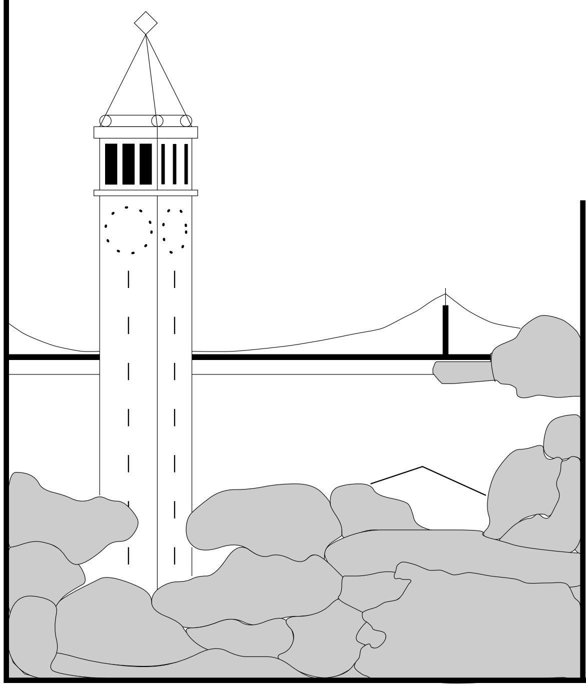
Generalized Data Naming and Scalable State Announcements for Reliable Multicast

Suchitra Raman and Steven R. McCanne



Report No. UCB/CSD-97-951

June 1997

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Generalized Data Naming and Scalable State Announcements for Reliable Multicast*

Suchitra Raman and Steven R. McCanne

June 1997

Abstract

Traditional ARQ-based reliable protocols for unicast (e.g., TCP) as well as multicast (e.g., Horus [24], RMTP [15], etc.) use sequential numbering of data units and detect losses from discontinuities in the sequence of received packets. The Application Level Framing (ALF) [7] model encourages application control over loss-detection and recovery. With sequence numbers, the application must express its reliability requirements using sub-sequences of the sequence space. This is both cumbersome and restrictive for applications that have no *a priori* knowledge of the data stream. Distributed whiteboard applications, webcast, and file system multicasting are some examples of applications where data is continuously generated and receivers cannot predict which sub-sequences must be received reliably.

In this paper, we propose an alternative data naming scheme which enhances the expressibility of applications' reliability and ordering requirements. We apply the new data naming scheme to build a framework for light-weight Scalable, Reliable Multicast (SRM) sessions, and develop a State Announcement Protocol (SAP) for loss detection. Using simulations, we study the scaling behavior of SAP and show that the protocol scales well for large group sizes. We also suggest heuristics for certain classes of applications that improve the convergence times and message complexity of the protocol.

1 Introduction

Widespread deployment of IP multicast [9] and the MBone [10] has made it possible to carry on large-scale multi-point communication. Applications like digital audio, video, and whiteboard are gaining popularity as desktop conferencing tools, as are other emerging applications including webcasting [13], file system multicast, distributed interactive simulation [22], network games [5], etc. All these applications have common requirements – scalable, reliable, flexible multipoint communication. However, each application differs significantly from the other in delay requirements, bandwidth characteristics, ordering guarantees, and degree of reliability.

*This work was supported by ARPA contract N66001-96-C-8508

In an earlier paper, Clark and Tennenhouse [7] predicted that numerous, heterogeneous distributed applications would emerge and protocol designers would have to design network protocols with this in mind. Their solution to overcome heterogeneity in applications was to involve the application in protocol decision making, which they called the *Application Level Framing* or ALF model. Their work stressed the need for a presentation layer, but left it as an open issue to design, implement and evaluate a generic system for achieving this.

In retrospect, a transport protocol like TCP, whose designers adopted the *one size fits all* approach, has had to be retrofitted against each new type of network technology ranging from wireless [2] to satellite links, and each new application, e.g., T-TCP [3] for WWW transport. The ALF or framework approach has been suggested for reliable multicast [12]. While previous work on the scalable, reliable multicast protocol suggested an architecture for a framework-based approach, no comprehensive set of mechanisms was specifically proposed. In this work, we propose *generalized data naming*, a key component of the ALF model.

The rest of this paper is organized as follows: Section 2 explains the need for generalized naming in the ALF model. The details of the naming scheme are described in Section 3. Section 4 shows how generalized data naming can be used in a framework for reliable multicast. Extensions to the scalable, reliable multicast protocol are discussed in 5. Section 6 presents simulations of the new algorithm under various scaling scenarios. We survey related work in 7 and conclude in 10.

2 ALF and Data Naming

In traditional transport protocols such as TCP for unicast and RMTP [15] for multicast, the data naming is tightly coupled with the error control. A sequence number is the minimum amount of information that is required for error detection and recovery in the absence of any other information from the application. However, the ALF model recommends application participation in error detection and recovery. For applications to express their reliability and ordering requirements, simple sequence numbers are often cumbersome and sometimes insufficient.

For example, consider the problem of file system mirroring, where updates are made to a master copy and need to be propagated to all the mirrors. The current solution is an offline approach that involves collapsing a structured file system into a linear stream of bytes, using the UNIX utility `tar` or some equivalent, and unicasting the resulting stream to multiple mirrors. This method suffers from two defects: mirror copies can be out of date for periods of time depending on the frequency of updates. For continuously evolving file systems this delay is often undesirable. This method also makes inefficient use of bandwidth because it transmits an entire copy of the file system by unicast instead of multicasting the changes. A different approach to file system mirroring is an online, instantaneous approach using reliable multicast and selectively choose the portions of the file system that require updates or repair using a generic naming scheme.

Another, more recent, example of a multicast application that can benefit from application level participation in error detection and recovery is distributed whiteboard application. The LBL *wb* [16] application uses a two-level hierarchy of pages and drawing operations to structure data. In order to enhance scalability, the application participates in error detection and recovery.

In the context of receiver-driven reliable multicast protocols such as the Scalable Reliable Multicast (SRM) protocol, where the session data is assumed to exist in the session at any given time requires an unlimited amount of buffering in the transport protocol. ALF pushes this responsibility to the application and the transport protocol can query the application, using *upcalls* [6], to retrieve the missing data. *In order to enable application participation via upcalls, there is a need for a common syntax that both the application and the transport understand.*

3 Hierarchical Naming of Application Data Units

Hierarchical naming is sufficiently general for most applications of interest. We use filename- or URL-like names for each application data unit (ADU). Hierarchical naming allows data transmission from one part of the name space to be interspersed with transmission from another. With this extension to traditional reliable protocols, we need additional mechanism to detect and recover from errors. Each application can also map data units into the hierarchical data space supported by the transport protocol. This mapping is flexible and under application control. The new naming scheme provides a *common syntax* that both the application and the transport protocol can reason about. To see how hierarchical naming improves expressibility of application requirements and enables application control in protocol functions, consider the case when a MediaBoard [20] wants to selectively repair losses on the current active page(s). With the new naming scheme, expressing these requirements translates added flexibility to the application in responding to the transport protocol's notification of losses of data units other than the current active page.

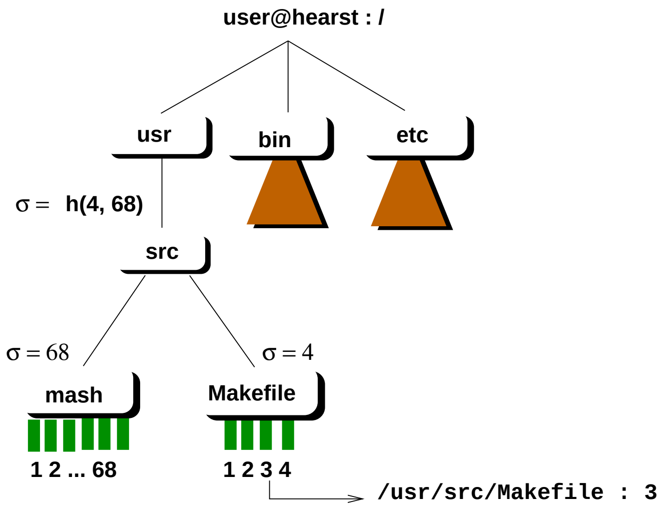


Figure 1: Example illustrating *containers*

3.1 Containers

Very often most applications of concern request reliable and ordered delivery at some granularity of data units. To provide a default behavior for such applications, the name space allows *containers* which are ordered sequences of data blocks, at the lowest-level of the hierarchy. Figure 1 illustrates this concept.

We also associate with each node of the data tree, a *signature*. The function of the signature is to map a data tree into a fixed length quantity. The signature of a node is defined recursively as follows

$$\sigma(n) = \begin{cases} \texttt{last\_} & , n \textit{ is a container.} \\ H(\sigma(c_1), \ldots, \sigma(c_N)) & , c_1, \ldots c_N \textit{ are} \\ & \textit{children of } n \end{cases} \quad (1)$$

For leaf-level containers, the signature is the last sequence number in its sequence. For internal nodes, the signature is computed as a hash function of the signatures of its children. A good example of a hash function that is suitable for this is MD-5 [21]. The likelihood of mapping two different trees to the same hash value is vanishingly small. This feature of MD-5 makes it well-suited for the purpose of representing the tree as a unique (with high probability) fixed length quantity.

A problem with long ASCII ADU names is that they incur a large per-packet overhead, especially for small packets like those in *MediaBoard* [20]. We propose a solution to this problem in the following sections.

3.2 Container Descriptors

Each container is mapped to a fixed length container descriptor and used as an identifier in packets. Explicit *bind* messages are used to disseminate the mapping information to receivers in the session. The container descriptor is assigned by the source that generated it.

Assigning container descriptors deterministically has the following inconsistency problem. Consider the problem of source crashes, where a source fails after generating some ADUs. Assume also the same source recovers from the failure and re-joins the session. New ADUs generated re-use the container descriptors that were already created during an earlier incarnation in the session. In order to overcome this inconsistency problem, each source picks an initial sequence number randomly. An identical problem in TCP is solved in a similar manner using a randomized Initial Sequence Number (ISN) [23].

In order to survive network partitions and avoid wrap around, we make the container descriptor a 32-bit integer. This can sustain network partitions of about 11 hours on the average for a session using 128 Kb/s on the average, and 200 B ADUs, typical of whiteboard applications.

4 Naming and Reliable Multicast

Floyd et al. [12] proposed the ALF model for light-weight sessions and scalable, reliable multicast. Although the application and the transport protocol were closely coupled in *wb*, the implementation lacked a generic framework for light-weight sessions. With generalized data naming it is possible to develop a customizable framework for applications that desire reliable multicast.

SRM uses two mechanisms to recover from losses – receiver-initiated repair requests, and source-initiated session announcements. Repair requests are used to recover from losses that can be detected by the receiver, from a discontinuity in the sequence numbers. If the last few bytes are lost in a transmission, receivers have no way of discovering such a loss on their own. Session announcements are periodic messages, containing the last sequence number transmitted so far, that a source multicasts to the session. Receivers use this information to recover from *tail* losses.

In order to use generic naming for light-weight receiver-driven reliable multicast, we need a different mechanism for discovering losses. We develop a new State Announcement Protocol (SAP) for recovering from tail losses in a container, losses of whole containers or branches of the ADU tree. The SRM repair request mechanism is used to recover from losses within a container.

5 State Announcement Protocol

SAP is designed to discover the location of losses that the receiver cannot deduce otherwise (i.e., lost containers and tail losses within a container). It uses the *signature* of the tree to determine the location of a loss. The source initiates the protocol by multicasting an `UPDATE` message periodically. [1] The `UPDATE` message carries the signature of a node in the ADU tree and a list of signatures and labels of its children.[2]

A receiver that hears an `UPDATE` message compares the value of the signature advertised in the packet against its own. If there is a mismatch, the receiver determines which children nodes conflict and multicasts a `QUERY` message for those nodes. The source, or any `ELIGIBLE` receiver can respond to this message with a new `UPDATE`. If the receiver mismatches on a leaf-level container, (i.e., `last_` $\neq$ signature), a repair request is scheduled for the interval `[last_`, signature]. The receiver also advances the right edge of the container to the value just received in the `UPDATE` message. This is necessary to avoid detecting the same loss more than once. The repair request machinery is persistent and eventually repairs the loss.

This is shown below in pseudo-code.

```
Agent/Message/SRM_SAP receive p
{
        ...
        $self parse p;
        ...
        set local [$self get-hash $name_]
        if {$local != $sig_} {
          if {$self isleaf $name_} {
           # Schedule SRM RREQ
           $self sched_rreq [$local, $sig_]
           set eligible_ 0
           return
          }
          # Schedule QUERY message for
          # offending child
          set child_ [$self cmp $name_]
          $self sched_query $name_/$child_]
        } else {
              set eligible_ 1
        }
}
```

5.1 Backoff and Suppression

The SAP protocol, like SRM repair requests, multicasts everything. In order to avoid multiple members from multicasting the same message (`UPDATE`, or `QUERY`), the protocol uses a technique similar to SRM. Each message is held back for a random amount of time depending on the RTT to the source. This gives a chance for members to suppress a

[1] To avoid synchronization between different sources in a session, we randomize the period. [11]

[2] If the maximum packet length cannot accomodate the entire list, the source may split this information across more than one packet.

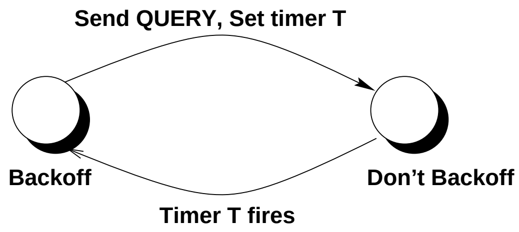


Figure 2: Modification to member's state machine to avoid false backoff

message if an identical message is heard. For very large-scale sessions, however, multicasting every control message to the entire group consumes precious bandwidth, and is unnecessary if the number of members experiencing a given loss is small and localized. In order to limit the traffic, local recovery schemes may be used to limit the scope of state announcement messages.

With backoff and suppression, each receiver sends a query only if it has not already heard an identical one. If a receiver does hear an identical query from another participant in the group, the timer for that query is exponentially backed off. The exponential backoff strategy works well if suppression works perfectly and there are never any duplicates. If more than one copy of a message are multicast, members backoff their timers more than is necessary. This condition of *false backoff* can be solved by extending the state machine of each member with the extra state `Don't Backoff`, shown in Figure 2. When a query has just been transmitted or suppressed suppressed by an identical message, the pending query moves to state `Don't Backoff`, where it does not react to similar query messages. The transition back to the `Backoff` state happens when a timer expires. The value of the timer is set to the perceived maximum round-trip time to any host in the session.

In response to a query, each receiver sends the current hash value of the requested node from the local tree. It is possible that some members match in a previous update, but have lose subsequent data and updates due to transient congestion. To prevent a member with an out-of-date tree from responding to queries from more up-to-date receivers, each message carries a timestamp. Each receiver maintains a timestamp per tree. The original source timestamps its update messages with a local timestamp. If a receiver matches at the root of the tree it advances its timestamp to the timestamp on the update message. The receiver includes the tree's timestamp with each update message. To avoid sending stale updates, a receiver refrains from responding to a query if the local timestamp is older than the timestamp on the query.

5.2 Heuristics for Announcements

The recursive descent method described above takes time proportional to the depth of the tree and the average RTT to converge. Heuristics can be applied in order to improve the convergence time. The recursive descent protocol described above does not use application-level information in selecting the contents of update messages. There are several application-level policies that can be invoked at this stage to determine what update information to send in each packet.

Working Set Approach

A heuristic used by *wb* is to send updates from the currently active pages. This heuristic can be generalized to tree-structured data. Only those banches of the tree that have had recent *activity* are updated. When a new receiver joins the session, the entire name space needs to be discovered. By selecting the right branches to query, the receiver can improve the latency for those data items. This heuristic is general enough for the transport protocol to carry on independently of the application. In addition to the existing mechanisms, this heuristic would require SAP to maintain a working set of containers, and age containers from this set as new active containers get touched. Each periodic update message sends updates for containers from the working set.

Skipping Long Chains

If the source encounters a chain of nodes at some stage during the descent, the chain can be skipped and the signature of the first higher degree node can be transmitted in the update message. Since all messages are stateless, and contain all the information required for lookup, this does not affect the receiver that sent the query. We evaluate the relative performance improvement with this scheme in the following section.

6 Performance Evaluation

We used simulations to study the behavior of SRM with SAP. In this section, we present the methodology and results of simulations.

6.1 Extensions to *ns*

We studied the behavior of SRM in combination with SSAP using simulations. The simulations were written in *ns* [18] written in C++ and Tcl/Tk. At the time we started, *ns* did not have a built-in reliable multicast module. We implemented SRM with scalable repairs and responses. *ns* has a split software architecture with objects living in C++ or Tcl/Tk [19] or both. We implemented an `SRM_Agent` module that runs the basic SRM timer mechanisms with suppression and backoff. The naming data structure (*k-ary trees*, with ASCII labels at each node) was implemented in

C++. [3]. The SAP protocol was implemented in OTcl [25], since OTcl is convenient for prototyping various policies.

The data for the session was from randomly generated traces. In all our experiments we used one source. Inter-arrival times for session data were picked from a uniform random distribution. We used random trees with sizes varying from 15 to 50 nodes. An example of the 15-nodes topology is shown in figure 4. Cross traffic was generated using TCP connections running ftp. On the average we maintained one congested link per 4-5 nodes. *ns* packet traces provided information on packet drops. A single run of 100 seconds with a 45-node toplogy took about 20 minutes of real time on a 200 MHz Pentium with 64 MB of RAM. NFS and disk I/O were the bottleneck. In order to stress test SAP, we restricted all containers to have a maximum sequence number of exactly 1. In this case, each packet drop translates to the loss of a container and can be recovered only with state announcement messages.

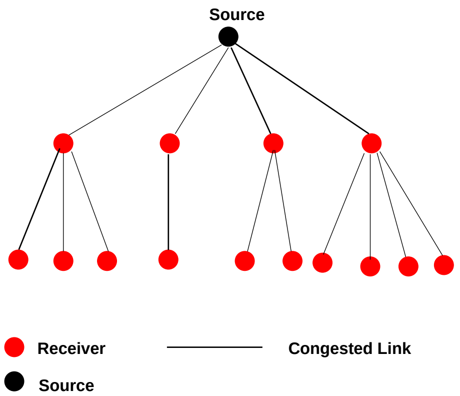


Figure 4: Example simulation topology

6.2 Metrics

We describe the main metrics we employed to evaluate the state announcement protocol and its variants.

Convergence

Convergence is related to the elapsed time from the instant at which a packet is dropped in the network (as a result of overflow at a router queue) to the instant that a receiver receives the packet. We measure the worst case convergence time in our simulations, i.e., the time from the drop to the instant the last receiver repairs the loss. Recall that the convergence time has three components, the average waiting time for the first update from the source, the time taken to discover the location of the loss using SAP, and the recovery time using repair requests and replies using SRM repair requests.

[3] The extensions to the simulator and the scripts used in this project are available at http://www.cs.berkeley.edu/s̃uchi/research/srm/ns.tar.gz

In all the experiments, the number of losses was kept constant. In an initial implementation of the protocol, a receiver would discard a packet if the timestamp on the packet was earlier than the current timestamp on the corresponding ADU tree. However, for small values of the SAP announcement period, this results in *thrashing*. The period is too short for the SAP protocol to descend to the leaf-level containers to schedule a repair request. For small values of the update period ($< 0.7s$), the session never recovers from container losses. We changed the protocol after observing this effect to reject old packets only if they were updates. For query packets, the SAP protocol simply sends its current view of the requested node.

Figure 6.2 shows the convergence behavior of the state update protocol with varing periodicity of updates. At small values of the period, SAP updates and queries consume a large amount of the session bandwidth, further increasing congestion on the bottleneck link. This combined with premature packet discard, results in large convergence times. The number of packets recovered in a given amount of simulation time is also small for high frequency state announcements, as seen in Figure 6.

There is a tradeoff between the amount of bandwidth expendable on control messages such as updates and queries, and the *sluggishness* of the protocol in discovering and reacting to losses. When the number of sources in the session is increased, the share of bandwidth available to a single source drops. The operating point on the curve in figure 6.2 depends on the fraction of control bandwidth that the source is entitled to. [4]

Scalability with session size

Our metrics for scalability are the number of copies of a message seen in a session, which is a measure of goodness of suppression. We also measured convergence times with increasing session size.

Effectiveness of Suppression

The chief concern with multicasting control messages such as repair requests, state updates and state queries is the amount of bandwidth consumed in very large session sizes. In order to evaluate the effectiveness of suppression in SAP,

[4] Later in the paper, we describe how application-level receiver interest can be used as a hint in allocating bandwidth between different sources. This requires that the receivers send back status reports to the session in a scalable way.

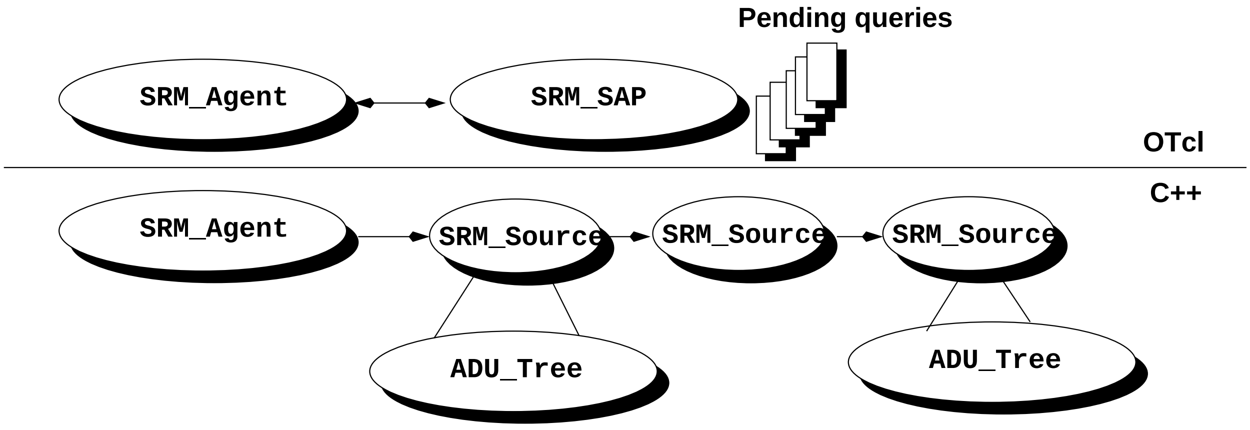


Figure 3: SRM and SAP agents in ns

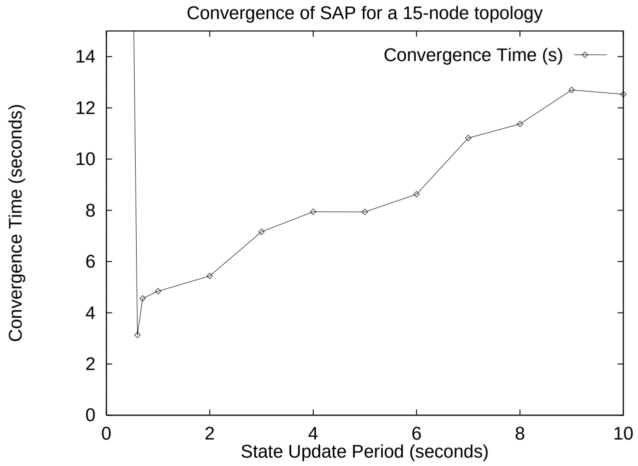


Figure 5: Convergence of SAP with decreasing frequency of updates

we looked at the number of copies of each control message multicast to the group. Figure 7 shows this behavior as the group size is scaled up to 50 nodes. For SAP updates, the average number of copies per message was about 3, and remained approximately constant with increasing session sizes. On the average, about 2 copies of a query message are transmitted to the session. This too, remains roughly constant with large group sizes.

From running several simulations for each topology, we observed that the success of suppression depends heavily on the choice of timers. This in turn relies on accurate RTT estimators. For

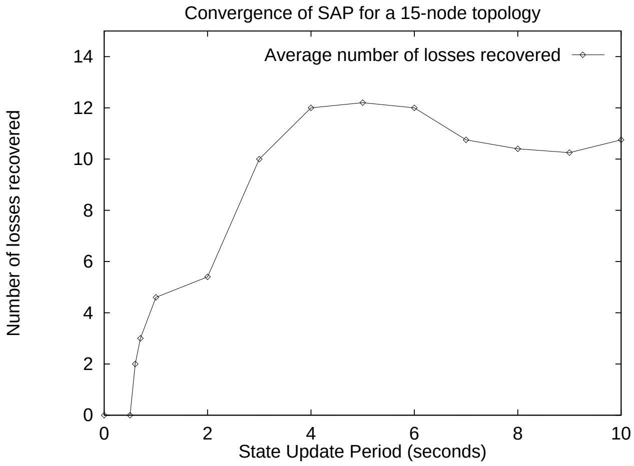


Figure 6: Number of packets recovered in 100 seconds of simulated time

very large sessions, on the order of millions, this involves maintaining heavy-weight state at the end-points. However, with local recovery, and about 50 nodes per local *region*, suppression ensures that a small constant number of messages will be transmitted to the local group.

Convergence Times

Figure 8 shows the convergence behavior of the protocol with increasing session size. In each of these simulations, the period of the session timer was varied between 5 and 8 seconds. From the graph we see that the convergence time does not change very much as we increase the session size. The convergence time is dependent on the SAP announcement period, the depth of the

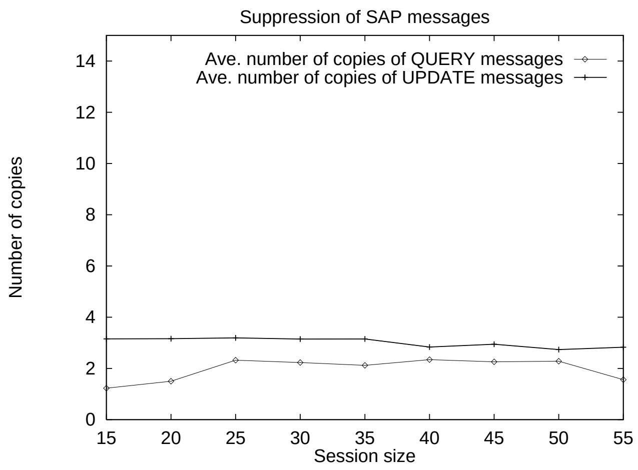


Figure 7: Suppression of State Announcement Messages

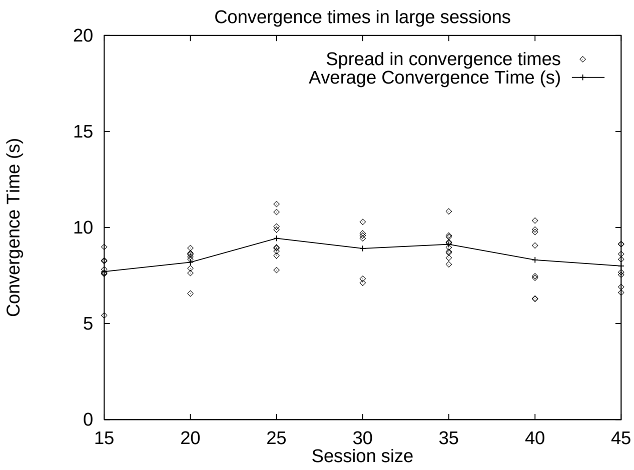


Figure 8: Convergence with increasing session size

ADU tree (in the absence of heuristics), and the backoff timers used during repair.

These experiments were performed for a single source. With multiple data sources, and in the absence of congestion-control or rate-control, the likelihood of congestion is higher. This could potentiallylead to greater losses in the network and cause the discovery-cum-repair process to take longer. The right solution in that case is to somehow signal this condition to the source and throttle its sending rate. At the time of this writing, there have not been conclusive studies on congestion control.

Heuristics

We implemented the short circuit heuristic described earlier in section 5.2. For data spaces with average width 10, and depth 13, we see that the convergence times are better with the heurictic. Recall that the overhead is composed of 3 components – time to receive next state announcement, time to discover loss, and the time to repair it. The short circuit algorithm reduces the second component of the delay. We see an improvement of about 20% in delay with this scheme.

7 Related Work

There has been a lot of recent and ongoing work in reliable multicast. Several groups have proposed to use a generic naming scheme. Specifically, a recent Internet draft [8] on RMFP proposes a two-layered data stream with OIDs and sequence offsets. However, each object simply represents a subsequence of the sequence space. The authors do not describe how objects are discovered by the receivers in the event of a loss. This scheme appears to be inflexible for general applications like file-system multicasting. It is also unclear how to map objects into OIDs.

8 Status and Future Work

SAP has been implemented and evaluated in the network simulator *ns* version 2 [18]. We plan to move the generalized naming scheme and the SAP protocol into the existing SRM implementation in the MASH shell [17]. We also plan to use it to develop real world applications such as MediaBoard [20], WebCast [14], and floor control applications. The real challenge to using ALF is in designing the right programming interface. Experience with building the vast suite of multimedia and file transfer-type applications that are within the scope of the MASH project can be invaluable in designing the right API. We see this scheme is a key enabler of the ALF or framework approach where code and protocol machinery can be re-used across many applications, while still retaining flexibility and customizability.

Other areas for exploration are generic and application-specific policies in selecting information transmitted in state update messages. The heuristic of *sending the most active k containers* is well-suited to shared whiteboards and file system mirroring applications.

9 Implementation Notes

We have charted out a scheme for implementing SAP in the SRM toolkit in *mash*. The packet formats for the various additional packets are shown in the figures below. By mapping each child node at a level onto a sequence number,

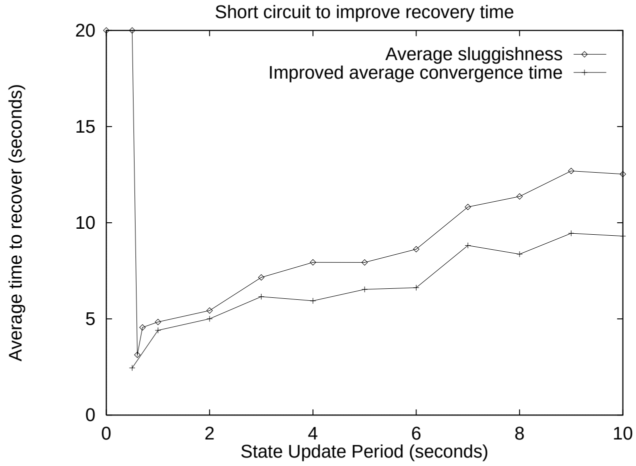


Figure 9: Performance improvement with the *short circuit* heuristic

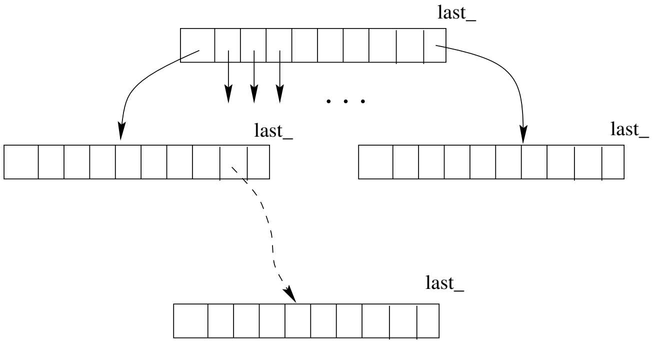


Figure 10: Trie implementation of hierarchical data

we make the mechanism for SAP queries the same as the mechanism for repair requests. The data structure now resembles a trie, shown in Figure 10. This modification leads to a common structure for both request and query messages. We also plan to implement variable length sequence numbers for long-lived sessions with huge amounts of data. Other factors that affect the implementation of generalized naming are interactions with SRM archiving agents. We plan to explore this topic with recent experience gained in developing a prototype recorder for SRM streams [4].

10 Summary and Conclusions

In this paper, we motivated the need for structured data name spaces, and proposed a heirarchical scheme for data naming. Hierarchical data naming that mirrors application data names can introduce a high overhead if these names are transmitted with every transmission unit. In order to

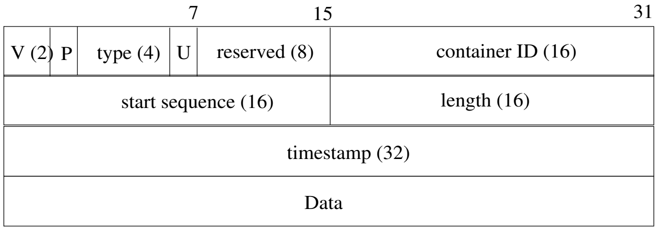


Figure 11: SRM_DATA Packets

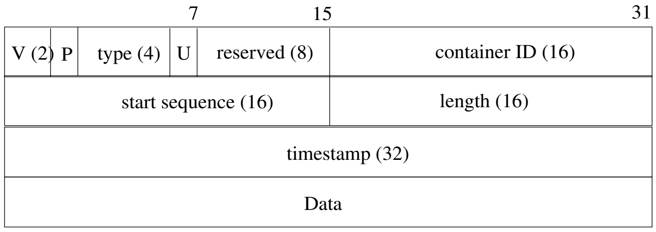


Figure 12: QUERY/REQUEST Packets

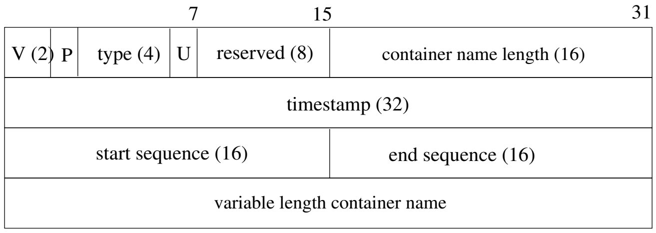


Figure 13: UPDATE Packets

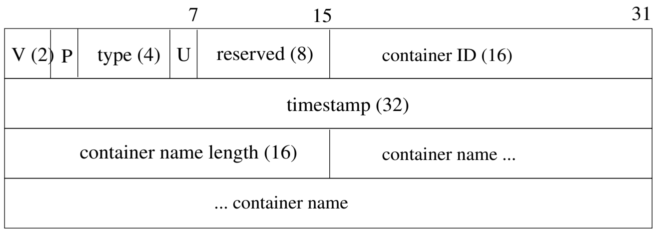


Figure 14: Bind Packets

overcome this, we propose a mapping scheme from name to fixed length integer. We applied the general naming scheme to reliable multicast and extend SRM with a state annoucement protocol to discover and recover from losses in hierarchical name spaces. We used simulations to study the performance impact of such a scheme and find that the protocol scales well to large group sizes. With multiple sources, each source consumes its fraction of the control bandwidth for state announcements. Alternatives to equal sharing involve voting based on receiver interest [1]. Hierarchical naming provides a framework for hierarchical rate-control among the different containers and internal nodes of the hierarchy.

Work on scaling control traffic to extremely large sessions by limiting its scope has received considerable attention in the recent past. Any solution proposed for local recovery can be used profitably in the SAP protocol. Reporting only a summary of the messages heard in a local area to "higher-ups" in the multicast distribution tree can reduce the amount of redundant traffic generated by global multicast.

One possible implementation of containers is by using multiple multicast groups (MMGs) by mapping a few containers to a single multicast channel. MMGs facilitate receiver-driven reception of selected containers. SAP can be used to discover the mapping and also detect lost containers. MMGs are merely a different way of implementing hierarchical data, and do not solve the problem of discovering the structure of the hierarchy.

11 Acknowledgements

We would like to thank Hari Balakrishnan and Teck-Lee Tung for suggestions that greatly improved the quality of this work.

References

[1] AMIR, E. Private communication, Jan. 1997.

[2] BALAKRISHNAN, H., SESHAN, S., AMIR, E., AND .KATZ, R. H. Improving the performance of TCP over wireless links. In *Proceedings of MobiCom* (Dec. 1995).

[3] BRADEN, R. *T/TCP – TCP Extensions for Transactions Functional Specification*. ARPANET Working Group Requests for Comment, DDN Network Information Center, ISI, 1994. RFC-1664.

[4] CHAWATHE, Y., AND WANG, H. J. A recorder for mediaboard streams. UCB CS 286 Project Report, May 1997.

[5] CKER CHIUEH, T. Distributed systems support for networked games. In *Proceedings of SPIE First International Symposium on Technologies and Systems for Voice, Video, and Data Communications* (Oct. 1995).

[6] CLARK, D. Structuring operating systems using upcalls. In *Proceedings of the Eleventh Symposium on Operating Systems Principles* (Dec. 1991).

[7] CLARK, D. D., AND TENNENHOUSE, D. L. Architectural considerations for a new generation of protocols. In *Proceedings of SIGCOMM '90* (Philadelphia, PA, Sept. 1990), ACM.

[8] CROWCROFT, J., WANG, Z., GHOSH, A., AND DIOT, C. Rmfp: A reliable multicast framing protocol, Mar. 1997. Internet Draft (RFC pending).

[9] DEERING, S. E. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Dec. 1991.

[10] ERIKSSON, H. Mbone: The multicast backbone. *Communications of the ACM 37*, 8 (1994), 54–60.

[11] FLOYD, S., AND JACOBSON, V. The synchronization of periodic routing messages. In *Proceedings of SIGCOMM '93* (San Francisco, CA, Sept. 1993), ACM, pp. 33–44.

[12] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A reliable multicast framework for lightweight sessions and application level framing. In *Proceedings of SIGCOMM '95* (Boston, MA, Sept. 1995), ACM, pp. 342–356.

[13] INC., P. *PointCast Home Page*. Software on-line[5].

[14] LIAO, T., ET AL. *WebCanal - Broadcast of Web Documents*. INRIA, France. Software on-line[6].

[15] LIN, J. C., AND PAUL, S. RMTP: A reliable multicast transport protocol. In *Proceedings IEEE Infocom '96* (San Francisco, CA, Mar. 1996), pp. 1414–1424.

[16] MCCANNE, S. A distributed whiteboard for network conferencing, May 1992. U.C. Berkeley CS268 Computer Networks term project and paper.

[17] MCCANNE, S., ET AL. Towards a common infrastructure for multimedia-networking middleware. In *Proceedings of the Seventh International Workshop on Network and OS Support for Digital Audio and Video* (St. Louis, CA, May 1997), ACM.

[18] MCCANNE, S., AND FLOYD, S. *The LBNL Network Simulator*. Lawrence Berkeley Laboratory. Software on-line[7].

[19] OUSTERHOUT, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[20] RAMAN, S., AND TUNG, T.-L. A distributed mediaboard using the scalable, reliable multicast toolkit. UCB CS 262 Project Report, Dec. 1996.

[21] RIVEST, R. *The MD5 Message-Digest Algorithm*. ARPANET Working Group Requests for Comment, DDN Network Information Center, MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. RFC-1321.

[22] SMITH, W. G., AND KOIFMAN, A. A distributed interactive simulation intranet using ramp, a reliable adaptive multicast protocol. In *Proceedings of the Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, FL* (Mar. 1996).

[5] http://www.pointcast.com

[6] http://monet.inria.fr

[7] http://www-nrg.ee.lbl.gov/ns/

[23] STEVENS, W. R. *TCP/IP Illustrated, Volume 1 – The Protocols*, first ed. Addison-Wesley, Dec. 1994.

[24] VAN RENESSE, R., BIRMAN, K. P., AND MAFFEIS, S. Horus, a flexible Group Communication System. *Communications of the ACM*, 4 (1996).

[25] WETHERALL, D., AND LINDBLAD, C. J. Extending tcl for dynamic object-oriented programming. In *Proceedings of the Tcl/Tk Workshop '95* (Toronto, July 1995).