s_p$, and $u_k.min\text{-}seqno(h') = \perp$, then the action $\texttt{process-mpkt}_h(p)$ sets $min\text{-}seqno(h')$ to i_p and affects neither $delivered(h')$ nor $expected(h')$. Since $u_k.min\text{-}seqno(h') = \perp$, the induction hypothesis implies that $u_k.expected(h') = \emptyset$. Moreover, from the induction hypothesis, it is the case that $u_k.delivered(h') \subseteq u_k.expected(h')$. Thus, since $u_k.expected(h') = \emptyset$, it follows that $u_k.delivered(h') = \emptyset$. Since $\texttt{process-mpkt}_h(p)$ affects neither $delivered(h')$ nor $expected(h')$, it follows that $u.delivered(h') = \emptyset$ and $u.expected(h') = \emptyset$. Thus, it follows that $u.delivered(h') \subseteq u.expected(h')$, as needed. Since $h \neq s_p$ and $s_p = h'$, it follows that $h \neq h'$. Thus, since $u.min\text{-}seqno(h') \neq \perp$, $u.delivered(h') \subseteq u.expected(h')$, $h \neq h'$, $u.expected(h') = \emptyset$, it follows that the invariant assertion is satisfied in u.

  Otherwise, $\texttt{process-mpkt}_h(p)$ does not affect $min\text{-}seqno(h')$, $delivered(h')$, and $expected(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.7 *Let $h \in H$ and u be any reachable state u of* SRM-REC$_h$. *For any $p \in P_{\text{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $p \in u.msend\text{-}buff$, it is the case that $id(p) \in u.archived\text{-}pkts?$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.msend\text{-}buff = \emptyset$. Thus, the invariant assertion is trivially satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables *msend-buff* and *archived-pkts*.

- ❒ $\texttt{rm-leave}_h$: the action $\texttt{rm-leave}_h$ initializes the variables *msend-buff* and *archived-pkts*. Thus, the invariant assertion holds in u.
- ❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: the action $\texttt{rm-send}_h(p)$ adds the packet $comp\text{-}data\text{-}pkt(p)$ to *msend-buff* if and only if it adds the element $\langle p, now \rangle$ to the variable *archived-pkts*. This fact and the induction hypothesis imply that the invariant assertion holds in u.
- ❒ $\texttt{send-repl}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$: the action $\texttt{send-repl}_h(s,i)$ adds the packet $pkt = comp\text{-}repl\text{-}pkt(h,p)$, for $p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0}$, such that $\langle p, t \rangle \in archived\text{-}pkts$, and $id(p) = \langle s, i \rangle$ to *msend-buff*. Since $id(pkt) \in u_k.archived\text{-}pkts?$ and the $\texttt{send-repl}_h(s,i)$ action does not affect the variable *archived-pkts*, it follows that $id(pkt) \in u.archived\text{-}pkts?$. The induction hypothesis and the facts that $pkt \in u.msend\text{-}buff$ and $id(pkt) \in u.archived\text{-}pkts?$ imply that the invariant assertion is satisfied in u.
- ❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $source(p) = h'$: $\texttt{process-mpkt}_h(p)$ does not affect *msend-buff* and may only add the element $id(p)$ to *archived-pkts?*. Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.8 *For $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and any reachable state u of* SRM-REC$_h$, *if $p \in$ $u.\textit{to-be-delivered}$, then $\textit{source}(p) \neq h$.*

Proof: From the effects of the $\textit{process-mpkt}_h(p)$ action, for $h \in H$ and $p \in P_{\text{SRM}}$, it follows that a packet p may be added to *to-be-delivered* only if $\textit{source}(p) \neq h$. ■

Invariant 5.9 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$, *if $u.\textit{expected}(h') \neq \emptyset$, then $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{expected}(h')$.*

Proof: Suppose that $u.\textit{expected}(h') \neq \emptyset$. Invariant 5.1 implies that $u.\textit{status} = \texttt{member}$. Moreover, Invariant 5.6 implies that $u.\textit{expected}(h') = u.\textit{proper?}(h')$. From Invariant 5.4, it is the case that $u.\textit{archived-pkts?}(h') \subseteq u.\textit{window?}(h')$. Moreover, since $u.\textit{status} = \texttt{member}$, Invariant 5.3 implies that $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{window?}(h')$. Since by definition $u.\textit{window?}(h') \subseteq u.\textit{proper?}(h')$, it follows that $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{proper?}(h')$. Finally, since $u.\textit{expected}(h') = u.\textit{proper?}(h')$, it follows that $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{expected}(h')$. ■

Invariant 5.10 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$, *it is the case that $u.\textit{to-be-requested}(h') \subseteq u.\textit{window?}(h')$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\textit{min-seqno}(h') = \perp$ and $u.\textit{to-be-requested}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\textit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\textit{min-seqno}(h')$, $\textit{max-seqno}(h')$, and $\textit{to-be-requested}(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.\textit{status} = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\textit{status} \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. It follows that $u.\textit{min-seqno}(h') = \perp$ and $u.\textit{to-be-requested}(h') = \emptyset$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\textit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \textit{id}(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.\textit{status} = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\textit{status} = \texttt{member} \wedge h = s_p$. If $u_k.\textit{min-seqno}(h') = \perp$, then $\texttt{rm-send}_h(p)$ sets $\textit{min-seqno}(h')$ and $\textit{max-seqno}(h')$ to i_p. Since $u_k.\textit{min-seqno}(h') = \perp$, it follows that $u_k.\textit{window?}(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.\textit{to-be-requested}(h') = \emptyset$. Since $\texttt{rm-send}_h(p)$ does not affect the variable *to-be-requested*, it follows that $u.\textit{to-be-requested}(h') = \emptyset$. Thus, the invariant assertion holds in u.

  Otherwise, if $u_k.\textit{min-seqno}(h') \neq \perp$, then $\texttt{rm-send}_h(p)$ may only increase the value of the variable $\textit{max-seqno}(h')$ and does not affect the variable *to-be-requested*; that is, $u_k.\textit{window?}(h') \subseteq u.\textit{window?}(h')$ and $u.\textit{to-be-requested}(h') = u_k.\textit{to-be-requested}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{rep-seqno}_h(s, i)$, for $s \in H$, $s \neq h$ and $i \in \mathbb{N}$, such that $s = h'$: first, if $\neg(u_k.\textit{status} = \texttt{member} \wedge u_k.\textit{min-seqno}(s) \neq \perp \wedge u_k.\textit{max-seqno}(s) < i)$, then $\texttt{rep-seqno}_h(s, i)$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Otherwise, if $u_k.\mathit{status} = \texttt{member}$, $u_k.\mathit{min\text{-}seqno}(s) \neq \perp$, and $u_k.\mathit{max\text{-}seqno}(s) < i$, then the action $\texttt{rep-seqno}_h(s,i)$ adds $\{\langle s,i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s) < i' \leq i\}$ to *to-be-requested* and sets $\mathit{max\text{-}seqno}(s)$ to i. Invariant 5.2 and the fact that $u_k.\mathit{max\text{-}seqno}(s) < i$ imply that $u_k.\mathit{min\text{-}seqno}(s) < i$. Since $\texttt{rep-seqno}_h(s,i)$ does not affect the variable $\mathit{min\text{-}seqno}(s)$, it follows that $u.\mathit{min\text{-}seqno}(s) < i$. Thus, since $u.\mathit{min\text{-}seqno}(s) < i$ and $u.\mathit{max\text{-}seqno}(s) = i$, it follows that $\{\langle s,i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s) < i' \leq i\} \subseteq u.\mathit{window}(h')$. This fact and the induction hypothesis imply that $u.\mathit{to\text{-}be\text{-}requested}(h') \subseteq u.\mathit{window?}(h')$.

❒ $\texttt{schdl-rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: the action $\texttt{schdl-rqst}_h(s,i)$ removes the element $\langle s,i\rangle$ from the set $u_k.\mathit{to\text{-}be\text{-}requested}$ and does not affect $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $\mathit{type}(p) = \texttt{DATA}$ and $\mathit{source}(p) = h'$: letting $\langle s_p, i_p\rangle = \mathit{id}(p)$, we analyze the effects of the $\texttt{process-mpkt}_h(p)$ action by cases. First, if $u_k.\mathit{status} \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\mathit{status} = \texttt{member}$. If $h \neq s_p$ and $u_k.\mathit{min\text{-}seqno}(s_p) = \perp$, then $\texttt{process-mpkt}_h(p)$ sets the variables $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$ to i_p and does not affect the variable *to-be-requested*. Since $u_k.\mathit{min\text{-}seqno}(h') = \perp$, it follows that $u_k.\mathit{window?}(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.\mathit{to\text{-}be\text{-}requested}(h') = \emptyset$. Since $\texttt{process-mpkt}_h(p)$ does not affect the variable *to-be-requested*, it follows that $u.\mathit{to\text{-}be\text{-}requested}(h') = \emptyset$. Thus, the invariant assertion holds in u.

If $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u_k.\mathit{min\text{-}seqno}(s_p) \leq i_p$, $h \neq s_p$, and $u_k.\mathit{max\text{-}seqno}(s_p) < i_p$, then the action $\texttt{process-mpkt}_h(p)$ adds $\{\langle s_p,i\rangle \mid i \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i < i_p\}$ to *to-be-requested* and sets $\mathit{max\text{-}seqno}(h')$ to i_p. Since $u_k.\mathit{min\text{-}seqno}(h') \leq i_p$ and $\texttt{process-mpkt}_h(p)$ does not affect the variable $\mathit{min\text{-}seqno}(h')$, it follows that $u.\mathit{min\text{-}seqno}(h') \leq i_p$. Since $u.\mathit{min\text{-}seqno}(h') \leq i$ and $u.\mathit{max\text{-}seqno}(h') = i$, it follows that $\{\langle s_p,i\rangle \mid i \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i < i_p\} \subseteq u.\mathit{window}(h')$. This fact and the induction hypothesis imply that $u.\mathit{to\text{-}be\text{-}requested}(h') \subseteq u.\mathit{window?}(h')$.

Otherwise, $\texttt{process-mpkt}_h(p)$ does not affect the variables $\mathit{min\text{-}seqno}(h')$, $\mathit{max\text{-}seqno}(h')$, and $\mathit{to\text{-}be\text{-}requested}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $\mathit{type}(p) \in \{\texttt{REPL}, \texttt{RQST}\}$ and $\mathit{source}(p) = h'$: letting $\langle s_p, i_p\rangle = \mathit{id}(p)$, we analyze the effects of the $\texttt{process-mpkt}_h(p)$ action by cases. First, if $u_k.\mathit{status} \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC_h. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\mathit{status} = \texttt{member}$. If it is the case that $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u_k.\mathit{min\text{-}seqno}(s_p) \leq i_p$, $h \neq s_p$, and $u_k.\mathit{max\text{-}seqno}(s_p) < i_p$, then the action $\texttt{process-mpkt}_h(p)$ adds $\{\langle s_p,i\rangle \mid i \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i < i_p\}$ to *to-be-requested* and sets $\mathit{max\text{-}seqno}(h')$ to i_p. Since $u_k.\mathit{min\text{-}seqno}(h') \leq i_p$ and $\texttt{process-mpkt}_h(p)$ does not affect the variable $\mathit{min\text{-}seqno}(h')$, it follows that $u.\mathit{min\text{-}seqno}(h') \leq i_p$. Thus, since $u.\mathit{min\text{-}seqno}(h') \leq i$ and $u.\mathit{max\text{-}seqno}(h') = i$, it follows that $\{\langle s_p,i\rangle \mid i \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i < i_p\} \subseteq u.\mathit{window}(h')$. This fact and the induction hypothesis imply that $u.\mathit{to\text{-}be\text{-}requested}(h') \subseteq u.\mathit{window?}(h')$.

Otherwise, $\texttt{process-mpkt}_h(p)$ does not affect the variables $\mathit{min\text{-}seqno}(h')$, $\mathit{max\text{-}seqno}(h')$, and $\mathit{to\text{-}be\text{-}requested}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.11 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.\mathit{scheduled\text{-}rqsts?}(h') \subseteq u.\mathit{window?}(h')$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\mathit{min\text{-}seqno}(h') = \perp$ and $u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\mathit{min\text{-}seqno}(h')$, $\mathit{max\text{-}seqno}(h')$, and $\mathit{scheduled\text{-}rqsts?}(h')$.

- ❒ $\mathtt{rm\text{-}leave}_h$: if $u_k.\mathit{status} = \mathtt{crashed}$, then $\mathtt{rm\text{-}leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} \neq \mathtt{crashed}$, then $\mathtt{rm\text{-}leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. It follows that $u.\mathit{min\text{-}seqno}(h') = \perp$ and $u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u.

- ❒ $\mathtt{rm\text{-}send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\mathtt{rm\text{-}send}_h(p)$ by cases. First, if $\neg(u_k.\mathit{status} = \mathtt{member} \wedge h = s_p)$, then $\mathtt{rm\text{-}send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\mathit{status} = \mathtt{member} \wedge h = s_p$. If $u_k.\mathit{min\text{-}seqno}(h') = \perp$, then $\mathtt{rm\text{-}send}_h(p)$ sets $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$ to i_p. Since $u_k.\mathit{min\text{-}seqno}(h') = \perp$, it follows that $u_k.\mathit{window?}(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$. Since $\mathtt{rm\text{-}send}_h(p)$ does not affect the variable $\mathit{scheduled\text{-}rqsts}$, it follows that $u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

  Otherwise, if $u_k.\mathit{min\text{-}seqno}(h') \neq \perp$, then $\mathtt{rm\text{-}send}_h(p)$ may only increase the value of the variable $\mathit{max\text{-}seqno}(h')$ and does not affect the variable $\mathit{scheduled\text{-}rqsts}$; that is, $u_k.\mathit{window?}(h') \subseteq u.\mathit{window?}(h')$ and $u.\mathit{scheduled\text{-}rqsts}(h') = u_k.\mathit{scheduled\text{-}rqsts}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

- ❒ $\mathtt{rep\text{-}seqno}_h(s,i)$, for $s \in H$, $s \neq h$ and $i \in \mathbb{N}$, such that $s = h'$: first, if $\neg(u_k.\mathit{status} = \mathtt{member} \wedge u_k.\mathit{min\text{-}seqno}(s) \neq \perp \wedge u_k.\mathit{max\text{-}seqno}(s) < i)$, then $\mathtt{rep\text{-}seqno}_h(s,i)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Otherwise, if $u_k.\mathit{status} = \mathtt{member}$, $u_k.\mathit{min\text{-}seqno}(s) \neq \perp$, and $u_k.\mathit{max\text{-}seqno}(s) < i$, then the action $\mathtt{rep\text{-}seqno}_h(s,i)$ sets $\mathit{max\text{-}seqno}(h')$ to i. Since $u_k.\mathit{max\text{-}seqno}(h') < i$ and $u.\mathit{max\text{-}seqno}(h') = i$, it follows that $u_k.\mathit{max\text{-}seqno}(h') < u.\mathit{max\text{-}seqno}(h')$. The induction hypothesis and the fact that $u_k.\mathit{max\text{-}seqno}(h') < u.\mathit{max\text{-}seqno}(h')$ imply that the invariant assertion holds in u.

- ❒ $\mathtt{schdl\text{-}rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: $\mathtt{schdl\text{-}rqst}_h(s,i)$ adds the tuple $\langle s,i \rangle$ to $\mathit{scheduled\text{-}rqsts?}(h')$. From the precondition of $\mathtt{schdl\text{-}rqst}_h(s,i)$, it follows that $\langle s,i \rangle \in u_k.\mathit{to\text{-}be\text{-}requested}(h')$. Thus, Invariant 5.10 implies that $\langle s,i \rangle \in u_k.\mathit{window?}(h')$. Since $\mathtt{schdl\text{-}rqst}_h(s,i)$ does not affect the variables $\mathit{min\text{-}seqno}(h')$ and $\mathit{max\text{-}seqno}(h')$, it follows that $u.\mathit{window?}(h') = u_k.\mathit{window?}(h')$. From the induction hypothesis, it is the case that $u_k.\mathit{scheduled\text{-}rqsts?}(h') \subseteq u_k.\mathit{window?}(h')$. Since $u.\mathit{window?}(h') = u_k.\mathit{window?}(h')$ and $u.\mathit{scheduled\text{-}rqsts?}(h') = u_k.\mathit{scheduled\text{-}rqsts?}(h') \cup \langle s,i \rangle$, it follows that the invariant assertion hold in u.

- ❒ $\mathtt{send\text{-}rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: from the precondition of the action $\mathtt{send\text{-}rqst}_h(s,i)$, it is the case that $\langle s,i \rangle \in u_k.\mathit{scheduled\text{-}rqsts?}(h')$. Since $\mathtt{send\text{-}rqst}_h(s,i)$ simply backs-off the request scheduled for $\langle s,i \rangle$, it does not affect $\mathit{min\text{-}seqno}(h')$, $\mathit{max\text{-}seqno}(h')$, and *scheduled-rqsts?*$(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \texttt{DATA}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\texttt{process-mpkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$. If p is neither the foremost nor a proper packet from s_p, then $\texttt{process-mpkt}_h(p)$ affects neither of the variables *min-seqno*(h'), *max-seqno*(h'), and *scheduled-rqsts?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

If p is the foremost packet from s_p, then $\texttt{process-mpkt}_h(p)$ sets the variables *min-seqno*(h') and *max-seqno*(h') to i_p. From the induction hypothesis, it follows that $u_k.scheduled\text{-}rqsts?(h') = \emptyset$. Since $\texttt{process-mpkt}_h(p)$ may only remove elements from *scheduled-rqsts?*(h'), it follows that $u.scheduled\text{-}rqsts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

Finally, if $u_k.min\text{-}seqno(s_p) \neq \perp$, then $\texttt{process-mpkt}_h(p)$ may only remove elements from the set *scheduled-rqsts?*(h') and increase the value of *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \texttt{REPL}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\texttt{process-mpkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$. If p is not a proper packet, then the action $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

If p is a proper packet, then $\texttt{process-mpkt}_h(p)$ may only remove elements from the variable *scheduled-rqsts?*(h') and increase the value of *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \texttt{RQST}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\texttt{process-mpkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$. If p does not pertain to a proper packet, then the action $\texttt{process-mpkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, in this case, the induction hypothesis implies that the invariant assertion holds in u.

If p pertains to a proper packet and h is not the source of p, then $\texttt{process-mpkt}_h(p)$ may add the tuple $id(p)$ to *scheduled-rqsts?*(h') and ensures that $i_p \leq u.max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.12 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$, *it is the case that $u.to\text{-}be\text{-}requested(h') \cap u.archived\text{-}pkts?(h') = \emptyset$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.to\text{-}be\text{-}requested(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\mathit{to\text{-}be\text{-}requested}(h')$ and $\mathit{archived\text{-}pkts?}(h')$.

- ❒ $\mathtt{rm\text{-}leave}_h$: if $u_k.\mathit{status} = \mathtt{crashed}$, then $\mathtt{rm\text{-}leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} \neq \mathtt{crashed}$, then $\mathtt{rm\text{-}leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. It follows that $u.\mathit{to\text{-}be\text{-}requested}(h') = \emptyset$ and $u.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

- ❒ $\mathtt{rm\text{-}send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p\rangle = \mathit{id}(p)$, we analyze the effects of $\mathtt{rm\text{-}send}_h(p)$ by cases. First, if $\neg(u_k.\mathit{status} = \mathtt{member} \wedge h = s_p)$, then $\mathtt{rm\text{-}send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\mathit{status} = \mathtt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by h', that is, $u_k.\mathit{min\text{-}seqno}(h') =\perp$, then it follows that $u_k.\mathit{window?}(h') = \emptyset$. Thus, Invariants 5.4 and 5.10 imply that $u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$ and $u_k.\mathit{to\text{-}be\text{-}requested}(h') = \emptyset$. If p is the next packet from h', that is, $u_k.\mathit{min\text{-}seqno}(h') \neq\perp$ and $i_p = u_k.\mathit{max\text{-}seqno}(h') + 1$, then it is the case that $\mathit{id}(p) \notin u_k.\mathit{window?}(h')$. Thus, Invariants 5.4 and 5.10 imply that $\mathit{id}(p) \notin u_k.\mathit{archived\text{-}pkts?}(h')$ and $\mathit{id}(p) \notin u_k.\mathit{to\text{-}be\text{-}requested}(h')$.

  In either case the $\mathtt{rm\text{-}send}_h(p)$ adds $\mathit{id}(p)$ to the variable $\mathit{archived\text{-}pkts?}(h')$ and does not affect $\mathit{to\text{-}be\text{-}requested}(h')$. It follows that $u.\mathit{to\text{-}be\text{-}requested}(h') = u_k.\mathit{to\text{-}be\text{-}requested}(h')$ and $u.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{archived\text{-}pkts?}(h') \cup \mathit{id}(p)$. From the induction hypothesis, it is the case that $u_k.\mathit{to\text{-}be\text{-}requested}(h') \cap u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Since it is the case that $\mathit{id}(p) \notin u_k.\mathit{to\text{-}be\text{-}requested}(h')$, it follows that $u.\mathit{to\text{-}be\text{-}requested}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$.

- ❒ $\mathtt{rep\text{-}seqno}_h(s,i)$, for $s \in H, s \neq h$ and $i \in \mathbb{N}$, such that $s = h'$: first, if $\neg(u_k.\mathit{status} = \mathtt{member} \wedge u_k.\mathit{min\text{-}seqno}(s) \neq\perp \wedge u_k.\mathit{max\text{-}seqno}(s) < i)$, then $\mathtt{rep\text{-}seqno}_h(s,i)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Otherwise, if $u_k.\mathit{status} = \mathtt{member}$, $u_k.\mathit{min\text{-}seqno}(s) \neq\perp$, and $u_k.\mathit{max\text{-}seqno}(s) < i$, then the action $\mathtt{rep\text{-}seqno}_h(s,i)$ adds $\{\langle s, i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s) < i' \leq i\}$ to $\mathit{to\text{-}be\text{-}requested}(h')$ and does not affect $\mathit{archived\text{-}pkts?}(h')$. From Invariant 5.4, it is the case that $u_k.\mathit{archived\text{-}pkts?}(h') \subseteq u_k.\mathit{window?}(h')$. Thus, it follows that $u_k.\mathit{archived\text{-}pkts?}(h') \cap \{\langle s, i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s) < i' \leq i\} = \emptyset$. From the induction hypothesis, it is the case that $u_k.\mathit{to\text{-}be\text{-}requested}(h') \cap u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, it follows that $u.\mathit{to\text{-}be\text{-}requested}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$, as needed.

- ❒ $\mathtt{schdl\text{-}rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: the $\mathtt{schdl\text{-}rqst}_h(s,i)$ action removes the element $\langle s, i\rangle$ from $\mathit{to\text{-}be\text{-}requested}(h')$ and does not affect $\mathit{archived\text{-}pkts?}(h')$. From the induction hypothesis, it is the case that $u_k.\mathit{to\text{-}be\text{-}requested}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, it follows that $u.\mathit{to\text{-}be\text{-}requested}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$, as needed.

- ❒ $\mathtt{process\text{-}mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\mathit{type}(p) \in \{\mathtt{DATA}, \mathtt{REPL}, \mathtt{RQST}\}$, $\langle s_p, i_p\rangle = \mathit{id}(p)$, and $s_p = h'$: the action $\mathtt{process\text{-}mpkt}_h(p)$ adds $\{\langle s_p, i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i' < i\}$ to $\mathit{to\text{-}be\text{-}requested}(h')$ only if $h \neq s_p$ and $u_k.\mathit{max\text{-}seqno}(s_p) < i$. Moreover, the action $\mathtt{process\text{-}mpkt}_h(p)$ removes $\langle s_p, i_p\rangle$ from $\mathit{to\text{-}be\text{-}requested}(h')$ whenever it adds it to $\mathit{archived\text{-}pkts?}(h')$.

  Invariant 5.4 implies that $u_k.\mathit{archived\text{-}pkts?} \cap \{\langle s_p, i'\rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i' < i\} = \emptyset$. From the induction hypothesis, it is the case that $u_k.\mathit{to\text{-}be\text{-}requested}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, it follows that the invariant assertion holds in u.

■

Invariant 5.13 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$, *it is the case that* $u.\mathit{scheduled\text{-}rqsts?}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$ and $u.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\mathit{scheduled\text{-}rqsts?}(h')$ and $\mathit{archived\text{-}pkts?}(h')$.

- ❒ $\texttt{rm-leave}_h$: if $u_k.\mathit{status} = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\mathit{status} \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. It follows that $u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$ and $u.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

- ❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.\mathit{status} = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\mathit{status} = \texttt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by h', that is, $u_k.\mathit{min\text{-}seqno}(h') = \perp$, then it follows that $u_k.\mathit{window?}(h') = \emptyset$. Thus, Invariants 5.4 and 5.11 imply that $u_k.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$ and $u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$. If p is the next packet from h', that is, $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$ and $i_p = u_k.\mathit{max\text{-}seqno}(s_p) + 1$, then it is the case that $\mathit{id}(p) \notin u_k.\mathit{window?}(h')$. Thus, Invariants 5.4 and 5.11 imply that $\mathit{id}(p) \notin u_k.\mathit{scheduled\text{-}rqsts?}(h')$ and $\mathit{id}(p) \notin u_k.\mathit{archived\text{-}pkts?}(h')$.

  In either case the $\texttt{rm-send}_h(p)$ adds $\mathit{id}(p)$ to the variable $\mathit{archived\text{-}pkts?}(h')$ and does not affect $\mathit{scheduled\text{-}rqsts?}(h')$. It follows that $u.\mathit{scheduled\text{-}rqsts?}(h') = u_k.\mathit{scheduled\text{-}rqsts?}(h')$ and $u.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{archived\text{-}pkts?}(h') \cup \mathit{id}(p)$. From the induction hypothesis, it is the case that $u_k.\mathit{scheduled\text{-}rqsts?}(h') \cap u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Since it is the case that $\mathit{id}(p) \notin u_k.\mathit{scheduled\text{-}rqsts?}(h')$, it follows that $u.\mathit{scheduled\text{-}rqsts?}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$, as needed.

- ❒ $\texttt{schdl-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: the $\texttt{schdl-rqst}_h(s,i)$ action schedules a request for $\langle s,i \rangle$ and does not affect $\mathit{archived\text{-}pkts?}(h')$; that is, $u.\mathit{scheduled\text{-}rqsts?}(h') = u_k.\mathit{scheduled\text{-}rqsts?}(h') \cup \langle s,i \rangle$ and $u.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{archived\text{-}pkts?}(h')$.

  From the precondition of $\texttt{schdl-rqst}_h(s,i)$, it follows that $\langle s,i \rangle \in u_k.\mathit{to\text{-}be\text{-}requested}(h')$. From Invariant 5.12, it follows that $\langle s,i \rangle \notin u_k.\mathit{archived\text{-}pkts?}(h')$. Since it is the case that $u.\mathit{archived\text{-}pkts?} = u_k.\mathit{archived\text{-}pkts?}$, it follows that $\langle s,i \rangle \notin u.\mathit{archived\text{-}pkts?}(h')$. From the induction hypothesis, it is the case that $u_k.\mathit{scheduled\text{-}rqsts?}(h') \cap u_k.\mathit{archived\text{-}pkts?}(h') = \emptyset$. Thus, it follows that $u.\mathit{scheduled\text{-}rqsts?}(h') \cap u.\mathit{archived\text{-}pkts?}(h') = \emptyset$, as needed.

- ❒ $\texttt{send-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: from the precondition of $\texttt{send-rqst}_h(s,i)$, it is the case that $\langle s,i \rangle \in u_k.\mathit{scheduled\text{-}rqsts?}(h')$. Since $\texttt{send-rqst}_h(s,i)$ simply backs-off the request scheduled for $\langle s,i \rangle$, it follows that $u.\mathit{scheduled\text{-}rqsts?}(h') = u_k.\mathit{scheduled\text{-}rqsts?}(h')$. Moreover, $\texttt{send-rqst}_h(s,i)$ does not affect the variable $\mathit{archived\text{-}pkts?}(h')$. Thus, it follows that $u.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{archived\text{-}pkts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

- ❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\mathit{type}(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $\mathit{source}(p) = h'$: in this case, if the $\texttt{process-mpkt}_h(p)$ action archives the packet $\mathit{strip}(p)$, then it also cancels any requests scheduled for $id(p)$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-mpkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \texttt{RQST}$ and $source(p) = h'$: in this case, the $\texttt{process-mpkt}_h(p)$ action schedules a request for $id(p)$ only if $h \neq s_p$ and $id(p) \notin u_k.archived\text{-}pkts?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

■

Invariant 5.14 *Let u be any reachable state of* SRM-REC$_h$*. For $s \in H$, $i \in \mathbb{N}$, $t, t' \in \mathbb{R}^{\geq 0}$, and $k \in \mathbb{N}^+$, if $\langle s, i, t\rangle \in pending\text{-}rqsts$ and $\langle s, i, t', k\rangle \in scheduled\text{-}rqsts$, then $t < t'$.*

Proof: From Assumption 5.1, it is the case that $C_3 < C_1$. Thus, the expiration time of the back-off abstinence period precedes the transmission time of the respective request. ■

Invariant 5.15 *Let u be any reachable state of* SRM-REC$_h$*. For $h, s \in H$ and $i \in \mathbb{N}$, if the action $\texttt{send-rqst}_h(s,i)$ is enabled in u, i.e., $u.Pre(\texttt{send-rqst}_h(s,i)) = \texttt{True}$, then $\langle s, i\rangle \notin u.pending\text{-}rqsts?$.*

Proof: Suppose that $u.Pre(\texttt{send-rqst}_h(s,i)) = \texttt{True}$. From the precondition of the action $\texttt{send-rqst}_h(s,i)$, it follows that there exists $k \in \mathbb{N}^+$ such that $\langle s, i, t', k\rangle \in scheduled\text{-}rqsts$, for $t' = u.now$. Invariant 5.14 implies that there does not exist $t \in \mathbb{R}^{\geq 0}$ such that $\langle s, i, t\rangle \in pending\text{-}rqsts$ and $t' \leq t$. Since $t' = u.now$, it follows that $\langle s, i\rangle \notin u.pending\text{-}rqsts?$. ■

We proceed by presenting several lemmas pertaining to the RM$_I$ automaton.

Lemma 5.2 *Let $p \in P_{\text{RM-CLIENT}}$, α be any finite timed execution fragment of* RM$_I$*, and $u, u' \in states(\text{RM}_I)$, such that $u = \alpha.fstate$ and $u' = \alpha.lstate$. If $p \in u[\text{SRM}].sent\text{-}pkts$, then it is the case that $p \in u'[\text{SRM}].sent\text{-}pkts$.*

Proof: Follows directly from the fact that the variable $trans\text{-}time(p)$ may only be set by the automaton RM$_I$ to a value other than $\perp$. In particular, the variable $trans\text{-}time(p)$ may only be set by the action $rm\text{-}send_h(p)$, for $h = source(p)$, to the value of the variable now of the automaton SRM-REC$_h$. ■

Lemma 5.3 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in states(\text{RM}_I)$ be any reachable state of* RM$_I$*, such that $\langle s, i\rangle \in u[\text{SRM-REC}_h].archived\text{-}pkts?$. Moreover, let α be any timed execution fragment of* RM$_I$ *that starts in u, does not contain a $\texttt{rm-leave}_h$ action, and ends in some $u' \in states(\text{RM}_I)$. Then, it is the case that $\langle s, i\rangle \in u'[\text{SRM-REC}_h].archived\text{-}pkts?$.*

Proof: Follows from a simple induction on the length of α. The key point of the induction is that none of the actions of SRM-REC$_h$, except the action $\texttt{rm-leave}_h$ which is not contained in α, remove elements from or initialize the set SRM-REC$_h.archived\text{-}pkts?$. ■

Lemma 5.4 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in states(\text{RM}_I)$ be any reachable state of* RM$_I$*, such that $\langle s, i\rangle \in u[\text{SRM-REC}_h].scheduled\text{-}rqsts?$. Moreover, let α be any timed execution fragment of* RM$_I$ *that starts in u, does not contain a $\texttt{rm-leave}_h$ action, and ends in some $u' \in states(\text{RM}_I)$. Then, either $\langle s, i\rangle \in u'[\text{SRM-REC}_h].scheduled\text{-}rqsts?$ or $\langle s, i\rangle \in u'[\text{SRM-REC}_h].archived\text{-}pkts?$.*

Proof: Follows from a simple induction on the length of α. The key points of the induction are that: i) whenever the elements of SRM-REC_h.*scheduled-rqsts* pertaining to $\langle s, i\rangle$ are removed from SRM-REC_h.*scheduled-rqsts* then either another element pertaining to $\langle s, i\rangle$ is added to SRM-REC_h.*scheduled-rqsts* or $\langle s, i\rangle \in$ SRM-REC_h.*archived-pkts?*, and ii) from Lemma 5.3, none of the actions of SRM-REC_h, except the action $\texttt{rm-leave}_h$ which is not contained in α, remove elements from the set SRM-REC_h.*archived-pkts?*. ∎

Lemma 5.5 *Let $s, h \in H$, $i \in \mathbb{N}$, $t \in \mathbb{R}^{\geq 0}$, $k \in \mathbb{N}^+$, and $u \in states(\text{RM}_I)$ be any reachable state of RM_I, such that $u[\text{SRM-REC}_h].status = \texttt{member}$ and $\langle s, i, t, k\rangle \in u[\text{SRM-REC}_h].scheduled\text{-}rqsts$. Moreover, let α be any timed execution fragment of RM_I that starts in u, contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, and ends in some $u' \in states(\text{RM}_I)$, such that $t < u'.now$ and $\langle s, i, t', k'\rangle \in u'[\text{SRM-REC}_h].scheduled\text{-}rqsts$, for $t' \in \mathbb{R}^{\geq 0}$ and $k' \in \mathbb{N}^+$. Then, it is the case that $k < k'$.*

Proof: Invariant 5.13 and Lemma 5.4 imply that in any state u'' in α it is the case that $\langle s, i\rangle \in u''[\text{SRM-REC}_h].scheduled\text{-}rqsts?$. However, since $\langle s, i, t, k\rangle \in u[\text{SRM-REC}_h].scheduled\text{-}rqsts$, $t < u'.now$ and time is not allowed to progress past the scheduled transmission time of any request, it follows that the request for $\langle s, i\rangle$ is rescheduled for transmission in α for a point in time no earlier than $u'.now$. The only actions that may reschedule the request for $\langle s, i\rangle$ are the actions *send-rqst*$_h(s, i)$ and *process-mpkt*$_h(p)$, for $p \in P_{\text{SRM}}$, such that $id(p) = \langle s, i\rangle$ and $type(p) = \texttt{RQST}$. Whenever either of these actions reschedule the request for $\langle s, i\rangle$, they increment the element of the tuple corresponding to the round count. ∎

Lemma 5.6 *The occurrence of an action $\texttt{send-rqst}_h(s, i)$, for $h, s \in H$, and $i \in \mathbb{N}$, in any admissible timed execution α of RM_I is instantaneously succeeded in α by the occurrence of either a $\texttt{crash}_h$, $\texttt{rm-leave}_h$, or $\texttt{rec-msend}_h(p)$ action, where $p \in P_{\text{SRM}}$ is a retransmission request for the packet $\langle s, i\rangle$.*

Proof: The $\texttt{send-rqst}_h(s, i)$ action adds a $\texttt{RQST}$ packet for $\langle s, i\rangle$ to the variable SRM-REC_h.*msend-buff*. Moreover, SRM-REC_h prevents time from elapsing while it is the case that SRM-REC_h.*status* $\neq \texttt{crashed} \wedge$ SRM-REC_h.*msend-buff* $\neq \emptyset$. ∎

Lemma 5.7 *The occurrence of an action $\texttt{send-repl}_h(s, i)$, for $h, s \in H$ and $i \in \mathbb{N}$, in any admissible timed execution α of RM_I is instantaneously succeeded in α by the occurrence of either a $\texttt{crash}_h$, $\texttt{rm-leave}_h$, or $\texttt{rec-msend}_h(p)$ action, where $p \in P_{\text{SRM}}$ is a retransmission of (reply for) the packet $\langle s, i\rangle$.*

Proof: The $\texttt{send-repl}_h(s, i)$ action adds a $\texttt{REPL}$ packet for $\langle s, i\rangle$ to the variable SRM-REC_h.*msend-buff*. Moreover, SRM-REC_h prevents time from elapsing while it is the case that SRM-REC_h.*status* $\neq \texttt{crashed} \wedge$ SRM-REC_h.*msend-buff* $\neq \emptyset$. ∎

Lemma 5.8 *The occurrence of an action $\texttt{rec-msend}_h(p)$, for $h \in H$ and $p \in P_{\text{SRM}}$, in any admissible timed execution α of RM_I is instantaneously succeeded in α by the occurrence of either a $\texttt{crash}_h$, $\texttt{rm-leave}_h$, or $\texttt{msend}_h(pkt)$ action, for $pkt \in P_{\text{IPMCAST-CLIENT}}$, such that $strip(pkt) = p$.*

Proof: The $\texttt{rec-msend}_h(p)$ action adds an element to the variable SRM-IPBUFF_h.*msend-buff*. Moreover, SRM-IPBUFF_h prevents time from elapsing while SRM-IPBUFF_h.*status* $\neq \texttt{crashed} \wedge$ SRM-IPBUFF_h.*msend-buff* $\neq \emptyset$. ∎

Lemma 5.9 *The occurrence of an action* $\texttt{mrecv}_h(pkt)$*, for* $h \in H$ *and* $pkt \in P_{\text{SRM}}$*, in a state* $u \in states(\text{RM}_I)$ *in any admissible timed execution* α *of* RM_I*, such that* $u[\text{SRM-MEM}_h].status =$ $\texttt{member}$*, is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{process-mpkt}_h(p)$ *action, for* $p \in P_{\text{SRM}}$*, such that* $p = strip(pkt)$*.*

Proof: Since $u[\text{SRM-MEM}_h].status = \texttt{member}$, the particular occurrence of the $\texttt{mrecv}_h(pkt)$ action adds an element pertaining to pkt to the variable $\text{SRM-IPBUFF}_h.mrecv\text{-}buff$. Moreover, SRM-IPBUFF_h prevents time from elapsing while $\text{SRM-IPBUFF}_h.status \neq \texttt{crashed} \wedge$ $\text{SRM-IPBUFF}_h.mrecv\text{-}buff \neq \emptyset$. ■

Lemma 5.10 *Let* α *be any admissible execution of* RM_I *containing the discrete transition* (u, π, u')*, for* $u, u' \in states(\text{RM}_I)$*,* $h \in H$*,* $p \in P_{\text{RM-CLIENT}}$*,* $\langle s_p, i_p \rangle = id(p)$*, and* $\pi = \texttt{rm-send}_h(p)$*. If it is the case that either* $u[\text{SRM-REC}_h].min\text{-}seqno(s_p) = \perp$ *or* $u[\text{SRM-REC}_h].min\text{-}seqno(s_p) \neq \perp$ $\wedge i_p = u[\text{SRM-REC}_h].max\text{-}seqno(s_p) + 1$*, then the discrete transition* (u, π, u') *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{rec-msend}_h(p')$ *action, for* $p' = comp\text{-}data\text{-}pkt(p)$*.*

Proof: Suppose that either $u[\text{SRM-REC}_h].min\text{-}seqno(s_p) = \perp$ or $u[\text{SRM-REC}_h].min\text{-}seqno(s_p) \neq \perp$ and $i_p = u[\text{SRM-REC}_h].max\text{-}seqno(s_p) + 1$. Then, the discrete transition (u, π, u') adds the element p' to $\text{SRM-REC}_h.msend\text{-}buff$. Moreover, SRM-REC_h prevents time from elapsing while $\text{SRM-REC}_h.status \neq \texttt{crashed} \wedge \text{SRM-REC}_h.msend\text{-}buff \neq \emptyset$. ■

We now present some invariants pertaining to the RM_I automaton.

Invariant 5.16 *For* $h \in H$ *and any reachable state* u *of* RM_I*, it is the case that:*

1. $u[\text{RM-CLIENT}_h].status = \texttt{idle} \Leftrightarrow u[\text{SRM-MEM}_h].status = \texttt{idle}$,
2. $u[\text{RM-CLIENT}_h].status = \texttt{member} \Leftrightarrow u[\text{SRM-MEM}_h].status = \texttt{member}$,
3. $u[\text{RM-CLIENT}_h].status = \texttt{crashed} \Leftrightarrow u[\text{SRM-MEM}_h].status = \texttt{crashed}$,
4. $u[\text{RM-CLIENT}_h].status = \texttt{joining} \Leftrightarrow u[\text{SRM-MEM}_h].status \in Joining$*, and*
5. $u[\text{RM-CLIENT}_h].status = \texttt{leaving} \Leftrightarrow u[\text{SRM-MEM}_h].status \in Leaving$*.*

Proof: Let α be any finite timed execution of RM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of RM_I, it is the case that $u[\text{RM-CLIENT}_h].status = \texttt{idle}$ and $u[\text{SRM-MEM}_h].status = \texttt{idle}$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $\text{RM-CLIENT}_h.status$ and $\text{SRM-MEM}_h.status$.

- ❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets both variables $\text{RM-CLIENT}_h.status$ and $\text{SRM-MEM}_h.status$ to the value $\texttt{crashed}$. Thus, the invariant assertion holds in u.
- ❒ $\texttt{rm-join}_h$: from the precondition of the $\texttt{rm-join}_h$ action, it follows that $u_k[\text{RM-CLIENT}_h].status = \texttt{idle}$. From the induction hypothesis it follows that $u_k[\text{SRM-MEM}_h].status = \texttt{idle}$. Thus, the action $\texttt{rm-join}_h$ sets $\text{RM-CLIENT}_h.status$ to $\texttt{joining}$ and $\text{SRM-MEM}_h.status$ to $\texttt{join-rqst-pending}$; that is, $u[\text{RM-CLIENT}_h].status = \texttt{joining}$ and $u[\text{SRM-MEM}_h].status \in Joining$. It follows that the invariant assertion holds in u.

- ❐ $\texttt{mjoin}_h$: from the precondition of the $\texttt{mjoin}_h$ action, it follows that $u_k[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. From the induction hypothesis it follows that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{joining}$. The action $\texttt{mjoin}_h$ sets the variable $\text{SRM-MEM}_h.\mathit{status}$ to $\texttt{join-pending}$ and does not affect the variable $\text{RM-CLIENT}_h.\mathit{status}$. Thus, it is the case that $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$ and $u[\text{RM-CLIENT}_h].\mathit{status} = \texttt{joining}$. It follows that the invariant assertion holds in u.
- ❐ $\texttt{mjoin-ack}_h$: we first consider the case where $u_k[\text{SRM-MEM}_h].\mathit{status} \notin \mathit{Joining}$. In this case, $\texttt{mjoin-ack}_h$ affects neither $\text{RM-CLIENT}_h.\mathit{status}$ nor $\text{SRM-MEM}_h.\mathit{status}$. Thus, the induction hypothesis implies the invariant assertion in u.

  Second, we consider the case where $u_k[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. In this case, $\texttt{mjoin-ack}_h$ sets the variable $\text{SRM-MEM}_h.\mathit{status}$ to $\texttt{join-ack-pending}$ and does not affect $\text{RM-CLIENT}_h.\mathit{status}$. Since $u_k[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$, the induction hypothesis implies that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{joining}$. Moreover, since $\texttt{mjoin-ack}_h$ does not affect $\text{RM-CLIENT}_h.\mathit{status}$, it follows that $u[\text{RM-CLIENT}_h].\mathit{status} = \texttt{joining}$. Thus, the invariant assertion holds in u.
- ❐ $\texttt{rm-join-ack}_h$: from the precondition of $\texttt{rm-join-ack}_h$, it follows that $u_k[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. From the induction hypothesis it follows that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{joining}$. Thus, the $\texttt{rm-join-ack}_h$ action sets both $\text{SRM-MEM}_h.\mathit{status}$ and $\text{RM-CLIENT}_h.\mathit{status}$ to $\texttt{member}$. It follows that the invariant assertion holds in u.
- ❐ $\texttt{rm-leave}_h$: the reasoning for this action is analogous to that of $\texttt{rm-join}_h$.
- ❐ $\texttt{mleave}_h$: the reasoning for this action is analogous to that of $\texttt{mjoin}_h$.
- ❐ $\texttt{mleave-ack}_h$: the reasoning for this action is analogous to that of $\texttt{mjoin-ack}_h$.
- ❐ $\texttt{rm-leave-ack}_h$: the reasoning for this action is analogous to that of $\texttt{rm-join-ack}_h$.

■

Invariant 5.17 *For $h \in H$ and any reachable state u of RM_I, it is the case that $u[\text{RM-CLIENT}_h].\mathit{seqno} = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$.*

Proof: Let α be any finite timed execution of RM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of RM_I, it follows that $u[\text{RM-CLIENT}_h].\mathit{seqno} =\perp$ and $u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) =\perp$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u, we consider only the $\texttt{rm-send}_h(p)$ action, since this is the only action that affects the variables $\text{RM-CLIENT}_h.\mathit{seqno}$ and $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h)$.

From the precondition of $\texttt{rm-send}_h(p)$, it is the case that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$, $\mathit{source}(p) = h$, and either $u_k[\text{RM-CLIENT}_h].\mathit{seqno} =\perp$ or $\mathit{seqno}(p) = u_k[\text{RM-CLIENT}_h].\mathit{seqno} + 1$. The effects of $\texttt{rm-send}_h(p)$ are to set $\text{RM-CLIENT}_h.\mathit{seqno}$ to $\mathit{seqno}(p)$.

Since $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$, Invariant 5.16 implies that it is the case that $u_k[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$. From the induction hypothesis, it is the case that $u_k[\text{RM-CLIENT}_h].\mathit{seqno} = u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$. Thus, it is the case that either $u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) =\perp$ or $\mathit{seqno}(p) = u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) + 1$. In either case, the $\texttt{rm-send}_h(p)$ sets $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h)$ to $\mathit{seqno}(p)$. Thus, it follows that $u[\text{RM-CLIENT}_h].\mathit{seqno} = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$. ■

Invariant 5.18 *For $h \in H$ and any reachable state u of RM_I, it is the case that:*

1. $u[\text{SRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{SRM-IPBUFF}_h].status = \texttt{crashed}$ $\wedge u[\text{SRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{SRM-IPBUFF}_h].status = \texttt{member}$,
2. $u[\text{SRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{SRM-REC}_h].status = \texttt{crashed}$ $\wedge u[\text{SRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{SRM-REC}_h].status = \texttt{member}$, *and*
3. $u[\text{SRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{SRM-REP}_h].status = \texttt{crashed}$ $\wedge u[\text{SRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{SRM-REP}_h].status = \texttt{member}$.

Proof: We prove that $u[\text{SRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{SRM-IPBUFF}_h].status = \texttt{crashed} \wedge u[\text{SRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{SRM-IPBUFF}_h].status = \texttt{member}$; the proofs of the remaining claims are analogous.

Let α be any finite timed execution of RM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of RM_I, it follows that $u[\text{SRM-MEM}_h].status = \texttt{idle}$ and $u[\text{SRM-IPBUFF}_h].status = \texttt{idle}$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u, we consider only the actions that affect the variables $\text{SRM-MEM}_h.status$ and $\text{SRM-IPBUFF}_h.status$.

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets both variables $\text{SRM-MEM}_h.status$ and $\text{SRM-IPBUFF}_h.status$ to the value $\texttt{crashed}$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-join}_h$: from the precondition of $\texttt{rm-join}_h$, it follows that $u_k[\text{RM-CLIENT}_h].status = \texttt{idle}$. Invariant 5.16 implies that $u_k[\text{SRM-MEM}_h].status = \texttt{idle}$. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

  Since $\texttt{rm-join}_h$ sets $\text{SRM-MEM}_h.status$ to $\texttt{join-rqst-pending}$, it follows that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since $\texttt{rm-join}_h$ does not affect the variable $\text{SRM-IPBUFF}_h.status$, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ $\texttt{mjoin}_h$: from the precondition of $\texttt{mjoin}_h$, it follows that $u_k[\text{SRM-MEM}_h].status \in Joining$; that is, $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

  Since the action $\texttt{mjoin}_h$ sets the variable $\text{SRM-MEM}_h.status$ to $\texttt{join-pending}$, it follows that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Moreover, since $\texttt{mjoin}_h$ does not affect the variable $\text{SRM-IPBUFF}_h.status$, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ $\texttt{mjoin-ack}_h$: first, consider the case where $u_k[\text{SRM-MEM}_h].status \notin Joining$. Since in this case $\texttt{mjoin-ack}_h$ affects neither $\text{SRM-MEM}_h.status$ nor $\text{SRM-IPBUFF}_h.status$, the induction hypothesis implies that the invariant assertion holds in u.

  Second, consider the case where $u_k[\text{SRM-MEM}_h].status \in Joining$. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since $u_k[\text{SRM-MEM}_h].status \in Joining$, the action $\texttt{mjoin-ack}_h$ sets $\text{SRM-MEM}_h.status$ to $\texttt{join-ack-pending}$; that is, $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since $\texttt{mjoin}_h$ does not affect the variable $\text{SRM-IPBUFF}_h.status$, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ $\texttt{rm-join-ack}_h$: from the precondition of $\texttt{rm-join-ack}_h$, it is the case that $u_k[\text{SRM-MEM}_h].status \in Joining$. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

The action $\texttt{rm-join-ack}_h$ sets SRM-MEM$_h$.*status* to $\texttt{member}$. Since u_k[SRM-IPBUFF$_h$].*status* $\neq$ $\texttt{crashed}$, it also sets SRM-IPBUFF$_h$.*status* to $\texttt{member}$. It follows that the invariant assertion holds in u.

- ❒ $\texttt{rm-leave}_h$: from the precondition of the action $\texttt{rm-leave}_h$, it follows that u_k[RM-CLIENT$_h$].*status* $=$ $\texttt{member}$. Thus, Invariant 5.16 implies that u_k[SRM-MEM$_h$].*status* $=$ $\texttt{member}$. Moreover, the induction hypothesis implies that u_k[SRM-IPBUFF$_h$].*status* $=$ $\texttt{member}$.

  Since u_k[SRM-MEM$_h$].*status* $=$ $\texttt{member}$, the $\texttt{rm-leave}_h$ action sets SRM-MEM$_h$.*status* to $\texttt{leave-rqst-pending}$ and SRM-IPBUFF$_h$.*status* to $\texttt{idle}$. Thus, it is the case that u[SRM-MEM$_h$].*status* $\notin \{\texttt{crashed}, \texttt{member}\}$ and u[SRM-IPBUFF$_h$].*status* $\notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

- ❒ $\texttt{mleave}_h$: the reasoning for this action is analogous to that of $\texttt{mjoin}_h$.

- ❒ $\texttt{mleave-ack}_h$: the reasoning for this action is analogous to that of $\texttt{mjoin-ack}_h$.

- ❒ $\texttt{rm-leave-ack}_h$: from the precondition of the action $\texttt{rm-leave-ack}_h$, it follows that u_k[SRM-MEM$_h$].*status* $=$ $\texttt{leave-ack-pending}$. Since u_k[SRM-MEM$_h$].*status* $\notin$ $\{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that u_k[SRM-IPBUFF$_h$].*status* $\notin$ $\{\texttt{crashed}, \texttt{member}\}$.

  The action $\texttt{rm-leave-ack}_h$ sets SRM-MEM$_h$.*status* to $\texttt{idle}$ and does not affect the variable SRM-IPBUFF$_h$.*status*. Thus, it follows that u[SRM-MEM$_h$].*status* $\notin \{\texttt{crashed}, \texttt{member}\}$ and u[SRM-IPBUFF$_h$].*status* $\notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

■

Invariant 5.19 *For any reachable state u of* RM$_I$, *it is the case that* u[SRM-REC$_h$].*archived-pkts?* $\subseteq$ u[SRM].*sent-pkts?*, *for all $h \in H$.*

Proof: Let α be any finite timed execution of RM$_I$ leading to u. The proof is by strong induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of RM$_I$, it is the case that u[SRM-REC$_h$].*archived-pkts?* $= \emptyset$, for all $h \in H$, and u[SRM].*sent-pkts?* $= \emptyset$. Thus, the invariant assertion is trivially satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables SRM-REC$_h$.*archived-pkts?*, for all $h \in H$, and SRM.*sent-pkts?*.

- ❒ $\texttt{rm-leave}_h$, for $h \in H$: the action $\texttt{rm-leave}_h$ reinitializes the variable SRM-REC$_h$.*archived-pkts*. Thus, since u[SRM-REC$_h$].*archived-pkts* $= \emptyset$, it follows that u[SRM-REC$_h$].*archived-pkts?* $\subseteq$ u[SRM].*sent-pkts?*.

- ❒ $\texttt{rm-send}_h(p)$, for $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the action $\texttt{rm-send}_h(p)$ adds the element $\langle p, now \rangle$ to the variable SRM-REC$_h$.*archived-pkts* if and only if it sets the variable SRM-REC$_h$.*trans-time*(p) to *now*; that is, it adds the element $id(p)$ to SRM-REC$_h$.*archived-pkts?* if and only if it adds it to SRM.*sent-pkts?*. Thus, the induction hypothesis implies that u[SRM-REC$_h$].*archived-pkts?* $\subseteq$ u[SRM].*sent-pkts?*.

- ❒ $\texttt{process-mpkt}_h(p)$, for $h \in H$ and $p \in P_{\text{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$: from the precondition of $\texttt{process-mpkt}_h(p)$, it follows that there exists $pkt \in u_k$[SRM-IPBUFF$_h$].*mrecv-buff*, such that $strip(pkt) = p$. Since the only action that may add pkt to the variable SRM-IPBUFF$_h$.*mrecv-buff* is $\texttt{mrecv}_h(pkt)$, it follows that the action $\texttt{process-mpkt}_h(p)$ is preceded in α_k by an action $\texttt{mrecv}_h(pkt)$. Let $(u_2, \texttt{mrecv}_h(pkt), u_1)$ be the discrete transition in

α_k corresponding to the particular occurrence of the action $\texttt{mrecv}_h(pkt)$. Lemma 5.1 implies that the action $\texttt{mrecv}_h(pkt)$ is preceded in α_k by an action $\texttt{msend}_{h'}(pkt)$, for some $h' \in H$. Let $(u_4, \texttt{msend}_{h'}(pkt), u_3)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{msend}_{h'}(pkt)$. From the precondition of the action $\texttt{msend}_{h'}(pkt)$, it follows that $pkt \in u_4[\text{SRM-IP}\textsc{buff}_{h'}].\textit{msend-buff}$.

Since the only action that may add packets of type DATA or REPL to the variable SRM-IP$\textsc{buff}_{h'}$.*msend-buff* is the action $\texttt{rec-msend}_{h'}(p)$, it follows that an action $\texttt{rec-msend}_{h'}(p)$ precedes u_4 in α_k. Let $(u_6, \texttt{rec-msend}_{h'}(p), u_5)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{rec-msend}_{h'}(p)$. From the precondition of the action $\texttt{rec-msend}_{h'}(p)$, it follows that $p \in u_6[\text{SRM-}\textsc{rec}_{h'}].\textit{msend-buff}$. Invariant 5.7 implies that $id(p) \in u_6[\text{SRM-}\textsc{rec}_{h'}].\textit{archived-pkts?}$. From the induction hypothesis it is the case that $u_6[\text{SRM-}\textsc{rec}_{h'}].\textit{archived-pkts?} \subseteq u_6[\text{SRM}].\textit{sent-pkts?}$. Thus, Lemma 5.2 implies that $id(p) \in u[\text{SRM}].\textit{sent-pkts?}$. Since the action $\texttt{process-mpkt}_h(p)$ may only add the tuple $\langle strip(p), now\rangle$ to the variable $u[\text{SRM-}\textsc{rec}_h].\textit{archived-pkts}$, the fact that $id(p) \in u[\text{SRM}].\textit{sent-pkts?}$ and the induction hypothesis imply that $u[\text{SRM-}\textsc{rec}_h].\textit{archived-pkts?} \subseteq u[\text{SRM}].\textit{sent-pkts?}$, as needed.

■

Invariant 5.20 *For* $h \in H$ *and any reachable state* u *of* RM_I, *it is the case that* $u[\text{SRM-}\textsc{rec}_h].\textit{to-be-delivered?} \subseteq u[\text{SRM}].\textit{sent-pkts?}$.

Proof: Invariant 5.3 implies that, for $h' \in H$, it is the case that $u[\text{SRM-}\textsc{rec}_h].\textit{to-be-delivered?}(h') \subseteq u[\text{SRM-}\textsc{rec}_h].\textit{archived-pkts?}(h')$. Thus, it is the case that $u[\text{SRM-}\textsc{rec}_h].\textit{to-be-delivered?} \subseteq u[\text{SRM-}\textsc{rec}_h].\textit{archived-pkts?}$. Invariant 5.19 implies that $u[\text{SRM-}\textsc{rec}_h].\textit{to-be-delivered?} \subseteq u[\text{SRM}].\textit{sent-pkts?}$. ■

5.4.3 Relation Definition

We define a relation, R, from RM_I to $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$.

Definition 5.1 *Let* R *be the relation between states of* RM_I *and* $\text{RM}_S(\Delta)$, *for any* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, *such that for any states* u *and* s *of* RM_I *and* $\text{RM}_S(\Delta)$, *respectively,* $(u, s) \in R$ *provided that, for all* $h, h' \in H$ *and* $p \in P_{\text{RM-}\textsc{Client}}$, *such that* $\langle s_p, i_p\rangle = id(p)$, *it is the case that:*

$$\begin{aligned}
s.now &= u.now \\
s[\text{RM-}\textsc{Client}_h].\textit{status} &= u[\text{RM-}\textsc{Client}_h].\textit{status} \\
s[\text{RM-}\textsc{Client}_h].\textit{seqno} &= u[\text{RM-}\textsc{Client}_h].\textit{seqno} \\
s[\text{RM}(\Delta)].\textit{status}(h) &= \begin{cases} \texttt{idle} & \text{if } u[\text{SRM-}\textsc{mem}_h].\textit{status} = \texttt{idle} \\ \texttt{joining} & \text{if } u[\text{SRM-}\textsc{mem}_h].\textit{status} \in \textit{Joining} \\ \texttt{leaving} & \text{if } u[\text{SRM-}\textsc{mem}_h].\textit{status} \in \textit{Leaving} \\ \texttt{member} & \text{if } u[\text{SRM-}\textsc{mem}_h].\textit{status} = \texttt{member} \\ \texttt{crashed} & \text{if } u[\text{SRM-}\textsc{mem}_h].\textit{status} = \texttt{crashed} \end{cases} \\
s[\text{RM}(\Delta)].\textit{trans-time}(p) &= u[\text{SRM-}\textsc{rec}_{s_p}].\textit{trans-time}(p) \\
s[\text{RM}(\Delta)].\textit{expected}(h, h') &= u[\text{SRM-}\textsc{rec}_h].\textit{expected}(h') \\
s[\text{RM}(\Delta)].\textit{delivered}(h, h') &= u[\text{SRM-}\textsc{rec}_h].\textit{delivered}(h')
\end{aligned}$$

5.4.4 Safety Analysis

In this section, we show that our reliable multicast implementation RM_I indeed implements the reliable multicast service specification $\text{RM}_S(\infty)$. The following lemma states that the relation R of Definition 5.1 is a timed forward simulation relation from RM_I to $\text{RM}_S(\infty)$.

Lemma 5.11 *R is a timed forward simulation relation from RM_I to $\text{RM}_S(\infty)$.*

Proof: We must show that: i) if $u \in \mathit{start}(\text{RM}_I)$, then there is some $s \in \mathit{start}(\text{RM}_S(\infty))$ such that $(u, s) \in R$, and ii) if u is a reachable state of RM_I, s is a reachable state of $\text{RM}_S(\infty)$ such that $(u, s) \in R$, and $(u, \pi, u') \in \mathit{trans}(\text{RM}_I)$, then there exists a timed execution fragment α of $\text{RM}_S(\infty)$ such that: $\alpha.\mathit{fstate} = s$, $\mathit{ttrace}(\alpha) = \mathit{ttrace}(u\pi u')$, the total amount of time-passage in α is the same as the total amount of time-passage in $u\pi u'$, and $(u', s') \in R$, for $s' = \alpha.\mathit{lstate}$.

The satisfaction of the start condition is straightforward. For the step, we consider only the actions in $\mathit{acts}(\text{RM}_I)$ that affect the variables of RM_I that are used in R to obtain the corresponding state in $\text{RM}_S(\infty)$. Moreover, since the client automata RM-CLIENT$_h$, for all $h \in H$, are identical in both RM_I and $\text{RM}_S(\infty)$, we do not consider the effect of the actions of RM_I on the state of the client automata. Thus, we consider only the actions of the SRM component of RM_I that affect the variables of SRM that are present in R.

- ❒ $\texttt{crash}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{crash}_h$ action. The $\texttt{crash}_h$ action of RM_I simply sets the variable $u[\text{SRM-MEM}_h].\mathit{status}$ to `crashed` and resets $u[\text{SRM-REC}_h].\mathit{expected}(h')$ and $u[\text{SRM-REC}_h].\mathit{completed}(h')$, for all $h' \in H$. It is straightforward to see that the $\texttt{crash}_h$ action of $\text{RM}_S(\infty)$ mirrors these effects. Thus, it follows that $(u', s') \in R$.

- ❒ $\texttt{rm-join}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-join}_h$ action. It is straightforward to see that the effects of the $\texttt{rm-join}_h$ action in the specification correspond to those in the implementation.

- ❒ $\texttt{mjoin}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment. Since the $\texttt{mjoin}_h$ action is enabled in state u, it follows that $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. Thus, R implies that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{joining}$. The effects of the $\texttt{mjoin}_h$ action are to set the *status* variable to `join-pending`. It follows that $u'[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].\mathit{status}(h) = \texttt{joining}$. Thus, it follows that $(u', s') \in R$.

- ❒ $\texttt{mjoin-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment. The $\texttt{mjoin-ack}_h$ action affects the state of the SRM-MEM$_h$ automaton only when the host h is in the process of joining the reliable multicast group; that is, $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. Thus, R implies that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{joining}$. The effects of the $\texttt{mjoin-ack}_h$ action are to set the *status* variable to `join-ack-pending`. It follows that $u'[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].\mathit{status}(h) =$ `joining`. Thus, it follows that $(u', s') \in R$.

- ❒ $\texttt{rm-leave}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-leave}_h$ action. From the precondition of the $\texttt{rm-leave}_h$ action in the RM-CLIENT$_h$ automaton, it follows that $u[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$. Thus, Invariant 5.16 implies that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$ and, since $(u, s) \in R$, it is the case that $s[\text{RM}(\infty)].\mathit{status}(h) =$ `member`.

Since $u[\text{SRM-MEM}_h].status = \texttt{member}$, the $\texttt{rm-leave}_h$ action of RM_I sets the *status* variable of SRM-MEM_h to $\texttt{leave-rqst-pending}$. The $\texttt{rm-leave}_h$ action of $\text{RM}_S(\infty)$ sets the $status(h)$ variable of $\text{RM}(\infty)$ to $\texttt{leaving}$. Thus, it follows that $u'[\text{SRM-MEM}_h].status \in Leaving$ and $s'[\text{RM}(\infty)].status(h) = \texttt{leaving}$, as required by R.

Moreover, the $\texttt{rm-leave}_h$ action of RM_I resets the expected and delivered packet sets of SRM-REC_h; that is, $u'[\text{SRM-REC}_h].expected(h') = \emptyset$ and $u'[\text{SRM-REC}_h].delivered(h') = \emptyset$, for all $h' \in H$. Similarly, the $\texttt{rm-leave}_h$ action of $\text{RM}_S(\infty)$ also resets the variables $expected(h, h')$ and $delivered(h, h')$, for $h' \in H$; that is, $s'[\text{RM}(\infty)].expected(h, h') = \emptyset$ and $s'[\text{RM}(\infty)].delivered(h, h') = \emptyset$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{mleave}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment. Since the $\texttt{mleave}_h$ action is enabled in state u, it follows that $u[\text{SRM-MEM}_h].status \in Leaving$. Thus, R implies that $s[\text{RM}(\infty)].status(h) = \texttt{leaving}$. The effects of the $\texttt{mleave}_h$ action of RM_I are to set the *status* variable of SRM-MEM_h to $\texttt{leave-pending}$. It follows that $u'[\text{SRM-MEM}_h].status \in Leaving$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].status(h) = \texttt{leaving}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{mleave-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment. The $\texttt{mleave-ack}_h$ action affects the state of the SRM-MEM_h automaton only when the host h is in the process of leaving the reliable multicast group; that is, $u[\text{SRM-MEM}_h].status \in Leaving$. In this case, R implies that $s[\text{RM}(\infty)].status(h) = \texttt{leaving}$. The effects of the $\texttt{mleave-ack}_h$ action of RM_I are to set the *status* variable of SRM-MEM_h to $\texttt{leave-ack-pending}$. It follows that $u'[\text{SRM-MEM}_h].status \in Leaving$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].status(h) = \texttt{leaving}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-join-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-join-ack}_h$ action. We begin by showing that the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s. The precondition of the $\texttt{rm-join-ack}_h$ action of RM_I implies that $u[\text{SRM-MEM}_h].status \in Joining$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].status(h) = \texttt{joining}$. Thus, it follows that the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s.

The $\texttt{rm-join-ack}_h$ action of RM_I sets the *status* variable of SRM-MEM_h to $\texttt{member}$. Similarly, the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ sets the $status(h)$ variable of $\text{RM}_S(\infty)$ to $\texttt{member}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-leave-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-leave-ack}_h$ action. We begin by showing that the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s. The precondition of the $\texttt{rm-leave-ack}_h$ action of RM_I implies that $u[\text{SRM-MEM}_h].status \in Leaving$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].status(h) = \texttt{leaving}$. Thus, it follows that the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s.

The $\texttt{rm-leave-ack}_h$ action of RM_I sets the *status* variable of SRM-MEM_h to $\texttt{idle}$. Similarly, the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ sets the $status(h)$ variable of $\text{RM}_S(\infty)$ to $\texttt{idle}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-send}_h(p)$, for any $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-send}_h(p)$ action. Let s_p and i_p denote the source and sequence number of p, respectively.

From the precondition of the $\texttt{rm-send}_h(p)$ action of RM_I, it follows that $u[\text{RM-CLIENT}_h].status = \texttt{member}$ and $h = s_p$. Invariant 5.16 implies that $u[\text{SRM-MEM}_h].status = \texttt{member}$ and, since $(u, s) \in R$, it is the case that $s[\text{RM}(\infty)].status(h) = \texttt{member}$.

We consider the effects of $\texttt{rm-send}_h(p)$ according to whether p is the foremost packet from h. First, consider the case where p is the foremost packet from h; that is, $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) = \perp$. In this case, the $\texttt{rm-send}_h(p)$ action of RM_I sets the expected set from h to the set $\mathit{suffix}(p)$, adds $\mathit{id}(p)$ to the set of delivered packets from h, and records the transmission time of p.

Since it is the case that $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) = \perp$, Invariant 5.6 implies that $u[\text{SRM-REC}_h].\mathit{expected}(s_p) = \emptyset$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].\mathit{expected}(h, h) = \emptyset$. Thus, the $\texttt{rm-send}_h(p)$ action of $\text{RM}_S(\infty)$ matches the effects of the $\texttt{rm-send}_h(p)$ action of RM_I. It follows that $(u', s') \in R$.

Second, consider the case where p is not the foremost packet from h; that is, $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) \neq \perp$. In this case, Invariant 5.17 and the precondition of $\texttt{rm-send}_h(p)$ imply that $i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$. Thus, the $\texttt{rm-send}_h(p)$ action of RM_I records the transmission time of p and adds $\mathit{id}(p)$ to the set of delivered packets from h.

Since it is the case that $i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$, Invariant 5.2 implies that $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) < i_p$. Thus, it follows that $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{proper?}(h)$. Since $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$, Invariant 5.6 implies that $u[\text{SRM-REC}_h].\mathit{expected}(h) = u[\text{SRM-REC}_h].\mathit{proper?}(h)$. Thus, it follows that $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(h)$. Since $(u, s) \in R$, it is the case that $s[\text{RM}(\infty)].\mathit{expected}(h, h) = u[\text{SRM-REC}_h].\mathit{expected}(h)$. Thus, it follows that $\mathit{id}(p) \in s[\text{RM}(\infty)].\mathit{expected}(h, h)$. Thus, the $\texttt{rm-send}_h(p)$ action of $\text{RM}_S(\infty)$ also records the transmission time of p and adds p to the set of delivered packets from h. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-recv}_h(p)$, for any $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-recv}_h(p)$ action. Let s_p and i_p denote the source and sequence number of p, respectively.

We first show that the $\texttt{rm-recv}_h(p)$ action of $\text{RM}_S(\infty)$ is enabled in the state s. From the precondition of the $\texttt{rm-recv}_h(p)$ action of RM_I, it follows that $u[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$ and $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$. Invariant 5.18 implies that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$ and, since $(u, s) \in R$, it follows $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{member}$. Since $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$, Invariant 5.8 implies that $h \neq \mathit{source}(p)$. Moreover, Invariant 5.20 implies that $p \in u[\text{SRM}].\mathit{sent\text{-}pkts}$. Since $(u, s) \in R$, it follows that $p \in s[\text{RM}(\infty)].\mathit{sent\text{-}pkts}$.

We proceed by showing that s satisfies the last two terms in the precondition of $\texttt{rm-recv}_h(p)$ in $\text{RM}_S(\infty)$. Since the delivery delay parameter Δ is equal to ∞ for the $\text{RM}_S(\infty)$ automaton, $s[\text{RM}(\infty)]$ trivially satisfies the term $\mathit{expected}(h, s_p) = \emptyset \Rightarrow \mathit{now} \leq \mathit{trans\text{-}time}(p) + \Delta$.

Finally, we show that $s[\text{RM}(\infty)]$ satisfies the term $\mathit{expected}(h, s_p) \neq \emptyset \Rightarrow \mathit{id}(p) \in \mathit{expected}(h, s_p)$. Suppose that it is the case that $s[\text{RM}(\infty)].\mathit{expected}(h, s_p) \neq \emptyset$. Since $(u, s) \in R$, it follows that $u[\text{SRM-REC}_h].\mathit{expected}(s_p) \neq \emptyset$. Thus, since $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$, Invariant 5.9 implies that $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Finally, since $(u, s) \in R$, it follows that $\mathit{id}(p) \in s[\text{RM}(\infty)].\mathit{expected}(h, s_p)$, as needed.

The $\texttt{rm-recv}_h(p)$ action of RM_I sets the expected set of packets from s_p to the set $\mathit{suffix}(p)$, unless already non-empty, and adds p to the set of delivered packets from s_p. The $\texttt{rm-recv}_h(p)$ action of $\text{RM}(\infty)$ matches precisely the effects of the $\texttt{rm-recv}_h(p)$ action of RM_I. Thus, it follows that $(u', s') \in R$.

❒ $\nu(t)$, for any $t \in \mathbb{R}^{\geq 0}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\nu(t)$ action. Since the effects of the $\nu(t)$ actions of the RM_I and the $\text{RM}_S(\infty)$ automata are identical, it suffices to show that the $\nu(t)$ action is enabled in s. Since the delivery delay parameter Δ is equal to ∞ for the $\text{RM}_S(\infty)$ automaton, the term $\mathit{now} + t \leq \mathit{trans\text{-}time}(p) + \Delta$

of the precondition of the $\nu(t)$ action of $\mathrm{RM}_S(\infty)$ is satisfied for all $p \in P_{\text{RM-CLIENT}}$. Thus, it follows that the $\nu(t)$ action of $\mathrm{RM}_S(\infty)$ is enabled in s.

■

Theorem 5.12 $\mathrm{RM}_I \leq \mathrm{RM}_S(\infty)$

Proof: Follows directly from Lemma 5.11. ■

5.4.5 Liveness Analysis

In this section, we show that, under certain constraints, RM_I implements $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0}$.

Definitions

Suppose $p \in P_{\text{RM-CLIENT}}$, $pkt \in P_{\text{SRM}}$, and α is an admissible timed execution of RM_I that contains the transmission of p; that is, α contains the action $\texttt{rm-send}_h(p)$, for $h \in H, h = source(p)$. For $pkt \in P_{\text{SRM}}$, we say that *pkt pertains to p* if $type(pkt) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}\}$ and $id(pkt) = id(p)$. We let $P_{\text{SRM}}[p]$ denote the elements of P_{SRM} that pertain to p.

We let the number of packet drops in α pertaining to p, denoted $\alpha.drops(p)$, be the number of packet drops suffered by packets pertaining to p; that is, $\alpha.drops(p)$ is the number of occurrences of an action $\texttt{mdrop}(pkt', H_d)$ in α, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$ and $H_d \subseteq H$, such that $strip(pkt') \in P_{\text{SRM}}[p]$.

We let $aexecs_k(\mathrm{RM}_I)$, for $k \in \mathbb{N}^+$, be the set of admissible timed executions of RM_I in which the number of packet drops suffered by the packets pertaining to the transmission and, potentially, the recovery of any packet p is at most k. That is, $\alpha \in aexecs_k(\mathrm{RM}_I)$ iff $\alpha.drops(p') \leq k$, for any packet $p' \in P_{\text{RM-CLIENT}}$ transmitted in α. Finally, we let $attraces_k(\mathrm{RM}_I)$ be the traces of all executions of RM_I in $aexecs_k(\mathrm{RM}_I)$.

We let the transmission time of p in α, denoted $\alpha.trans\text{-}time(p)$, be the point in time in α at which p is transmitted; that is, the time of occurrence of $\texttt{rm-send}_h(p)$ in α. Since packets are transmitted by the clients of the reliable multicast service at most once (Lemma 4.2), it follows that the transmission time of any packet transmitted in any admissible timed execution of RM_I is well-defined and unique.

Execution Constraints

We proceed by defining several constraints on admissible executions of RM_I. These constraints facilitate the statement of conditional claims regarding the timely transmission of packets for RM_I.

Constraint 5.1 (No Crashes) *Let α be any admissible timed execution of* RM_I. *None of the hosts crash in α; that is, for any $h \in H$, no* $\texttt{crash}_h$ *actions occur in α.*

Constraint 5.2 (No Leaves) *Let α be any admissible timed execution of* RM_I. *None of the hosts leave the reliable multicast group in α; that is, for any $h \in H$, no* $\texttt{rm-leave}_h$ *actions occur in α.*

Let $\underline{d}, \overline{d} \in \mathbb{R}^{\geq 0}$, such that $\underline{d} > 0$, $\overline{d} > 0$, and $\underline{d} \leq \overline{d}$. The following constraint specifies the set of executions of RM_I in which the transmission latency between any two hosts $h, h' \in H, h \neq h'$ is bounded from below and above by $\underline{d}$ and $\overline{d}$, respectively.

Constraint 5.3 (Bounded Inter-host Transmission Latencies) *Let α be any admissible timed execution of* RM_I *and* h, h' *be any two distinct hosts in* H. *The transmission latency incurred by any packet multicast using the IP multicast service by* h *and received by* h' *in* α *lies in the interval* $[\underline{d}, \overline{d}]$; *that is, if* $p \in P_{\text{IPMCAST-CLIENT}}$ *is a packet multicast by* h *in* α, *then the time elapsing from the time of occurrence of the action* $\texttt{msend}_h(p)$ *to that of any action* $\texttt{mrecv}_{h'}(p)$ *lies in the interval* $[\underline{d}, \overline{d}]$.

The following constraint specifies the set of executions of RM_I in which the fate of any packet transmitted using the IP multicast service is resolved within $\underline{d}$ time units.

Constraint 5.4 (Bounded Transmission Resolution) *Let* α *be any admissible execution of* RM_I *containing the discrete transition* (u, π, u'), *for* $u, u' \in \mathit{states}(\mathrm{RM}_I)$, $h \in H$, $p \in P_{\text{IPMCAST-CLIENT}}$, *and* $\pi = \texttt{msend}_h(p)$. *Then, for all* $h' \in u[\text{IPMCAST}].\mathit{members}, h' \neq h$, *either a* $\texttt{crash}_{h'}$, $\texttt{rm-leave}_{h'}$, $\texttt{mrecv}_{h'}(p)$, *or* $\texttt{mdrop}(p, H_d)$, *for* $H_d \subseteq H$, $h' \in H_d$, *action occurs no later than* $\overline{d}$ *time units after the particular occurrence of the discrete transition* (u, π, u') *in* α.

The following constraint specifies the set of executions of RM_I in which the inter-host distance estimates of any host always lie in the interval $[\underline{d}, \overline{d}]$. The satisfaction of this constraint requires that $\texttt{DFLT-DIST} \in [\underline{d}, \overline{d}]$.

Constraint 5.5 (Bounded Inter-host Distance Estimates) *Let* α *be any admissible timed execution of* RM_I. *For any state* u *of* RM_I *in* α, *the inter-host distance estimates of the recovery component of each reliable multicast process of* RM_I *lie in the interval* $[\underline{d}, \overline{d}]$; *that is,* $u[\text{SRM-REC}_h].\mathit{dist}(h') \in [\underline{d}, \overline{d}]$, *for all* $h, h' \in H, h \neq h'$.

Letting $\texttt{DET-BOUND} \in \mathbb{R}^{\geq 0}$, such that $\overline{d} \leq \texttt{DET-BOUND}$, the following constraint specifies the set of executions of RM_I in which the delay in detecting packet losses is bounded by $\texttt{DET-BOUND}$.

Constraint 5.6 (Bounded Detection Latency) *Let* α *be any admissible timed execution of* RM_I. *Let* $p \in P_{\text{RM-CLIENT}}$ *be any packet transmitted in* α, $\mathit{id}(p) = \langle s_p, i_p \rangle$, *and* $h \in H, h \neq s_p$. *Moreover, let* u *be any state of* RM_I *in* α *such that* $\alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND} < u.\mathit{now}$. *Then, if* $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$, *then either* $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{delivered}(s_p)$ *or* $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$.

Let $\mathit{C\text{-}aexecs}(\mathrm{RM}_I)$ be the set of all admissible timed executions of RM_I in $\mathit{aexecs}(\mathrm{RM}_I)$ that satisfy Constraints 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6. Let $\mathit{C\text{-}attraces}(\mathrm{RM}_I)$ be the traces of all the executions of RM_I in $\mathit{C\text{-}aexecs}(\mathrm{RM}_I)$. Let $\mathit{C\text{-}aexecs}_k(\mathrm{RM}_I)$, for $k \in \mathbb{N}^+$, be the subset of $\mathit{aexecs}_k(\mathrm{RM}_I)$ comprised of all admissible timed executions of RM_I that satisfy Constraints 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6; that is, for $k \in \mathbb{N}^+$, $\mathit{C\text{-}aexecs}_k(\mathrm{RM}_I) = \mathit{aexecs}_k(\mathrm{RM}_I) \cap \mathit{C\text{-}aexecs}(\mathrm{RM}_I)$. Moreover, let $\mathit{C\text{-}attraces}_k(\mathrm{RM}_I)$ be the traces of all executions of RM_I in $\mathit{C\text{-}aexecs}_k(\mathrm{RM}_I)$.

Execution Definitions

Let α' be any admissible timed execution in $\mathit{C\text{-}aexecs}(\mathrm{RM}_I)$. We say that *the host h detects the loss of p in α'* if it schedules a request for $p \in P_{\text{RM-CLIENT}}$ in α'. If the host h detects the loss of p in α', then we let $\alpha'.\mathit{det\text{-}time}_h(p)$ denote the point in time in α' at which h detects the loss of p. We let $\alpha'.\mathit{det\text{-}latency}_h(p)$ denote *the loss detection latency of p for h* in α'; that is, the time elapsing from the time p is transmitted to the time the host h detects the loss of p in α'. We let

$\alpha'.rec\text{-}latency_h(p)$ denote the *loss recovery latency of p for h in α'*; that is, the time elapsing from the time the host h detects the loss of p to the time it receives p in α'.

When a host $h \in H$ schedules a request for $p \in P_{\text{RM-CLIENT}}$ with a back-off of $k-1$, for any $k \in \mathbb{N}^+$, we say that it initiates a k-th recovery round for p. Each recovery round (except the first) also initiates a back-off abstinence period. Any request for p received during this back-off abstinence period is discarded. If the packet p is received while a scheduled request for p by h is awaiting transmission, then the scheduled request is canceled. Once the back-off abstinence period expires, either the reception of a request for p or the transmission of the scheduled request for p by h initiates the $k+1$-st recovery round for p at h. In this case, we define the *k-th round request of h for p* to be the request for p upon whose reception or transmission the host h initiates the $k+1$-st recovery round for p. Moreover, we define the *completion time* of the k-th recovery round for p of h to be the point in time at which h either receives p or initiates its $k+1$-st recovery round for p.

Suppose that a host $h' \in H$ receives the k-th round request of h for p while it is a member of the reliable multicast group and after archiving the packet p. When h' receives this request, either i) a reply for p is already scheduled, ii) a reply for p is already pending, or iii) a reply for p is neither scheduled, nor pending. In the case where a reply for p is already scheduled, h's request for p is discarded. Moreover, the reply that is already scheduled at h' is considered to be the reply pertaining to the k-th round request of h for p. In the case where a reply for p is already pending, h's request for p is discarded. Moreover, the reply that is pending at h' is considered to be the reply pertaining to the k-th round request of h for p. Finally, in the case where a reply for p is neither scheduled, nor pending, h' schedules a reply for p. The reply that is either received or transmitted by h' and that results in the cancellation of the reply scheduled by h' for p is considered to be the reply to the k-th round request of h for p.

Liveness Proof

Lemma 5.13 *Let α be any admissible timed execution of RM_I that satisfies Constraint 5.3 and contains the occurrence of a discrete transition (u, π, u'), for $u, u' \in states(\text{RM}_I)$, $h \in H$, $p \in P_{\text{IPMCAST-CLIENT}}$, and $\pi = \texttt{mrecv}_h(p)$. Then, any other $\texttt{mrecv}_{h'}(p)$, for $h' \in H, h' \neq h$, in α occurs no earlier and no later than $\overline{d} - \underline{d}$ time units from the particular occurrence of (u, π, u') in α.*

Proof: Let (v, π, v'), for $v, v' \in states(\text{RM}_I)$, $h' \in H, h' \neq h$, $p \in P_{\text{IPMCAST-CLIENT}}$, and $\pi = \texttt{msend}_{h'}(p)$ be the discrete transition in α involving the transmission of p. Constraint 5.3 implies that the time elapsing from the time of occurrence of the action $\texttt{msend}_{h'}(p)$ to that of any action $\texttt{mrecv}_{h''}(p)$, for $h'' \in H, h'' \neq h'$ lies in the interval $[\underline{d}, \overline{d}]$. Thus, any two such actions are separated in time by at most $\overline{d} - \underline{d}$ time units. ■

Definition 5.2 *Let $h \in H$, $k \in \mathbb{N}^+$, $p \in P_{\text{RM-CLIENT}}$, $\langle s, i \rangle = id(p)$, and $\alpha \in C\text{-}aexecs(\text{RM}_I)$. We say that h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p upon the occurrence of a discrete transition (u, π, u') in α such that $\langle s, i, t, k \rangle \in u[\text{SRM-REC}_h].scheduled\text{-}rqsts$ and $\langle s, i, t', k+1 \rangle \in u'[\text{SRM-REC}_h].scheduled\text{-}rqsts$, for some $t, t' \in \mathbb{R}^{\geq 0}$.*

Lemma 5.14 *Let $k \in \mathbb{N}^+$, $k > 1$, $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and $\alpha \in C\text{-}aexecs(\text{RM}_I)$ such that α contains the transmission of p. Suppose that the host h schedules k-th and $k+1$-st round requests for the packet p in α. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in α at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_{k+1} \leq t_k + 2^{k-1}(C_1 + C_2)\overline{d}$.*

Proof: This follows from the fact that time in the SRM-REC_h automaton is not allowed to elapse past the transmission time of any scheduled request. Constraint 5.5 implies that the k-th round request is scheduled for transmission no later than $t_k + 2^{k-1}(C_1 + C_2)\overline{d}$. Thus, if no request is received by h prior to the time at which its k-th round request for p is scheduled for transmission, then h transmits its k-th round request. Thus, h either sends or receives its k-th round request for p no later than $t_k + 2^{k-1}(C_1 + C_2)\overline{d}$, as required. ■

Corollary 5.15 *Let $k \in \mathbb{N}^+$, $h \in H$, $p \in P_{\text{RM-Client}}$, and $\alpha \in C\text{-}aexecs(\text{RM}_I)$ such that α contains the transmission of p. Suppose that the host h schedules k-th and $k+1$-st round requests for the packet p in α. Let $t_{k+1} \in \mathbb{R}^{\geq 0}$ be the point in time in α at which the host h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p. Then, it is the case that $t_{k+1} \leq \alpha.det\text{-}time_h(p) + (2^k - 1)(C_1 + C_2)\overline{d}$.*

Proof: Follows from Lemma 5.14 and the fact that h detects the loss of p at the point in time when it first schedules a request for p. According to the SRM-REC_h automaton, the first request scheduled for a packet is either a 1-st or 2-nd round request for the given packet. ■

Lemma 5.16 *Let $k \in \mathbb{N}^+$, $k > 1$, $h \in H$, $p \in P_{\text{RM-Client}}$, and $\alpha \in C\text{-}aexecs(\text{RM}_I)$ such that α contains the transmission of p. Suppose that the host h schedules k-th and $k+1$-st round requests for the packet p in α. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in α at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_k + 2^{k-1}C_3\underline{d} < t_{k+1}$.*

Proof: Constraint 5.5 implies that the k-th round back-off abstinence period expires no earlier than $2^{k-1}C_3\underline{d}$ time units past t_k; that is, no earlier than $t_k + 2^{k-1}C_3\underline{d}$ in α. From Assumption 5.1, it is the case that $C_3 < C_1$. Thus, the k-th round request is scheduled for transmission at a point in time that succeeds $t_k + 2^{k-1}C_3\underline{d}$ in α.

The host h schedules its $k+1$-st round request for p when it either sends or receives its k-th round request for p; that is, upon the occurrence of either a $\texttt{send-rqst}_h(s, i)$ action, such that $\langle s, i\rangle = id(p)$, or a $\texttt{process-mpkt}_h(pkt)$ action, for $pkt \in P_{\text{SRM}}$, such that $id(pkt) = id(p)$ and $type(pkt) = \texttt{RQST}$. In the case of a $\texttt{send-rqst}_h(s, i)$ action, Invariant 5.15 implies that if the $\texttt{send-rqst}_h(s, i)$ action is enabled, then a request for p is not pending. In the case of a $\texttt{process-mpkt}_h(pkt)$ action, the effects of the action $\texttt{process-mpkt}_h(pkt)$ imply that the k-th round request for p is backed-off only while a request for p is not pending.

It follows that the point in time at which the host h either sends or receives its k-th round request for p succeeds the expiration time of the back-off abstinence period of the k-th round request of h for p; that is, $t_k + 2^{k-1}C_3\underline{d} < t_{k+1}$. ■

Lemma 5.17 *Let $h, h' \in H, h \neq h'$, $p \in P_{\text{RM-Client}}$, and $\alpha \in C\text{-}aexecs(\text{RM}_I)$ such that α contains the transmission of p. Suppose that h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in α. Suppose that when h' receives this request, it is a member of the reliable multicast group and has already archived p. Then, the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than $t' + (D_1 + D_2)\overline{d}$ in α.*

Proof: Constraint 5.5 implies a reply is scheduled for transmission no later than $(D_1 + D_2)\overline{d}$ time units past its scheduling time. When h' receives the request of h for p, a reply for p is either already scheduled, already pending, or neither scheduled nor pending. We consider each of these scenarios separately. First, if a reply for p is already scheduled, its transmission time is no later than $t' + (D_1 + D_2)\overline{d}$ in α. Thus, if either an original transmission or a reply for p is not received by h' by the scheduled transmission time of its own reply, then the host h' transmits its own reply. It follows that the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than the point in time $t' + (D_1 + D_2)\overline{d}$ in α. Second, if a reply for p is already pending, then the reply of h' pertaining to the particular request of h for p has already been either sent or received; that is, the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than t'. Finally, if a reply for p is neither scheduled nor pending, then the reply of h' pertaining to the particular request for p from h is scheduled for no later than $t' + (D_1 + D_2)\overline{d}$. In either scenario, the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than $t' + (D_1 + D_2)\overline{d}$ in α. ■

Lemma 5.18 *Let $h, h' \in H, h \neq h'$, $p \in P_{\text{RM-CLIENT}}$, and $\alpha \in$ C-aexecs(RM_I) such that α contains the transmission of p. Suppose that h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in α. Suppose that when h' receives this request, it is a member of the reliable multicast group and has already archived p. Then, the reply abstinence period of the reply of h' pertaining to the particular request of h for p expires no later than $t' + (D_1 + D_2 + D_3)\overline{d}$ in α.*

Proof: Constraint 5.5 implies that the reply abstinence period of any reply expires no later than $(D_1 + D_2 + D_3)\overline{d}$ time units past its scheduling time. The rest of the proof is analogous to the proof of Lemma 5.17. ■

Lemma 5.19 *Let $k \in \mathbb{N}^+$, $h, h' \in H, h \neq h'$, $p \in P_{\text{RM-CLIENT}}$, and $\alpha \in$ C-aexecs(RM_I) such that α contains the transmission of p. Suppose that the host h schedules k-th and $k+1$-st round requests for the packet p in α. Suppose that the host h' receives the k-th round request of h for p. Let $t_{k+1} \in \mathbb{R}^{\geq 0}$ be the point in time in α at which the host h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p. Then, the host h' receives the k-th round request of h for p no later than $t_{k+1} + \overline{d}$ in α.*

Proof: The host h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p upon the occurrence of either a $\texttt{send-rqst}_h(s, i)$ or a $\texttt{process-mpkt}_h(pkt)$ action, where $id(pkt) = id(p)$ and $type(pkt) = \texttt{RQST}$. We consider there two cases separately.

First, in the case of a $\texttt{send-rqst}_h(s, i)$ action, Constraints 5.1 and 5.2 and Lemmas 5.6 and 5.8 imply that the $\texttt{send-rqst}_h(s, i)$ action is instantaneously followed by a $\texttt{msend}_h(pkt')$ action, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$, such that $id(strip(pkt')) = id(p)$ and $type(strip(pkt')) = \texttt{RQST}$. Furthermore, Constraint 5.3 implies that h' receives this request within at most $\overline{d}$ time units.

Second, in the case of a $\texttt{process-mpkt}_h(pkt)$ action, a $\texttt{mrecv}_h(pkt')$ action, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$, such that $pkt = strip(pkt')$, instantaneously precedes $\texttt{process-mpkt}_h(pkt)$. Lemma 5.13 implies that h' receives this request within at most $\overline{d} - \underline{d}$ time units. ■

Lemma 5.20 *Let α be any admissible timed execution of RM_I that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. For any state $u \in states(\text{RM}_I)$ in α, if $u.trans\text{-}time(p) \neq \perp$, then $u.trans\text{-}time(p) = \alpha.trans\text{-}time(p)$.*

Proof: The only action that sets the variable $trans\text{-}time(p)$ is the action $\texttt{rm-send}_h(p)$, for $h = source(p)$. By Lemma 4.2, the action $\texttt{rm-send}_h(p)$ occurs only once in α. Let $(v, \texttt{rm-send}_h(p), v')$ be the discrete transition in α involving the action $\texttt{rm-send}_h(p)$. By the definition of $\alpha.trans\text{-}time(p)$, it follows that $\alpha.trans\text{-}time(p) = v.now$. The action $\texttt{rm-send}_h(p)$ sets the variable $trans\text{-}time(p)$ to the value of now. It follows that $v'.trans\text{-}time(p) = \alpha.trans\text{-}time(p)$.

Since the action $\texttt{rm-send}_h(p)$ occurs in α only once, it follows that for any $v_-, v_+ \in \alpha$, such that $v_- \leq_\alpha v$ and $v' \leq_\alpha v_+$, it is the case that $v_-.\mathit{trans\text{-}time}(p) = \perp$ and $v_+.\mathit{trans\text{-}time}(p) = v'.\mathit{trans\text{-}time}(p)$. Since $v'.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$, it follows that $v_+.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$. ■

Lemma 5.21 *Let $h, h' \in H$, $\alpha \in \mathit{aexecs}(\mathrm{RM}_I)$, $u, u' \in \mathit{states}(\mathrm{RM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the finite execution fragment of α starting in u and ending in u'. If $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$ and $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$.*

Proof: The proof is by induction on the length $n \in \mathbb{N}$ of $\alpha_{uu'}$. For the base case, consider a finite execution fragment $\alpha_{uu'}$ of length $n = 0$. Since $u = u'$, it trivially follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$.

For the inductive step, consider an execution fragment $\alpha_{uu'}$ of length $n = k+1$. Let α_k be the prefix of $\alpha_{uu'}$ involving the first k steps and $u_k = \alpha_k.\mathit{lstate}$. Suppose that $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$ and $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. The induction hypothesis implies that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u_k[\text{SRM-REC}_h].\mathit{expected}(h')$.

Now, consider the step from u_k to u'. The only actions of SRM-REC_h that may affect the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ are the actions $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{rm-send}_h(p)$, and $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$. $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. The action $\texttt{rm-send}_h(p)$ affects the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ only when $h' = h = \mathit{source}(p)$ and $\text{SRM-REC}_h.\mathit{expected}(h') = \emptyset$. The action $\texttt{rm-recv}_h(p)$ affects the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ only when $h' = \mathit{source}(p)$ and $\text{SRM-REC}_h.\mathit{expected}(h') = \emptyset$. Since $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$, the step from u_k to u' does not affect the variable $\text{SRM-REC}_h.\mathit{expected}(h')$; that is, $u_k[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. Since $u[\text{SRM-REC}_h].\mathit{expected}(h') = u_k[\text{SRM-REC}_h].\mathit{expected}(h')$, it follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. ■

Lemma 5.22 *Let $h, h' \in H$, $\alpha \in \mathit{aexecs}(\mathrm{RM}_I)$, $u, u' \in \mathit{states}(\mathrm{RM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the execution fragment of α starting in u and ending in u'. If $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$.*

Proof: Suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. If it is the case that $u[\text{SRM-REC}_h].\mathit{expected}(h') = \emptyset$, then it trivially follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$. Otherwise, if $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$, then Lemma 5.21 implies that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. It follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$. ■

Lemma 5.23 *Let $h, h' \in H$, $\alpha \in \mathit{aexecs}(\mathrm{RM}_I)$, $u, u' \in \mathit{states}(\mathrm{RM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the finite execution fragment of α starting in u and ending in u'. If $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{SRM-REC}_h].\mathit{delivered}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{delivered}(h')$.*

Proof: Follows by induction on the length $n \in \mathbb{N}$ of the finite execution fragment $\alpha_{uu'}$ after recognizing that all actions, except $\texttt{crash}_h$ and $\texttt{rm-leave}_h$, may only add elements to the variable $\text{SRM-REC}_h.\mathit{delivered}(h')$. ■

Lemma 5.24 *Let* $k \in \mathbb{N}^+$, $p \in P_{\text{RM-CLIENT}}$, *and* α *be any admissible timed execution of* RM_I *in* $C\text{-}aexecs_k(\text{RM}_I)$ *that contains the transmission of* p. *Moreover, let* $h \in H$ *and* u *be any state of* RM_I *in* α *such that* $\alpha.\mathit{trans\text{-}time}(p) + \overline{d} < u.\mathit{now}$ *and* $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(\mathit{source}(p))$. *For any state* $u' \in \mathit{states}(\text{RM}_I)$ *in* α *such that* $\alpha.\mathit{trans\text{-}time}(p) + \overline{d} < u'.\mathit{now}$ *and* $u' \leq_\alpha u$, *it is the case that* $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(\mathit{source}(p))$.

Proof: Let $\mathit{id}(p) = \langle s_p, i_p \rangle$ and $p' \in P_{\text{RM-CLIENT}}$, such that $\mathit{id}(p') = \langle s_p, i' \rangle$, be the earliest packet expected from s_p by h in the state u; that is, $\mathit{id}(p') \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$ and for all $\langle s_p, i'' \rangle \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$ it is the case that $i' \leq i''$. Thus, it follows that $i' \leq i$.

The variable $\text{SRM-REC}_h.\mathit{expected}(s_p)$ is set in α upon either the transmission (when $h = s_p$) or the reception (when $h \neq s_p$) of p'. Let $v \in \mathit{states}(\text{RM}_I)$ be the state following either the transmission or the reception of p' by h in α, respectively. By definition of v, it is the case that $v[\text{SRM-REC}_h].\mathit{expected}(s_p) \neq \emptyset$. Since α contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions (Constraints 5.1 and 5.2), Lemma 5.21 implies that for any $v' \in \mathit{states}(\text{RM}_I)$ in α, such that $v \leq_\alpha v'$, it is the case that $v[\text{SRM-REC}_h].\mathit{expected}(s_p) = v'[\text{SRM-REC}_h].\mathit{expected}(s_p)$.

Constraint 5.3 implies that $v.\mathit{now} \leq \alpha.\mathit{trans\text{-}time}(p') + \overline{d}$. Moreover, Lemma 4.3 implies that $\alpha.\mathit{trans\text{-}time}(p') \leq \alpha.\mathit{trans\text{-}time}(p)$. Since $\alpha.\mathit{trans\text{-}time}(p) + \overline{d} < u'.\mathit{now}$, it follows that $v.\mathit{now} < u'.\mathit{now}$. Since $v.\mathit{now} < u'.\mathit{now}$, it follows that $v \leq_\alpha u'$. Thus, since $v \leq_\alpha u'$, $u' \leq_\alpha u$, and $v[\text{SRM-REC}_h].\mathit{expected}(s_p) \neq \emptyset$, Lemma 5.21 implies that $v[\text{SRM-REC}_h].\mathit{expected}(s_p) = u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$ and $v[\text{SRM-REC}_h].\mathit{expected}(s_p) = u[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Thus, it is the case that $u'[\text{SRM-REC}_h].\mathit{expected}(s_p) = u[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Since $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$, it follows that $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$. ■

Let $k^* = \lceil \log_2[(D_1 + D_2 + D_3 + 2)\overline{d} - \underline{d}] - \log_2(C_3\underline{d}) \rceil$. The following lemma states that, under Constraints 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6, k^* is the number of requests that must be scheduled before the request scheduling delays become large enough to ensure that one round's replies do not interfere with the next round's requests.

Lemma 5.25 *Let* $k \in \mathbb{N}^+, k \geq k^*$, $p \in P_{\text{RM-CLIENT}}$, $h, h' \in H, h \neq h'$, *and* $\alpha \in C\text{-}aexecs_k(\text{RM}_I)$, *such that* α *contains the transmission of* p.

Let $u \in \mathit{states}(\text{RM}_I)$ *be any state in* α, *such that* $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{expected}(\mathit{source}(p))$ *and* $\mathit{id}(p) \notin u[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$, *following which* h *schedules a* $k+2$*-nd round request for* p.

Let $u' \in \mathit{states}(\text{RM}_I)$ *be any state in* α, *such that* $\mathit{id}(p) \in u'[\text{SRM-REC}_{h'}].\mathit{delivered}(\mathit{source}(p))$, *following which* h' *receives the* k*-th and* $k+1$*-st round requests of* h *for* p.

The replies of h' *to the* k*-th and* $k+1$*-st round requests of* h *for* p *are distinct.*

Proof: It suffices to show that the reply abstinence period pertaining to h''s reply to the k-th round request of h for p expires prior to the time at which h' receives the $k+1$-st round request of h for p.

Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in α at which h schedules its k-th and $k+1$-st round requests for p. From Lemma 5.19, h' receives the k-th round request of h for p no later than $t_{k+1} + \overline{d}$. From Lemma 5.18, the abstinence period of the reply of h to the k-th round request of h for p expires no later than $t_{k+1} + \overline{d} + (D_1 + D_2 + D_3)\overline{d}$.

From Lemma 5.16, h either receives or transmits its $k+1$-st round request after the point in time $t_{k+1} + 2^k C_3 \underline{d}$. From Lemma 5.13, h' receives such a request after the point in time $t_{k+1} + 2^k C_3 \underline{d} - \overline{d} + \underline{d}$. Since $k^* = \lceil \log_2[(D_1 + D_2 + D_3 + 2)\overline{d} - \underline{d}] - \log_2(C_3\underline{d}) \rceil$ and $k \geq k^*$, it follows that $t_{k+1} + \overline{d} + (D_1 + D_2 + D_3)\overline{d} \leq t_{k+1} + 2^k C_3 \underline{d} - \overline{d} + \underline{d}$.

Recall that h' receives the $k+1$-st round request of h for p after the point in time $t_{k+1}+2^k C_3\underline{d}-\overline{d}+\underline{d}$. Since $t_{k+1} + \overline{d} + (D_1 + D_2 + D_3)\overline{d} \leq t_{k+1} + 2^k C_3\underline{d} - \overline{d} + \underline{d}$, it follows that h' receives the $k+1$-st round request of h for p after the expiration of the abstinence period of the reply of h' to the k-th round request of h for p. It follows that the replies of h' to the k-th and $k+1$-st round requests of h for p are distinct. ■

Let $\texttt{REC-BOUND}(k) = [(2^k - 1)(C_1 + C_2) + D_1 + D_2 + 2]\overline{d}$, for $k \in \mathbb{N}^+$. The following lemma states that, for $k \in \mathbb{N}^+$, the recovery of any packet in an admissible execution $\alpha \in \textit{C-aexecs}_k(\text{RM}_I)$ involves at most $k^* + k$ recovery rounds. Following the k^*-th recovery round, one round's replies do not interfere with the next round's requests. Thus, all recovery rounds that follow the first k^* recovery rounds may fail only due to packet drops. Since the number of packet drops pertaining to the recovery of any packet in α is at most k, it follows that at most $k^* + k$ recovery rounds are needed to recover any packet in α.

Lemma 5.26 *Let $k \in \mathbb{N}^+$, $\alpha \in \textit{C-aexecs}_k(\text{RM}_I)$, and $u, u' \in \textit{states}(\text{RM}_I)$ be any states in α such that $u.\textit{now} + \texttt{REC-BOUND}(k^* + k) < u'.\textit{now}$. For any $h \in H$ and $p \in P_{\text{RM-CLIENT}}$, if $\textit{id}(p) \in u[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$, then $\textit{id}(p) \in u'[\text{SRM-REC}_h].\textit{delivered}(\textit{source}(p))$.*

Proof: Since $\alpha \in \textit{C-aexecs}_k(\text{RM}_I)$, Constraints 5.1 and 5.2 imply that the source s_p of p neither crashes nor leaves the reliable multicast group following the transmission of p. Thus, it is capable of replying to any of the retransmission requests for p sent in α.

Suppose that $\textit{id}(p) \in u[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$ and let $v \in \textit{states}(\text{RM}_I)$ be the first state in α such that $\textit{id}(p) \in v[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$ and $v' \in \textit{states}(\text{RM}_I)$ be the first state in α such that $v.\textit{now} + \texttt{REC-BOUND}(k^* + k) < v'.\textit{now}$. By definition, it follows that $v \leq_\alpha u$ and $v' \leq_\alpha u'$.

Since $\alpha \in \textit{C-aexecs}_k(\text{RM}_I)$, it contains at most k packet drops pertaining to the transmission and recovery of p. The loss of the original transmission of the packet p accounts for at least one such packet drop. Thus, at most $k-1$ packet drops may occur during the recovery p. Lemmas 5.4 and 5.5 imply that following the state v in α, the host h continues initiating recovery rounds for p until p is recovered. We proceed by showing that the host h recovers p by the completion time of its $k^* + k$ recovery round for p.

Consider the interaction of s_p and h pertaining to h's recovery of p. From Lemma 5.25, the replies of s_p to the requests of the recovery rounds of h following the k^*-th round of h are distinct. Thus, each recovery round following the k^*-th recovery round may fail either due to the loss of the request or the loss of the reply of the given round; that is, each recovery round following the k^*-th recovery round that fails accounts for at least one packet drop. It follows that at most k^*+k recovery rounds are required for h to successfully recover p.

Corollary 5.15, Lemma 5.17, and Constraint 5.3 imply that h completes its $k^* + k$ recovery rounds no later than $\texttt{REC-BOUND}(k^* + k)$ time units past the point in time at which it schedules its first request for p. Since v is the first state in α such that $\textit{id}(p) \in v[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$ and $v.\textit{now} + \texttt{REC-BOUND}(k^* + k) < v'.\textit{now}$, it follows that h receives p prior to v' in α. Lemma 5.23 implies that $\textit{id}(p) \in v'[\text{SRM-REC}_h].\textit{delivered}(s_p)$.

Since $v' \leq_\alpha u'$ and $\textit{id}(p) \in v'[\text{SRM-REC}_h].\textit{delivered}(s_p)$, Lemma 5.23 implies that $\textit{id}(p) \in u'[\text{SRM-REC}_h].\textit{delivered}(s_p)$. ■

Lemma 5.27 *Let $k \in \mathbb{N}^+$, $\Delta = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $p \in P_{\text{RM-CLIENT}}$, α be any admissible timed execution of RM_I in $\textit{C-aexecs}_k(\text{RM}_I)$ that contains the transmission of p, and $u \in \textit{states}(\text{RM}_I)$ be any state in α such that $\alpha.\textit{trans-time}(p) + \Delta < u.\textit{now}$. For any $h \in H$, if $h \in u.\textit{intended}(p)$, then it is the case that $h \in u.\textit{completed}(p)$.*

Proof: Let $s_p = \mathit{source}(p)$ and suppose that $h \in u.\mathit{intended}(p)$. Since $h \in u.\mathit{intended}(p)$, it follows that $\mathit{id}(p) \in u[\text{SRM-}\textsc{rec}_h].\mathit{expected}(s_p)$. Let $u' \in \mathit{states}(\text{RM}_I)$ be the earliest state in α such that $\alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND} < u'.\mathit{now}$. Since $\overline{d} \leq \texttt{DET-BOUND}$, it follows that $\alpha.\mathit{trans\text{-}time}(p) + \overline{d} < u'.\mathit{now}$. Since $\mathit{id}(p) \in u[\text{SRM-}\textsc{rec}_h].\mathit{expected}(s_p)$, $\alpha.\mathit{trans\text{-}time}(p) + \overline{d} < u'.\mathit{now}$, and $u' \leq_\alpha u$, Lemma 5.24 implies that $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{expected}(s_p)$. Constraint 5.6 implies that either $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{delivered}(s_p)$ or $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{scheduled\text{-}rqsts?}$.

First, consider the case where $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{delivered}(s_p)$. Since either $u' \leq_\alpha u$ and $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{delivered}(s_p)$, Lemma 5.23 implies that $\mathit{id}(p) \in u[\text{SRM-}\textsc{rec}_h].\mathit{delivered}(s_p)$. It follows that $h \in u.\mathit{completed}(p)$.

Second, consider the case where $\mathit{id}(p) \in u'[\text{SRM-}\textsc{rec}_h].\mathit{scheduled\text{-}rqsts?}$. Let (u'_-, π, u') be the discrete transition in α leading to the particular occurrence of u'. Since, u' is the earliest state in α such that $\alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND} < u'.\mathit{now}$, it follows that π is a non-stuttering time-passage action, $u'_-.\mathit{now} < u'.\mathit{now}$, and $u'_-.\mathit{now} \leq \alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND}$. Since time-passage actions do not affect the derived variable $\text{SRM-}\textsc{rec}_h.\mathit{scheduled\text{-}rqsts?}$, it follows that $\mathit{id}(p) \in u'_-[\text{SRM-}\textsc{rec}_h].\mathit{scheduled\text{-}rqsts?}$. Since $u'_-.\mathit{now} \leq \alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND}$ and $\alpha.\mathit{trans\text{-}time}(p) + \Delta < u.\mathit{now}$, it follows that $u'_-.\mathit{now} + \texttt{REC-BOUND}(k^* + k) < u.\mathit{now}$.

Since $u'_-.\mathit{now} + \texttt{REC-BOUND}(k^* + k) < u.\mathit{now}$ and $\mathit{id}(p) \in u'_-[\text{SRM-}\textsc{rec}_h].\mathit{scheduled\text{-}rqsts?}$, Lemma 5.26 implies that $\mathit{id}(p) \in u[\text{SRM-}\textsc{rec}_h].\mathit{delivered}(s_p)$; that is, $h \in u.\mathit{completed}(p)$. ■

We conclude by showing that any timed trace of RM_I in the set $C\text{-}\mathit{attraces}_k(\text{RM}_I)$ is also a timed trace of the specification automaton $\text{RM}_S(\Delta)$, for $\Delta = \texttt{DET-BOUND}+\texttt{REC-BOUND}(k^*+k)$. Thus, given Constraints 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6 and assuming that the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded, RM_I implements the timely reliable multicast service specification $\text{RM}_S(\Delta)$.

The proof of this claim involves showing that the relation R of Definition 5.1 is a timed forward simulation relation from RM_I to $\text{RM}_S(\Delta)$, under the aforementioned constraints and assumptions. The key part of the proof involves showing the correspondence of the time-passage steps. In particular, we show that active packets are delivered to all the hosts is their intended delivery sets within Δ time units.

Theorem 5.28 *Let $k \in \mathbb{N}^+$ and $\Delta = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Then, it is the case that $C\text{-}\mathit{attraces}_k(\text{RM}_I) \subseteq \mathit{attraces}(\text{RM}_S(\Delta))$.*

Proof: It suffices to show that the relation R of Definition 5.1 is a timed forward simulation relation from RM_I to $\text{RM}_S(\Delta)$, for any execution in the set $C\text{-}\mathit{attraces}_k(\text{RM}_I)$.

The proof that R is indeed a timed forward simulation relation is identical to that of Lemma 5.11 with the exception that in this case showing the correspondence of the time passage transitions is nontrivial.

Consider any discrete transition $(u, \pi, u') \in \mathit{trans}(\text{RM}_I)$, where $\pi = \nu(t)$, for some $t \in \mathbb{R}^{\geq 0}$, that occurs in any admissible execution of RM_I in the set $C\text{-}\mathit{attraces}_k(\text{RM}_I)$. It suffices to show that, for any reachable state s of $\text{RM}_S(\Delta)$ such that $(u, s) \in R$, there exists a timed execution fragment α of $\text{RM}_S(\Delta)$ such that $\alpha.\mathit{fstate} = s$, $\alpha.\mathit{lstate} = s'$, $\mathit{ttrace}(\alpha) = \mathit{ttrace}(u\pi u')$, the total amount of time-passage in α is the same as the total amount of time-passage in $u\pi u'$, and $(u', s') \in R$.

Let s be any reachable state of $\text{RM}_S(\Delta)$ such that $(u, s) \in R$. The timed execution fragment of $\text{RM}_S(\Delta)$ corresponding to the step (u, π, u') is comprised solely of the $\nu(t)$ action. We must show that the $\nu(t)$ action is enabled in s; that is, we must show that, for any active packet $p \in s.\mathit{active\text{-}pkts}$, it is the case that either $s.\mathit{now} + t \leq s.\mathit{trans\text{-}time}(p) + \Delta$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$. Since $(u, s) \in R$, it suffices to show that, for any active packet $p \in u.\mathit{active\text{-}pkts}$, it is the case that either $u.now + t \leq u.\mathit{trans\text{-}time}(p) + \Delta$ or $u.\mathit{intended}(p) \subseteq u.\mathit{completed}(p)$.

Consider any active packet $p \in u.\mathit{active\text{-}pkts}$. It suffices to show that if $u.\mathit{trans\text{-}time}(p) + \Delta < u.now + t$, then $u.\mathit{intended}(p) \subseteq u.\mathit{completed}(p)$. Let $h \in H$ be any host in $u.\mathit{intended}(p)$. Since the action $\nu(t)$ of RM_I does not affect the derived history variable SRM.*intended*(p), it follows that $h \in u'.\mathit{intended}(p)$. Moreover, since $u.\mathit{trans\text{-}time}(p) + \Delta < u.now + t$ and the action $\nu(t)$ increments the *now* variable by t time units, it follows that $u.\mathit{trans\text{-}time}(p) + \Delta < u'.now$. Since $\Delta = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $u.\mathit{trans\text{-}time}(p) + \Delta < u'.now$, and $h \in u'.\mathit{intended}(p)$, Lemmas 5.20 and 5.27 imply that $h \in u'.\mathit{completed}(p)$. Since the action $\nu(t)$ of RM_I does not affect the derived history variable SRM.*completed*(p), it follows that $h \in u.\mathit{completed}(p)$. ■

6 Contributions & Future Work

The contributions of this paper are several. First, we present a timed I/O automaton model of the reliable multicast service. This model formally specifies the behavior of several reliable multicast protocols that strive to provide eventual delivery with, possibly, some timeliness guarantees. In particular, it dictates what it means to be a member of a reliable multicast group and which packets are guaranteed delivery to which members of the reliable multicast group. Moreover, we present a timed I/O automaton model of the SRM protocol. This model decomposes the functionality of the reliable multicast service, thus facilitating reasoning and the future modeling of either variations and extensions to SRM's recovery scheme, or other reliable multicast protocols altogether. We show that our model of SRM is safe, in the sense that it may only deliver appropriate packets to each member of the reliable multicast group. We also show that, under certain constraints, our implementation is live, in the sense that it guarantees the timely delivery of the appropriate packets to each member of the reliable multicast group.

In the future, we intend to relax the constraints used in our liveness analysis of SRM and to analyze the performance of SRM in the context of a dynamic group membership. We also intend to model, analyze, and compare the performance of extensions to SRM and other reliable multicast protocols. The safety analysis of each such protocol will guarantee that the protocols are compared on an equal footing; something rarely done precisely when comparing protocols.

Acknowledgments

We thank Idit Keidar for helpful comments and suggestions.

References

[1] Floyd, S., Jacobson, V., McCanne, S., Liu, C.-G., and Zhang, L. A Reliable Multicast Framework For Light-Weight Sessions And Application Level Framing. *IEEE/ACM Transactions on Networking 5*, 6 (Dec. 1997), 784–803.

[2] Holbrook, H. W., Singhal, S. K., and Cheriton, D. R. Log-Based Receiver-Reliable Multicast For Distributed Interactive Simulation. In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM Special Interest Group on Data Communication (ACM/SIGCOMM'95)* (1995), ACM Press, New York, pp. 328–341.

[3] Li, D., and Cheriton, D. R. OTERS (On-Tree Efficient Recovery using Subcasting): A Reliable Multicast Protocol. In *Proc. 6th IEEE International Conference on Network Protocols (IEEE/ICNP'98)* (Austin, Texas, 1998), pp. 237–245.

[4] Lin, J. C., and Paul, S. RMTP: Reliable Multicast Transport Protocol. In *Proc. 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation (IEEE/INFOCOM'96)* (San Francisco, CA, Mar. 1996), vol. 3, pp. 1414–1424.

[5] Liu, C.-G., Estrin, D., Shenker, S., and Zhang, L. Local Error Recovery in SRM: Comparison of Two Approaches. *IEEE/ACM Transactions on Networking 6*, 6 (Dec. 1998), 686–692.

[6] Lynch, N. A. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[7] Papadopoulos, C., Parulkar, G., and Varghese, G. An Error Control Scheme For Large-Scale Multicast Applications. In *Proc. 17th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE/INFOCOM'98)* (San Francisco, CA, Mar. 1998), vol. 3, pp. 1188–1196.

[8] Paul, S., Sabnani, K. K., Lin, J. C., and Bhattacharyya, S. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications 15*, 3 (Apr. 1997), 407–421.