Scalable Timers for Soft State Protocols

Puneet Sharma* **Deborah Estrin*** **Sally Floyd†** **Van Jacobson†**
Information Sciences Institute
University of Southern California
puneet@catarina.usc.edu, estrin@usc.edu

Lawrence Berkeley National Laboratory
floyd@ee.lbl.gov, van@ee.lbl.gov

Abstract

Soft state protocols use periodic refresh messages to keep network state alive while adapting to changing network conditions; this has raised concerns regarding the scalability of protocols that use the soft-state approach. In existing soft state protocols, the values of the timers that control the sending of these messages, and the timers for aging out state, are chosen by matching empirical observations with desired recovery and response times. These fixed timer-values fail because they use time as a metric for bandwidth; they adapt neither to (1) the wide range of link speeds that exist in most wide-area internets, nor to (2) fluctuations in the amount of network state over time.

We propose and evaluate a new approach in which timer-values adapt dynamically to the volume of control traffic and available bandwidth on the link. The essential mechanisms required to realize this **scalable timers** *approach are: (1) dynamic adjustment of the senders' refresh rate so that the bandwidth allocated for control traffic is not exceeded, and (2) estimation of the senders' refresh rate at the receiver in order to determine when the state can be timed-out and deleted. The refresh messages are sent in a round robin manner not exceeding the bandwidth allocated to control traffic, and taking into account message priorities. We evaluate two receiver estimation methods for dynamically adjusting network state timeout values: (1) counting of the rounds and (2) exponential weighted moving average.*

1 Introduction

A number of proposed enhancements to the Internet architecture require addition of new state (i.e., stored information) in network nodes. In the context of various kinds of network element failures, a key design choice is the manner in which this information is established and maintained. Soft state protocols maintain state in intermediate nodes using refreshes that are periodically initiated by endpoints. When the endpoints stop initiating refreshes the state automatically times out. Similarly, if the intermediate state disappears, it is re-established by the end-point initiated refreshes [1].

In this paper we propose a new approach of scalable timers to improve the scaling properties of soft state mechanisms. Scalable timers replace the fixed timer settings used by existing soft state protocols with timers that adapt to the volume of control traffic and available bandwidth on the link. Scalable timers regulate the amount of control traffic independent of the amount of soft state.

In the next section we present an overview of state management in networks. In Section 3 we describe how the fixed timers are currently used by soft state protocols to exchange refresh messages and discard stale state and we motivate our approach of scalable timers that makes soft state protocols more scalable. Mechanisms required for the proposed approach of regulating control traffic are discussed in Section 4 and Section 5. We look at the application of the scalable timers approach to PIM [2, 3] in Section 6, followed by the simulation results in Section 7. Section 8 compares the traditional and the proposed approaches. We conclude with a summary and a few comments on future directions in Section 9.

2 State Management in Networks

State in network nodes refers to information stored by networking protocols about the conditions of the network. The state of the network is stored in a distributed manner across various nodes and various protocols. For instance, a teleconference application might run on top of multiple protocols. Internet Group Management Protocol (IGMP) [4] in a host stores information about the multicast groups in which that host is participating. Based on the membership information multicast routing protocols such as DVMRP [5], PIM [2, 3] or CBT [6] create multicast forwarding state in the routers. A reservation protocol such as RSVP [7] or ST-II [8] may reserve network resources for the teleconferencing session along the multicast tree.

The state has to be modified to reflect the changes in network conditions. The network nodes communicate with each other to exchange the information regarding change in the network conditions. Based on these control messages the network nodes modify their stored state. For instance, in the event of a change in network topology, the routers exchange messages resulting in modification of the multicast forwarding state if needed.

Based on the roles played by the nodes with respect to the particular state being referenced, the control message exchange among network nodes can be modelled as an exchange of messages between two entities; the *sender* and the *receiver*. The *sender* is the network node that (re)generates control messages to install, keep alive, and

*P. Sharma and D. Estrin were supported by the National Science Foundation under Cooperative Agreement NCR-9321043.

†S. Floyd and V. Jacobson were supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

remove state from the other node. The *receiver* is the node that creates, maintains and removes state, based on the control messages that it receives from the sender.

2.1 Paradigms for maintaining state in network

The state maintained by nodes in a network can be categorized as *hard* state and *soft* state. *Hard* state is that which is installed in nodes upon receiving a set-up message and is removed only on receiving an explicit tear-down message. The reservation protocol ST-II [8, 9] and multicast routing protocol Core Based Tree (CBT)[6] are examples of protocols that use the hard state paradigm. Hard state architectures use explicitly acknowledged, reliable messages. This reliable message transport entails increased protocol complexity.

Soft state, on the other hand, uses refresh messages to keep it alive and is discarded if the state is not refreshed for some time interval. Resource Reservation Protocol (RSVP) [7] and Protocol Independent Multicast (PIM) [2, 3] use periodic refreshes to maintain soft state in network nodes. The refreshes are sent periodically after one *refresh period*. The time that the receiver waits before discarding a state entry is a small multiple of the refresh period[1]. The multiplying factor is determined by the degree of robustness required and lossiness of the link or path[2].

During steady conditions, hard state protocols require less control traffic as there are no periodic control messages. Soft state protocols provide better (faster) adaptation and greater robustness to changes in the underlying network conditions, but at the expense of periodic refresh messages. However, when the network is dynamic, hard-state control messages must be generated to adapt to the changes. In such a scenario the bandwidth used by control messages would be comparable for both paradigms.

3 Timers in Soft State Protocols

Soft state protocols have two timers associated with the control traffic, one at the sender and one at the receiver. The sender maintains a *refresh timer* that is used to clock out the refresh messages for the existing state. The receiver discards a state entry if it does not receive a refresh message for that state before the *state timeout timer* expires.

Traditional protocols have fixed settings for the timer values. The values of the refresh and timeout timers are chosen by empirical observations with desired recovery and response in mind. These values are then used as fixed timers for sending the periodic refreshes. Such fixed refresh timers for the state update fail to address the heterogeneity of the networking environments (e.g. range of link bandwidths) and the growth of network state.

As the size and usage of the networks grow the amount of state to be maintained also increases. State in the network might exist even when data sources are idle (e.g. state related to shared trees in PIM routers, or IGMP membership information in Designated Routers). With fixed timers the control traffic grows linearly with the increase in the amount of state in the network. In the center of the network aggregate traffic levels will contribute to large quantities of state and traffic. At the edges, lower speed links make even a lower traffic level a concern. The primary goal of a data network is to carry data traffic. Unconstrained growth of control traffic can jeopardize this primary goal. This is exacerbated by the fact that traffic levels are higher when congestion and network events are reducing overall network resource availability. Soft state protocols can be made scalable only if control messages can be constrained independent of the amount of state in the network.

To date when the overhead becomes excessive the fixed timer values have to be changed globally. In summary, fixed timer settings fail because they use time as a metric for bandwidth and do not address the range of link speeds present across the network.

3.1 Scalable Timers

Our *scalable timers* approach fixes the **control traffic bandwidth** instead of the **refresh interval**. The control traffic is regulated by dynamically adjusting timer values to the amount of state, using mechanisms similar to those for regulating control traffic in RTP [10]. The design objective is to make the bandwidth used by control traffic negligible as compared to the link bandwidth and the data traffic. Consequently, a fixed portion of the link bandwidth is allocated to the control traffic. The sender adjusts the refresh interval according to the fixed available control traffic bandwidth and the amount of state to be refreshed. Because the refresh interval is no longer fixed, the receiver has the additional job of estimating the refresh rate used by the sender.

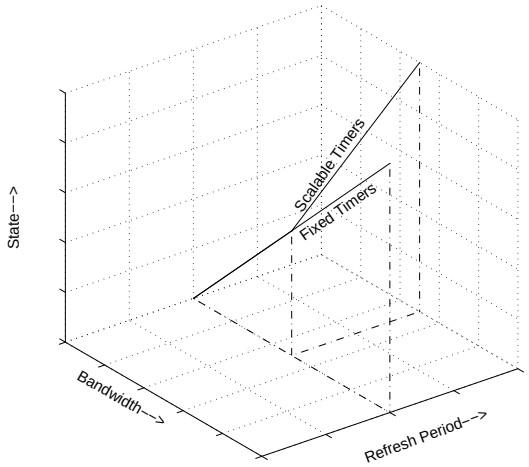

Figure 1: **Fixed Timers v/s Scalable Timers**

Figure 1 compares the changes in the refresh interval and control bandwidth for fixed and scalable timers. Figure 1 has a line for the scalable timers approach and a line for the fixed timers approach, showing how the amount of bandwidth used by control messages and refresh period vary as the amount of state to be refreshed increases. In the fixed timers approach the amount of bandwidth used by control messages increases as the amount of state to be refreshed

[1] We assume for simplicity that the soft state protocols do not use explicit tear-down messages. A soft state protocol might optionally use explicit tear-down messages to achieve faster action.

[2] Degree of robustness is a tradeoff of tolerance to dropped packets and overhead of maintaining state that may no longer be required.

increases. However, in our scalable timers approach the refresh interval increases with the increase in amount of state, keeping the volume of control traffic constant. Scalable timers address the wide range of link speeds as the bandwidth allocated to control traffic is proportional to the link speed. In the next section we present the mechanisms required for applying scalable timers to soft state protocols.

4 Mechanisms for Scalable Timers

The essential mechanisms required by scalable timers are twofold: (1) the *sender* dynamically adjusts the refresh rate, and (2) the *receiver* estimates the rate at which the sender is refreshing the state, in order to determine when state can be considered stale and deleted.

In this section we discuss these mechanisms for servicing the control traffic and timing out state at network nodes for point-to-point links. These mechanisms can be extended to multiaccess LANs by treating each of them as multiple point-to-point links[3].

4.1 Servicing the control traffic at the sender

The sender needs to generate refreshes for its state entries such that the bandwidth allocated for control traffic is not exceeded. A simple model for generating refreshes at the sender is to equally divide the allocated control bandwidth ($control\_bw_{total}$) among all the state entries by servicing them in a round robin fashion. We need to modify this simple model to accommodate control messages with different priorities. For instance, control messages that are generated to create new state (trigger traffic) should be serviced faster than control messages that refresh already existing state (refresh traffic). Simple round robin servicing of the control messages fails to include such priorities associated with various control messages for a protocol.

However, a simple priority model is also not suitable, as it can lead to starvation of the lower priority messages. Instead, isolation of bandwidth for various classes of traffic is required, such that each class receives a guaranteed share of the control bandwidth even during heavy load. There is no starvation as each class of control traffic is guaranteed a share of the control bandwidth. When bandwidth allocated to a class is not being used, it is available to other classes sharing the control traffic bandwidth. The class structure of the control messages in different protocols can be set differently based on the requirements of that protocol. This use of isolation to protect the lower classes from starvation is similar to the link sharing approach in Class based Queueing (CBQ) [11].

One possible class structure divides the control messages into two classes, *trigger messages* and *refresh messages*. The trigger traffic class is guaranteed a very large fraction (close to 1) of the control bandwidth so that trigger messages can be serviced very quickly for better response time. Refresh messages are allocated the remaining *non-zero* fraction of bandwidth. Guaranteeing a non-zero fraction of the control traffic bandwidth to refresh messages precludes starvation and premature state timeout at the receiver. In this class structure there is a tradeoff between the latency in establishment of new state and delay in removing stale state. If a larger fraction of control bandwidth is allocated to trigger traffic the latency in establishing new state is less, but there might be more delay in discarding stale state.

A token bucket rate limiter can be used to serve bursts in trigger traffic. The sender generates refreshes for the existing state in a round robin fashion. To avoid sending refresh messages too frequently all protocols have a minimum refresh period $refresh\_interval_{min}$ that is the same as the refresh period used by the fixed timers approach.

4.2 Timing out network state at the receiver

In the fixed timer approach, the receiver has prior knowledge of the refresh period and can compute the timeout interval a priori (e.g., a small multiple of the refresh period to allow for packet loss). In contrast, in the scalable timers approach the rate at which the sender refreshes the state varies based on the total amount of state at the sender, and the receiver has to track the refresh frequency and update the timeout interval accordingly. One possibility would be for the sender to explicitly notify the receiver about the change in the refresh period in the control messages. Another possibility is for the receiver to estimate the refresh frequency from the control message arrival rate.

If the receiver relies on information conveyed by the sender, the receiver is implicitly putting trust in the sender to behave properly. This is not an issue of maliciousness but of lack of adequate incentives to motivate strict adherence. In addition, the sender cannot foresee the changes in the refresh period and hence might convey incorrect information to the receiver. For example, when a link comes up there could be a big burst of trigger traffic resulting in changes in refresh intervals not foreseen by the sender. Therefore, even if information is sent explicitly by the sender, the receiver still needs alternate mechanisms to avoid premature state deletion. If the receiver does not trust the sender and has its own mechanism for the estimation of the timeouts, then the information sent by the sender is redundant.

In summary, the explicit notification approach unnecessarily couples system entities. We adopt a general architectural principle to place less trust in detailed system behavior in order to make the architecture more robust.

In the absence of explicit notification of change in the refresh interval, the receiver needs to estimate the current refresh interval and adjust the timeout accordingly. In the next section we discuss mechanisms for estimation of the refresh interval at the receiver.

5 Estimating the Refresh Periods

The receiver estimates the refresh period from the time between two consecutive refresh messages and must adapt to the changes in this refresh period over time. The receiver might observe changes in the refresh interval due to trigger messages, new state and packet loss.

Unlike the fixed refresh period approach, where the timeout interval has to be robust only to deal with dropped packets, the estimator used in scalable timers has to be more carefully designed to be both adaptive and efficient. We need timers to be robust not only in the presence of an occasional dropped update, but also in the presence of a rapid

[3] It might require additional mechanisms for sharing the bandwidth.

non-linear increase in the inter-update interval.

Receivers use the estimate of the refresh interval for discarding old state. The estimate must be conservative so as not to timeout state prematurely. The consequences of timing out state prematurely are generally more serious than the consequences of moderate delays in timing out state.

In this section we discuss two approaches that can be used at the receiver to age the old state.

5.1 Counting of the Rounds

The first approach that we discuss is *counting of the rounds*. A *round* refers to one cycle of refreshes at the sender for all the state that it has to refresh. In this approach, instead of trying to estimate the refresh period and then using a multiple of this period to throw away the state, the state is thrown away if the receiver does not receive the refresh for a particular state for some a rounds (where a is a small integer and is set to provide robustness to lost packets, as with the fixed timers). Thus by counting the rounds the receiver can track the sender closely.

When the sender is servicing the states in a round robin fashion, all the state will have been serviced by the sender at least once between any two consecutive refreshes for a particular state entry. The receiver marks the beginning of a round by a *round marker*. For each state the receiver maintains the last time that a refresh was received for that state. If two refresh messages are received for a particular state since the beginning of the current round, a new round marker is set to the current time. This marks the completion of a new round, during which the sender should have sent at least one refresh message for each state to be refreshed.

The receiver maintains round markers for each link. Every time a new round starts, checks are made to see if any state has to be discarded. The receiver can count the rounds easily using this algorithm. The algorithm is further explained in the pseudocode below.

For state i, let previous_refresh[i] denote the last time that a refresh was received for state i.

```
On receiving a refresh message for state i:
  if (previous_refresh[i] ≥ last_round_marker) {
        last_round_marker ← current_time;
        shift_round_markers();
        age_state();
  }
  previous_refresh[i] ← current_time;
```

The age_state() routine discards any state that has not been refreshed for last three rounds.

With the counting-the-rounds approach, the receiver is essentially assuming that the sender is using round robin for sending refresh messages. Refinements to the basic algorithm to address the problems associated with faulty senders, interoperability, and end cases are described in [12]. In the next section we discuss Exponential Weighted Moving Average for estimating the refresh period at the receiver.

5.2 Exponential Weighted Moving Average

The second approach that we consider is use of Exponential Weighted Moving Average (EWMA) estimators. The EWMA estimator is a low pass filter that has been used widely for estimation (e.g. TCP [13] round trip time).

In this scheme a network node runs one EWMA estimator for each coincident link. Each EWMA estimator tracks the refresh interval average ($interval_{avg}$) and mean deviation ($interval_{mdev}$) for the refreshes received on the associated link.

On receiving a new measurement for refresh interval I, the receiver updates the average of the refresh interval as:

$$interval_{avg} \leftarrow (1 - w) * interval_{avg} + w * I,$$

where w is the smoothing constant of the estimator.

As shown in [14], EWMA can be computed faster if it is rearranged as:

$$interval_{avg} \leftarrow interval_{avg} + w * (err),$$

where $err = I - interval_{avg}$.

Mean deviation is computed similarly as:

$$interval_{mdev} \leftarrow interval_{mdev} + w * (|err| - interval_{mdev}).$$

Once the average and mean deviation for the refresh interval are updated, the timeout value for the state is then set to:

$$a * (interval_{avg} + b * interval_{mdev}),$$

where a, b are both small integers. Here, a is the degree of robustness used for setting the timeout period in the fixed timers approach and b is the weight given to the refresh interval deviation in computing the timeout period.

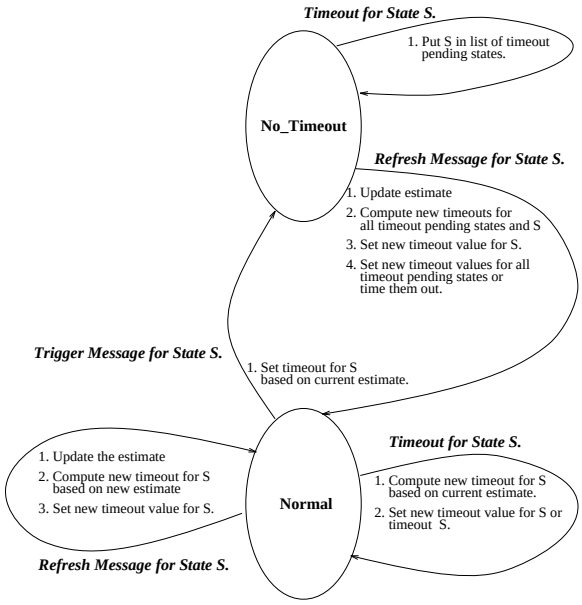

Figure 2: **State Diagram for an EWMA receiver**

The use of mean deviation in computing the timeout period allows the estimator to track sharp increases in the refresh interval. This prevents the receiver from deleting state prematurely when there are sudden increases in the refresh interval. When a state timeout occurs, a new timeout value is computed based on the current values of $interval_{avg}$ and $interval_{mdev}$. If the new timeout value is not greater than the previous value, the state is discarded. Otherwise the new value for state timeout is set.

In the event of long bursts of trigger traffic, the estimator does not receive any refreshes and the estimate is not updated. In such a scenario, the estimator lags behind the actual refresh interval and a premature timeout of state might occur. To avoid this problem, the receiver does not timeout state during a burst of trigger traffic. The receiver has two modes of operation as shown in the state diagram for the receiver in the Figure 2. While the receiver is not receiving any trigger messages it stays in the *Normal* mode. It enters the *No_Timeout* mode when it receives a trigger message. The mode of the receiver is changed back to *Normal* upon receiving a refresh message.

For the EWMA estimator the value of the smoothing constant needs to be determined. The smoothing constant should track the refresh period reasonably closely without overreacting to occasional long refresh intervals that result from either bursts of trigger traffic or occasional drops of refresh packets. At the same time it should be able to respond quickly to increases in the refresh period that result from an increase in the amount of state.

In this and the previous sections we have discussed our approach of scalable timers and the required mechanisms. In the coming sections we present our simulation studies of scalable timers for PIM control traffic.

6 Using Scalable Timers for PIM Control Traffic

Protocol Independent Multicast(PIM) [2, 3] is a multicast routing protocol. It uses soft state mechanisms to adapt to the underlying network conditions and multicast group dynamics. Explicit hop-by-hop *join* messages from members are sent towards the data source or Rendezvous Point. In steady state each router generates periodic join messages for each piece of forwarding state it maintains. These periodic join messages are examples of the refresh traffic discussed earlier. Besides the exchange of refresh messages for the existing state, PIM has trigger traffic due to membership and network topology changes. A join message can be triggered by: i) a change in group membership, ii) a new data source to a group, iii) a switch from a shared tree to a source-specific tree and vice versa, iv) a topology change such as partitions and network healings, and v) a change in Rendezvous Point reachability. A multicast forwarding entry is discarded if no join messages to refresh this entry are received for the timeout interval.

Members of a group graft to a shared multicast distribution tree centered at the group's Rendezvous Point. Though members join source-specific trees for those sources whose data traffic warrants it, they are always joined to the shared tree listening for new senders. Thus groups that are active but have no active senders still need to refresh the state on the shared tree. Consequently, in PIM there is some minimal control traffic for every active group even if there are no active senders to the group. So the bandwidth used by control traffic is not a strict function of the data traffic being carried by the network. Currently the PIM join messages are sent at a fixed interval of 60 seconds, and the PIM control traffic grows with the amount of the multicast forwarding state maintained in the routers.

It is important to give higher priority to trigger traffic than refresh traffic because delay in servicing trigger traffic can increase the join and leave latency of a group. Based on data traffic being sent to a group, multicast groups can be of two types. A multicast group is said to be *data-active* at some time if there are senders that are sending data to the group. It is more important to refresh the state related to the data-active groups as compared to the data-inactive groups, as the data-inactive groups suffer less, or no, data loss while adapting to changes.

Links with less bandwidth can not support a large number of simultaneous conversations. Link bandwidth naturally limits the number of source specific state entries needed for a multicast group on each link. Hence the scaling with respect to the source specific state is not of major concern. We analyzed the number of multicast groups that can be supported on various links without inflating the refresh intervals by a large amounts [12]. For instance, 1% bandwidth of an ISDN link can support 150 groups with the refresh period of 60 seconds. Similarly, a T1 link can support 15000 groups with 1% of link bandwidth. For conducting our experiments we used 1% of link bandwidth for PIM control traffic

6.1 Class structure at the sender

At the sender we divided PIM control traffic into two classes, trigger traffic and refresh traffic, with higher priority given to the trigger traffic. A token bucket is used to control the bursts in trigger traffic. This class structure has two parameters, the fraction of the PIM control bandwidth for trigger traffic, and the size of the token bucket for trigger traffic. Since trigger traffic needs to be serviced as fast as possible, we allocated 99% of control traffic bandwidth to trigger traffic.

If the size of the token bucket is large, the sender can service bigger bursts of trigger traffic without increasing the join/leave latency. Because there is a high level of correlation among multicast groups, bursts of trigger traffic are not uncommon. For instance, groups created for audio and video streams of the same session are coupled together. Similarly there can be multiple groups for sending hierarchical video streams that are coupled with each other and exhibit related dynamics. The bucket size should be large enough to accommodate these correlations.

Another possible refinement that we have not investigated would be to service control traffic related to data-active and data-inactive groups in different classes.

6.2 Aging out the multicast state at the receiver

Multicast routers age out the forwarding entries if they do not receive refresh messages for that entry. Delaying too long in timing out a forwarding entry can result in the following overheads:

- bandwidth and memory in maintaining soft state upstream for inactive trees, and
- multicasting application traffic downstream after it is no longer needed.

On the other hand, if the state is aged prematurely, it results in the failure of multicast routing to send data to downstream locations until the state is re-established. It is more

important to be careful not to timeout prematurely, than to be overly precise in timing out state as soon as possible.

As described in Section 4, two approaches can be adopted at the receiver node to discard soft state. The next section discusses the simulation studies conducted with the scalable timers for PIM.

7 Simulation Studies of Scalable Timers for PIM

An existing PIM simulator, *pimsim* [15] was modified to include the scalable timers approach for PIM control traffic. A sender module was included to schedule the PIM control messages based on the class structure described earlier. For a PIM receiver we implemented both the counting-the-rounds approach and exponential moving average estimator. We conducted simulations on a wide range of scenarios having topologies with different link bandwidths, loss rates and membership changes. The simulation runs were validated by feeding the packet traces generated from the simulations to the Network Animator *nam* [4]. In this section, we present only a few simple, yet representative, scenarios for understanding the behavior of scalable timers.

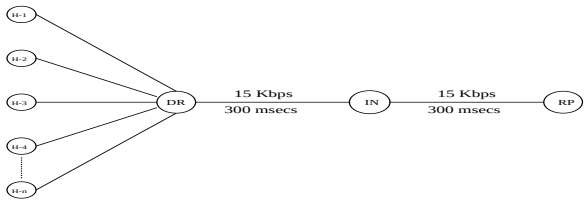

Figure 3: **Simple chain topology**

Since PIM messages are router to router, in this paper we study the behavior of mechanisms only on a single link. Figure 3 shows a chain topology with three PIM routers. The designated router *DR* sends join messages towards the Rendezvous Point *RP* for the groups that are active in the attached hosts. These join messages are received by the intermediate router *IN*. The studies presented here focus on the behavior of the sender at the *DR*. The receiver's behavior was studied at *IN*. We also studied end-to-end behavior on larger topologies.

In the simulations shown here the parameters were set so that the allocated control bandwidth is sufficient to refresh two state entries without exceeding the $refresh\_interval_{min}$ of 5 seconds. The size of the trigger token bucket used was 5. The bandwidth of the link between the *DR* and the node *IN* was 15 Kbps.

Change in the amount of state at a node can occur either in steps or in bursts. Simulation results presented here describe these two basic changes.

Scenario 1 : Step Change. In this case a new group becomes active at the hosts attached to *DR* every 100 seconds until 600 seconds. After 1000 seconds are over, one group expires every 100 seconds.

[4] *nam* is a animation tool for networking protocols being developed at LBNL.

Scenario 2 : Burst Change. In this scenario at 100 seconds 7 new groups become active. The burst size was chosen to be bigger than the token bucket size.

7.1 PIM Sender

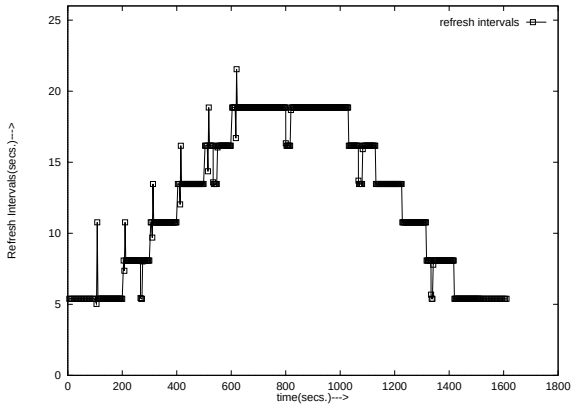

Figure 4: **Change in the refresh interval for step changes.**

In this section we demonstrate the sender behavior in various scenarios. If the link does not drop any packets then the actions of the sender can be captured by the refresh messages as seen by the receiver. In Figures 4 and 5 each refresh message has been marked by a "□". For each refresh message received, the x-axis shows the time that the message was received at the receiver, and the y-axis shows the time in seconds since the previous refresh message was received for that state.

Figure 4 illustrates the behavior of a sender in Scenario 1. When the amount of state is increased from one to two the refresh intervals do not change as enough bandwidth is available to refresh all the state within the minimum refresh period. As more state is created and needs to be refreshed, the refresh interval increases with each step. This is because the sender adapts to the increase in the amount of state by reducing the refresh frequency (i.e., increasing the refresh interval). As the amount of state decreases after 1000 seconds, we notice that refreshes are sent more frequently, consuming the available control bandwidth.

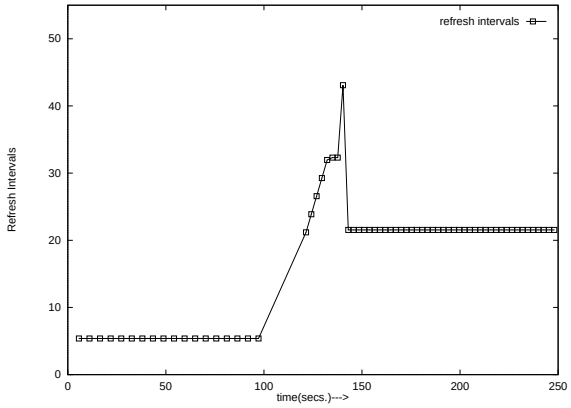

Figure 5: **Change in the refresh interval for trigger traffic bursts.**

Figure 5 shows the response of the sender in Scenario 2. In steady state all the control bandwidth is used for sending refresh messages. When a new group becomes active, trigger state tokens are used for servicing the associated trigger traffic. Spikes in the refresh intervals are observed when there is trigger traffic to be served. The height of the spike is a function of the amount of trigger traffic to be serviced and the size of the bucket. The sender sends the trigger traffic for all but two groups back to back at time 100. The trigger traffic for the remaining two groups is served as more trigger tokens are generated at time 103 and 106. No refresh packet is seen during this time[5].

One of the parameters that needs to be configured at the sender is the size of the trigger traffic token bucket. Trigger traffic suffers latency greater than the fixed timers approach only if the trigger traffic burst is greater than the tokens available in the token bucket. The larger the bucket size, the smaller is the associated join/prune latency. The worst case delay suffered by a trigger packet is given by: $burst\_size/control\_bw_{trigger}$. However, in most cases the token bucket is full and the delay suffered by a trigger packet in a burst larger than the bucket size $token\_bucket_{size}$ is given by:

$$(burst\_size - token\_bucket_{size})/control\_bw_{trigger}.$$

The end-to-end join/prune latency is the sum of the delays incurred by the trigger message at each node.

7.2 PIM Receiver

For the PIM receiver we studied both the counting-the-rounds and exponential weighted moving average approaches for aging state. For an EWMA receive, the value of the smoothing constant (w) is set to 0.05, and the weight for the interval mean deviation (b) is set to 4. The state timeout timer was set to three times the current estimate of the refresh period.

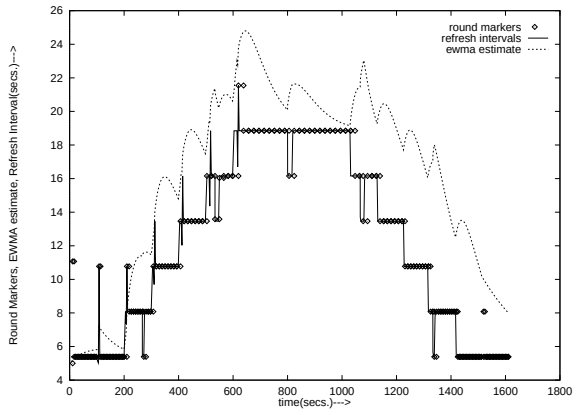

Figure 6: **Receiver response to step changes.**

We studied the receiver's tracking behavior during both step and burst changes in the amount of state. Figures 6 and 7 plot refresh interval estimates as observed by the receivers over time, for the Scenarios 1 and 2 respectively. In these figures, the "$\diamond$" shows the response of a counting-the-rounds receiver, and the dotted line shows the response of an EWMA receiver. For a counting-the-rounds receiver, each time the round marker is moved we plot "$\diamond$" to show the length of the previous round. For an EWMA receiver, for each refresh message received we plot the new refresh interval estimate ($= a * (interval_{avg} + b * interval_{mdev})$).

We observe from Figure 6 that the counting-the-rounds receiver is able to track the round times very closely during step changes. The EWMA receiver is also able to follow the changes in refresh interval at the sender. The rate of convergence for EWMA is dependent on the amount of state being refreshed. If the amount of state is large, the EWMA estimator has more observations of the refresh interval in one round, and therefore converges faster to the refresh interval. For the scenario in Figure 6 no premature timing out of state was observed with either of the two approaches.

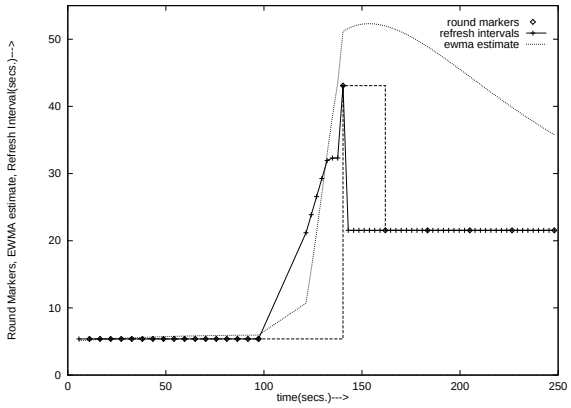

Figure 7: **Receiver response to trigger traffic bursts.**

Figure 7 depicts the tracking of round times during the burst of trigger traffic in Scenario 2. The counting-the-rounds approach advances the round markers only when refreshes are received. Hence, it is able to track the round times during bursts of trigger traffic. As shown in the Figure 7, the EWMA estimator can underestimate the refresh intervals when there are big bursts of trigger traffic (because the refresh interval suddenly increases). However, no state is timed out in this period as the receiver switches to the *No_Timeout* mode.

We also simulated scenarios with faulty senders and lossy links. The counting-the-rounds receiver is able to filter out the duplicates of the refresh messages [12]. Neither of the receivers prematurely timeout state on lossy links [12].

In the simulation studies we observed that the counting-the-rounds receiver is able to respond to the events at the sender and track the round times more precisely than a receiver running an EWMA estimator. However, the counting-the-rounds receiver fails to follow a non-round robin sender successfully. If the sender generates refreshes in an arbitrary order the EWMA estimator can also underestimate the refresh interval. However, in such scenarios EWMA estimator is more conservative than the counting-the-rounds receiver.

Both approaches *work* in that they allow the receiver to

[5]The figures only show refresh messages. Trigger messages have not been shown but can be inferred from the absence of refresh messages for a period of time.

adapt to changes at the sender; consequently, there is no clear winner. Since the mechanism for aging state is local to a node, different nodes can use different approaches. There is no need for standardizing the approach across the network as a whole.

8 Scalable Timers as compared to Fixed Timers approach

We can evaluate the fixed timers and our scalable timers approaches along three dimensions: control bandwidth, responsiveness and complexity.

Control Bandwidth: In the traditional fixed timers approach the bandwidth used by control traffic grows with the amount of state. As shown in Figure 1 scalable timers adapt to the growth in the amount of state to be refreshed and limit the control traffic bandwidth.

Responsiveness: The refresh interval determines the response time of the protocol to the changes in the network conditions. When the refresh interval is smaller the protocol adapts faster to the changes. The responsiveness in soft state protocols concerns initiation of new state and timing out the old state. Since the trigger messages are served instantaneously, the responsiveness of the protocol using scalable timers for control traffic is comparable to the fixed timers approach. The increase in the refresh interval in scalable timers might cause longer delay in aging out old state than in the fixed timers approach. However, in most cases the lifetime of a state entry is considerably longer than the period for which the state is kept after the last refresh. So the overhead in bandwidth used due to late timing out of old state (longer response time) should be negligible compared to the savings in the bandwidth used by control traffic over the lifetime of the state.

Complexity and Overhead: The additional mechanisms required for scalable timers are simple and introduce very small memory and computation overhead to the protocol. Moreover, the scalable timers module is generic and is not closely coupled with the protocol. The scalable timers approach does not require any changes to be made to the protocol.

9 Conclusion and Future Directions

In this paper we presented a scalable approach for regulating control traffic in soft state protocols. With scalable timers the control traffic consumes a fixed amount of bandwidth by dynamically adjusting the refresh interval. Scalable timers require mechanisms at the sender and the receiver of the control messages. Through simulations we have illustrated the effectiveness of scalable timers in terms of overhead reduction with minimal impact on protocol responsiveness; moreover there is only a little increase in complexity. We plan to further study the effect of changes in the parameters, such as class structure, and smoothing constant etc., on the performance of the proposed mechanisms.

We have studied scalable timers in the context of a multicast routing protocol (i.e. PIM), which is an example of a router-to-router protocol. Multiparty, end-to-end protocols were actually the first protocols to incorporate the notion of scalable timer techniques for reduction of periodic traffic (e.g., wb, RTP [10]). Future work will investigate the applicability of the detailed mechanisms discussed in this paper to such end-to-end protocols and soft-state protocols in general.

References

[1] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 106–114. ACM, August 1988.

[2] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol Independent Multicast (PIM) : Motivation and Architecture. *Internet Draft*, March 1996.

[3] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol Independent Multicast (PIM), Sparse Mode Protocol : Specification. *Internet Draft*, March 1996.

[4] William C. Fenner. Internet Group Management Protocol, Version 2. *Internet Draft*, May 1996.

[5] D. Waitzman S. Deering, C. Partridge. Distance Vector Multicast Routing Protocol, nov 1988. RFC1075.

[6] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT). In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 85–95, San Francisco, California, September 1993. ACM.

[7] Lixia Zhang, Bob Braden, Deborah Estrin, Shai Herzog, and Sugih Jamin. Resource Reservation Protocol (RSVP) – Version 1 Functional Specification. Internet Draft, Xerox PARC, October 1993. Work in progress.

[8] C. Topolcic. Experimental Internet Stream Protocol: Version 2 (ST-II). Request for Comments RFC1190, Internet Engineering Task Force, October 1990.

[9] Danny J. Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang. An Architectural Comparison of ST-II and RSVP. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Toronto, Canada, June 1994.

[10] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. Internet draft (work-in-progress), November 1995.

[11] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[12] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable Timers for Soft State Protocols. *ftp://catarina.usc.edu/pub/puneetsh/papers/CS-TR-96-640.ps*, University of Southern California, Technical Report TR96-640, June 1996.

[13] J. Postel. Transmission Control Protocol. Request for Comments (Standard) STD 7, RFC 793, Internet Engineering Task Force, September 1981.

[14] Van Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review*, 18(4):314–329, August 1988. Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[15] Liming Wei. The Design of the USC PIM Simulator (pimsim). Technical report, TR95-604, University of Southern California, June 1995.