Hierarchical Message Stability Tracking Protocols *

Katherine Guo, Robbert van Renesse, Werner Vogels and Ken Birman

Department of Computer Science

Cornell University

Ithaca NY, 14853

{kguo,rvr,vogels,ken}@cs.cornell.edu

Abstract

Protocols which track message stability are an important part of reliable multicast protocols in fault-tolerant distributed systems. To reliably deliver multicast messages in a process group, each process maintains copies of all messages it sends and receives. If a member fails to receive a message, any process which has the message in its buffer can retransmit it. In order to prevent these buffers from growing out of bound, stability tracking protocols must be used. That is, whenever a process learns that a message has been received by everyone, it declares this message *stable* and releases it from the buffer.

We investigate several message stability tracking protocols commonly used in a number of popular reliable multicast protocols with a focus on their performance in large scale settings with thousands of participants. To improve the scalability of these protocols significantly, we derive a set of new protocols using a spanning tree structure which scale to at least tens of thousands of participants.

Keywords: message stability, protocol simulation, reliable multicast

*This work was supported by the ARPA/ONR grant N00014-96-1-1014 and the GTE graduate student grant to Horus research group.

1 Introduction

Group communication has been used in many applications in fault-tolerant distributed systems, such as brokerage and trading systems, replicated database systems and replicated file systems. Informally, reliable multicast in a group requires that all correct processes deliver the same set of messages, and that this set include all messages multicast by correct processes, and no spurious messages [15].

Protocols which track message stability are an important part of reliable multicast protocols. To deliver multicast messages reliably in a process group, each process maintains in a buffer copies of all messages it sends and receives. In the case where a member fails to receive a message, any process which has a copy of the message can retransmit it. Stability tracking protocols must be used to prevent these buffers from growing out of bound. That is, whenever a process learns that a message in its buffer has been received by everyone, it declares this message *stable* and releases it from the buffer.

The concept of message stability has been used in more traditional areas such as distributed database management and parallel computing. In distributed database systems, partial failures of transactions can lead to inconsistent results. Therefore, termination of a transaction that updates distributed data has to be coordinated among its participants. In the atomic commit protocols [3], a process can not commit a transaction until everybody else has agreed to commit. This is similar to message stability protocols in which all processes must deliver a message if any does so.

In parallel computing, barrier synchronization [17] requires that all processes execute the barrier construct before any process can proceed past it to the next statement. Every process has to know if all other processes have reached the barrier before it can proceed again. This is also an agreement problem just like message stability and atomic commit.

With the expansion of the Internet and the advent of ubiquitous, world-wide distributed systems, reliable multicast protocols will be used in large-scale settings involving thousands of participants. For example, distribution of consistent routing information to thousands of routers of the multicast backbone (MBone) of the Internet will use reliable multicast; management of thousands of computers in a corporate intranet has already made use of reliable multicast. It is time to examine the commonly used basic stability protocols with a focus on their performance in large-scale settings involving thousands of participants. To overcome the scaling limitations associated with the existing protocols, we derive a set of new protocols using hierarchical structures with the ability to handle tens of thousands of participants.

In this paper, we start in Section 2 with the basic assumptions and performance indices. In Section 3 we describe the basic message stability protocols commonly used in existing distributed systems. We then derive a set of structured stability protocols by introducing a spanning tree in Section 4. We use extensive simulation to compare all the protocols in Section 5. Section 6 discusses the triggering mechanism for stability detection protocols and Section 7 concludes the paper.

2 The basic assumptions and performance indices

The protocol that collects message stability and distributes this information to every group member is called a *message stability tracking protocol*, or *stability protocol* for short. Such protocols are implemented as an integral part of reliable multicast protocols in many distributed systems [2, 4, 5, 7, 10, 19]. We study three representative protocols dubbed **CoordP**, **FullDist** and **Train**.

2.1 The basic assumptions

To compare these stability protocols, we assume that the underlying communication layer offers FIFO delivery and that group membership remains constant. We also make the following assumptions, because our focus is on the performance of stability tracking, not of reliable multicast or group membership protocols[1].

- The process group size is n and members of the group are numbered 1 through n.
- Any member can be a sender and every member has to be a receiver. Each message is assigned a sequence number by its sender, and therefore each message is uniquely identified by the pair (sender ID, sequence number).
- Each sequence number occupies 4 bytes. (The sequence number space is between 0 and 2^{32}, which is big enough for most applications).
- A multicast message is always sent to the entire group, and therefore a sender also receives a multicast message from itself.

[1]In this paper we assume a static membership. It is beyond the scope of this paper to show that these message stability protocols can be made fault-tolerant in the case of message loss or process failure.

- A routing architecture similar to IP multicast [8] is used.

2.2 Performance indices

We define five indices that can be used to characterize the performance of stability protocols. To measure message complexity, we use *total number of messages* on all hops in the system. To measure time complexity, we use *round-trip time* and *message rounds*. Round-trip time is defined as the time it takes for one execution of the protocol to complete. Message rounds are used to describe the protocols. Since not every round takes the same amount of time, number of message rounds is not a good time complexity index. To measure the distribution of processing load among members, we use *number of messages processed* and *maximum queue size* over all processes. Number of messages processed is the sum of the number of messages sent and received by each member. Maximum queue size measures how large a buffer needs to be at a process to handle all the incoming messages. The total number of messages, round-trip time and maximum queue size can be affected by the underlying network environment, whereas the other two indices can be determined without information about the network. Therefore, we analyze number of message rounds and number of messages processed first and then compare the other three indices in the simulation section.

3 The basic protocols

3.1 CoordP

CoordP is a centralized protocol run by one of the members of the group, designated as the *coordinator*. Each member i maintains an n-element array R_i where its j-th element $R_i[j]$ is the sequence number of the last FIFO message received by member i from member j. Three types of messages are used in this protocol: the START message is of size 1 byte[2], the ACK and INFO messages are of size $4n$ bytes. There are three rounds as shown in Figure 1:

- **Round 1:** The coordinator multicasts a START message in the group;

- **Round 2:** After receiving the START message, each member i sends its array R_i to the coordinator as an ACK message by point-to-point links;

[2]In implementation of protocols, one field in message header normally designates the message type. The message body for a START message is empty. In the analysis of protocols, message headers are not considered. Hence the START message size is set to 1 byte. In our simulation, a START message only has a header.

- **Round 3:** After collecting sequence number arrays from all the members, the coordinator calculates $R = ArrayMin_{\{i\}}(R_i)$, where $R[j] = min(R_1[j], R_2[j], \ldots, R_n[j])$. The array R is then multicast in the group as an INFO message. Based on the received R, any member in the group can label a message received from member s as stable if the message sequence number $P_s \leq R[s]$.

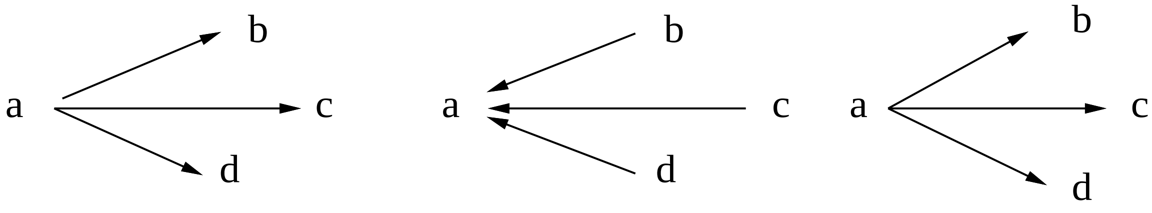


Figure 1: Rounds of protocol **CoordP**

In this protocol, there are 2 multicasts by the coordinator and $n-1$ point-to-point messages from the non-coordinators. The coordinator sends 1 START and 1 INFO multicast, it receives 1 START, 1 INFO, and $n-1$ ACKs. Therefore, the total number of messages processed by the coordinator is $n+3$, of which 2 messages are sent and $n+1$ are received. The multicast is counted as a message sent and received by the coordinator. A non-coordinator sends 1 point-to-point ACK message, receives 1 START and 1 INFO messages. The total number of messages processed by a non-coordinator is 3, of which 1 is sent and 2 are received.

In the Tandem global update protocol [5] and the Amoeba total ordering protocol [10] [12], a particular version of **CoordP** is employed as their stability tracking algorithm. In both systems, there is a sequencer (or a coordinator) which assigns the global sequence number to each data message. Other members only need to send a 4-byte field — the last consecutive sequence number to the sequencer. After receiving these numbers, the sequencer calculates their minimum and announces the sequence number of the last stable message in the group.

3.2 FullDist

FullDist is a fully distributed protocol in the sense that every member periodically multicasts its information about message stability in the entire group. In **FullDist**, each member keeps a stability matrix M of size $n \times n$, where $M[i,j]$ stores the sequence number of the last message that is sent by member j and has been received by member i. The i-th row of the matrix at member i stores the last sequence numbers of messages that have been received by member i from all the members of the group. The minimum of the j-th column, represents the last sequence number whose corresponding message is sent by the j-th member and has been received by every member. Messages sent from member j with this sequence number or lower are stable by definition. This protocol only uses one type of $4n$-byte INFO messages. Periodically, each member multicasts its row of its stability matrix M in the group. For the purpose of measuring round-trip time for the simulation in Section 5, we introduce two rounds in **FullDist** as illustrated in Figure 2:

- **Round 1:** The first member multicasts the first row of its matrix M via an INFO message. No significance is attached to the choice of the first member.
- **Round 2:** After receiving the INFO message, the i-th member multicasts the i-th row of its stability matrix M via an INFO message. Every member replaces the i-th row of its matrix with the received row information from member i.

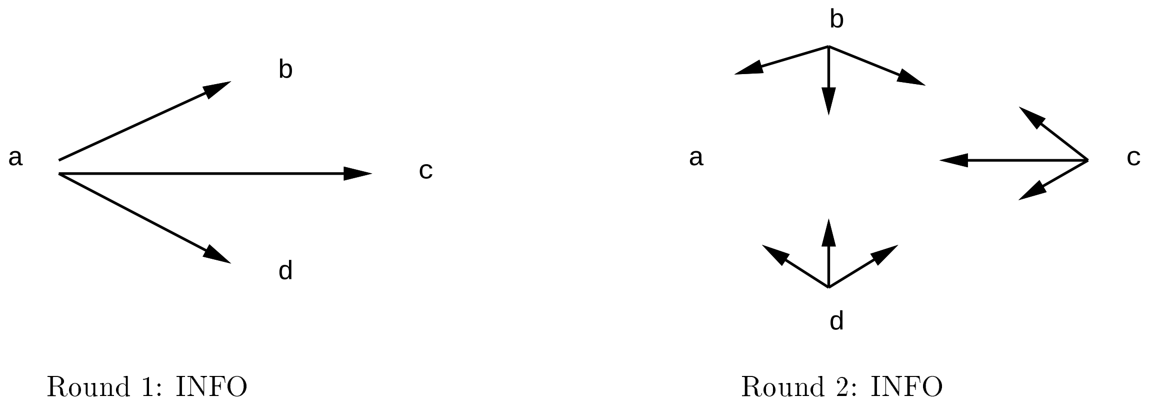


Figure 2: Rounds of protocol **FullDist**

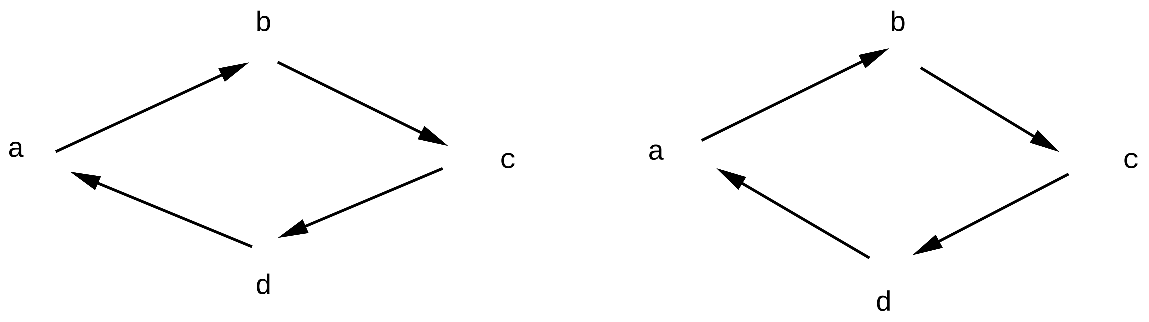
As every member maintains its own stability matrix and determines whether or not any data messages in the system are stable, **FullDist** is decentralized without any coordinator. This protocol includes n multicast INFO messages. Every member sends 1 INFO multicast and receives n INFO messages. Hence, the total number of messages processed by each member is $n + 1$.

The Horus/Ensemble [9, 19] system implements a set of stability protocols in separate layers so users can pick the appropriate one for their application. **FullDist** is one of the stability protocols offered by Horus/Ensemble.

3.3 Train

Train is a decentralized linear protocol in the sense that a fixed size "train" is passed around group members to spread the message stability information. In the **Train** protocol, each member i keeps a sequence number array R_i of size n. There is a cyclic order among group members 1, 2, $\ldots$, n. A "train" with a fixed size of $4n$ bytes passes through all the members in this cyclic order. Member 1 starts the protocol by putting its stability array on the train. When the "train" arrives at any other member, this member gets the array S from the "train", calculates the minimum of S and its own stability array using ArrayMin, then puts the result back in the "train". After one circulation, the first member gets back in the "train" the minimum of stability arrays of all the members, since the "train" has visited every member once. The "train" containing this minimum array then passes around the group in the circle again. In this second round, every member takes this minimum array and marks stable messages accordingly.

The two types of messages used by the **Train** protocol are the $4n$-byte ACK and INFO messages. The first member starts the protocol by assigning $S_1 = R_1$, then sending a point-to-point ACK message containing S_1 to the second member. Upon receiving this ACK, the second member calculates $S_2 = ArrayMin(S_1, R_2)$, and sends S_2 to the third member via an ACK. In general, after receiving an ACK message containing S_{i-1}, member i calculates $S_i = ArrayMin(S_{i-1}, R_i)$, and sends S_i on an ACK message to member $i+1$. After an ACK message from member n arrives at the first member, that is, after the ACK finishes circulating one round in the group, the first member starts passing S_n in the same circle in an INFO message. At this point, $S_n = ArrayMin_{\{i\}} R_i$, that is, S_n contains the sequence numbers for the last stable messages sent from each member. The **Train** protocol is shown in Figure 3.



Round 1 to n: ACK   Round $n+1$ to $2n$: INFO

Figure 3: Rounds of protocol **Train**

In this protocol, the ACK message needs to circulate the group once, and so does the INFO message. Hence, $2n$ rounds are required. Each member sends out and also receives 1 ACK and 1 INFO message. The total number of messages processed by each member is 4, out of which 2 are sent and 2 are received.

In the Train [6], Pinwheel [7] and Totem [14] protocols, the **Train** protocol is used to offer message stability tracking.

4 The structured protocols

Later in this section, we will show that the three basic protocols have their limitation in scalability. The most obvious way to improve scalability is to use hierarchy. Tree structures have been used in reliable multicast protocols in distributed systems [9, 14] and barrier synchronization algorithms [13] in parallel systems. To improve scalability significantly, we derive two structured stability tracking protocols by adding a spanning tree structure to the basic protocols. They are dubbed **S_CoordP** and **S_Train** since they are derived from **CoordP** and **Train** respectively. As we will see later, **S_FullDist** would be equivalent to **S_CoordP**.

To describe the structured protocols, we use the same assumptions made in Section 2. We also assume the group with n members is organized into a complete tree with height p and fixed degree b at all levels. The root of the tree is the first member. The size of the group can be expressed by tree parameters b and p as $n = 1 + b + b^2 + \cdots + b^p = \frac{b^{p+1}-1}{b-1}$. The sibling members under the same parent constitute a subgroup of size b. In the following description, a node is identified by a pair (d, e), where d is its depth and e is its location on level d of the tree. On level d, the left most node is labeled as 1 and the right most node is labeled as b^d. Hence d ranges from 0 to p and e ranges from 1 to b^d. Each node (d, e) also has a global number $q = 1 + b + b^2 + \cdots + b^{d-1} + e$.

4.1 S_CoordP

We derive **S_CoordP** from **CoordP** by employing a hierarchical structure. We assign the root to be the coordinator. Each member (d, e) maintains an array $R_{(d,e)}$ of size n, where its j-th element $R_{(d,e)}[j]$ stores the sequence number of the last message received from member j. Three types of messages are used: a 1-byte START message, and $4n$-byte ACK and INFO messages. The protocol has $p + 2$ rounds as illustrated in Figure 4:

- **Round 1:** The root starts the protocol by multicasting a START message.

- **Round 2:** After receiving START, each leaf member sets $S_{(p,e)} = R_{(p,e)}$ and sends a point-to-point ACK message containing $S_{(p,e)}$ to its parent with node number $(p-1, l(e))$ where $l(e) = ceiling(e/b)$.

- **Round 3 to p+1:** (The tree height is p, hence $p-1$ rounds are needed for the ACKs from the next to bottom level to reach the root). Upon receiving $S_{(d+1,e)}$ from all its children, an internal member (d, c) (that is, a member that is neither the root nor a leaf) sets $S_{(d,c)} = ArrayMin(R_{(d,c)}, S_{(d+1,e)})$ over $e = (c-1)b + 1$, $(c-1)b + 2$, $\ldots$, cb. It then sends $S_{(d,c)}$ to its parent.

- **Round p+2:** With all the $S_{(1,e)}$'s collected from its children, the root sets $S_{(0,1)} = ArrayMin(R_{(0,1)}, S_{(1,e)})$ over $e = 1, 2, \ldots, b$. Then it multicasts an INFO message containing $S_{(0,1)}$. After receiving $S_{(0,1)}$, a member can label any message from member s stable if it has a sequence number $P_s \leq S_{(0,1)}[s]$.

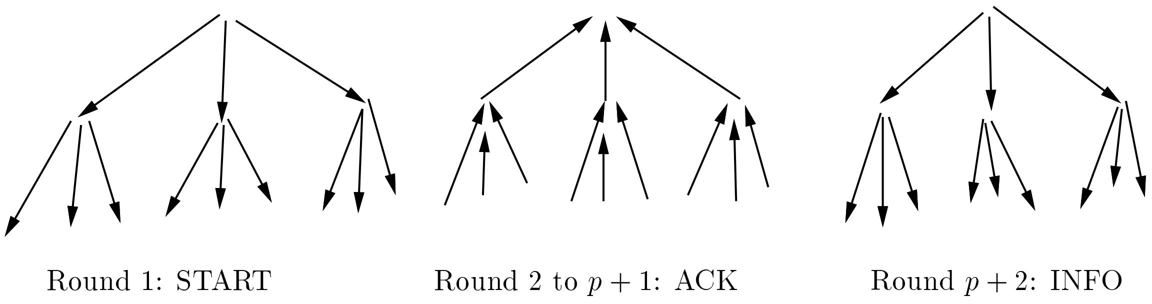


Figure 4: Rounds of protocol **S_CoordP**

In this protocol, there is 1 multicast of START, 1 multicast of INFO from the coordinator located at the root and $n-1$ point-to-point ACK messages from other members. The root sends 1 START and 1 INFO multicast, while it receives 1 START, 1 INFO and b ACKs. Hence, the number of messages processed by the root is $b+4$, out of which 2 are sent and $b+2$ are received. An internal member sends 1 point-to-point ACK message, and receives 1 START, 1 INFO and b ACK messages. Therefore the number of messages processed is $b+3$, out of which 1 is sent and $b+2$ are received. A leaf member sends 1 point-to-point ACK message, receives 1 START and 1 INFO message. The number of messages processed is 3, out of which 1 is outgoing and 2 are incoming.

By adding a tree structure to **FullDist**, we end up with a protocol where each member multicasts its array of sequence numbers in its subgroup and each member collects information from its children before it multicasts the combined information in its own subgroup. But this multicast is redundant — a member only needs to report its sequence number array to its parent in order for the information to propagate to the top of the tree. By changing the redundant multicast into a point-to-point message to its parent, we end up with protocol **S_CoordP**.

4.2 S_Train

Adding a tree structure to **Train** results in **S_Train**. We assign the root to be the coordinator. There is a cyclic order 1, 2, ..., b among members of each subgroup. Each member (d, e) maintains a sequence number array $R_{(d,e)}$ of size n, where its j-th element $R_{(d,e)}[j]$ stores the sequence number of the last message received from member j. The size of every ACK1, ACK2 and INFO message is $4n$ bytes. As in Figure 5, $pb + 2$ rounds are needed.

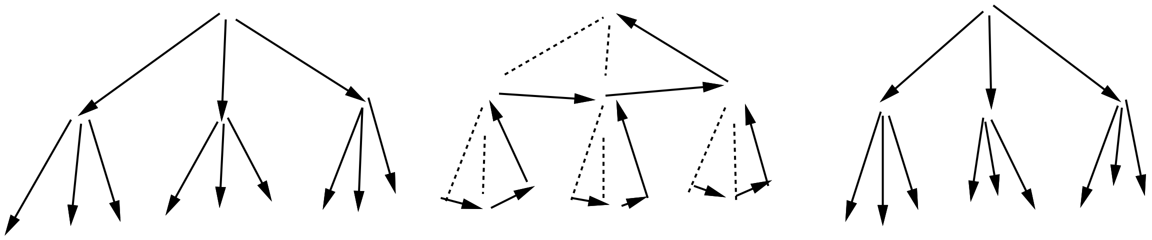
- **Round 1:** The root starts the protocol by multicasting a 1−byte START message.
- **Round 2:** After receiving START, the first member (p, e) of each leaf group sets $S_{(p,e)} = R_{(p,e)}$ where $e = 1, b+1, 2b+1, \ldots, (p-1)b+1$, and sends a point-to-point ACK1 message containing $S_{(p,e)}$ to the second member of the same leaf subgroup.
- **Round 3 to b+2:** Upon receiving ACK1 from its sibling $(p, e+i-1)$, member $(p, e+i)$ with $(0 < i < b)$ sets $S_{(p,e+i)}$ $=Min(R_{(p,e+i)}, S_{(p,e+i-1)})$, then member $(p, e+i)$ sends $S_{(p,e+i)}$ on an ACK1 message to member $(p, e+i+1)$. Whereas member $(p, e+b-1)$ sends $S_{(p,e+b-1)}$ on an ACK2 message to its parent member $(p-1, l(e+b-1))$.
- **Round b+3 to bp+1:** (Each level requires b rounds to pass all its information to its parent node, hence a total of bp rounds is required for all the ACKs to reach the root). After collecting the START and ACK2 from its b-th child, the first member (d, e) in an internal subgroup, where $e = 1, b+1, 2b+1, \ldots, (d-1)b+1$, sets $S_{(d,e)}$ $=ArrayMin(R_{(d,e)}, S_{(d+1,eb)})$, and then sends $S_{(d,e)}$ to the second member $(d, e+1)$ of the same subgroup via an ACK1 message. After receiving ACK2 from its b-th child, and ACK1 containing $S_{(d,e+i-1)}$ from its sibling $(d, e+i-1)$, a member $(d, e+i)$ with $(0 < i < b)$ sets

  $S_{(d,e+i)}$ $=ArrayMin(R_{(d,e+i)}, S_{(d,e+i-1)}, S_{(d+1,(e+i)b)})$, and then member $(d, e+i)$ sends $S_{(d,e+i)}$ via an ACK1 message to its sibling $(d, e+i+1)$, whereas member $(d, e+b-1)$ sends $S_{(d,e+b-1)}$ via an ACK2 message to its parent $(d-1, l(e+b-1))$.
- **Round bp+2:** After the root receives an ACK2 message from its b-th child $(1, b)$, it sets $S_{(0,1)} = Min(R_{(0,1)}, S_{(1,b)})$, then multicasts an INFO message containing $S_{(0,1)}$ in the entire

	CoordP	**FullDist**	**Train**	**S_CoordP**	**S_Train**
# rounds	3	2	$2n$	$p+2$	$bp+2$
# msgs processed	$n+3$ (coord) 3 (other)	$n+1$	4	$b+4$ (root) $b+3$ (internal) 3 (leaf)	5 (root) 4 or 5 (internal) 3 or 4 (leaf)

Table 1: Complexity of protocols (exact formula)

group. After receiving $S_{(0,1)}$, a member can label any message from member s stable if it has a sequence number $P_s \leq S_{(0,1)}[s]$.



Round 1: START

Round 2 to $bp+1$: ACK1 and ACK2

Round $bp+2$: INFO

Figure 5: Rounds of protocol **S_Train**

4.3 Comparison of the five protocols

Indices for time complexity (number of message rounds) and processor load (number of messages processed) for the five protocols are summarized in Tables 1 and 2. The processing load is reduced to $O(1)$ at every node in both structured protocols. This indicates that message implosion is completely eliminated, therefore we expect a better scalability from the structured protocols.

S_CoordP increases the number of rounds to $O(\log n)$ from $O(1)$ of the basic protocols **CoordP** and **FullDist**. One might argue that it takes longer for the structured protocol **S_CoordP** to complete than its corresponding basic protocols. However, if the tree in the structured protocols is built according to the physical layout of the nodes on the network, a message from a leaf to the root needs to go through p hops in any case, and the latency is not increased, but rather decreased because of elimination of implosion.

S_Train reduces the number of rounds to $O(\log n)$ from $O(n)$ of the basic protocol **Train**, while keeping the processing load unchanged at each member. In **S_Train**, the tree structure

	CoordP	**FullDist**	**Train**	**S_CoordP**	**S_Train**
# rounds	$O(1)$	$O(1)$	$O(n)$	$O(\log n)$	$O(\log n)$
# msgs processed	$O(n)$ (coord) $O(1)$ (other)	$O(n)$	$O(1)$	$O(1)$	$O(1)$

Table 2: Complexity of protocols

is used to help passing the arrays of sequence numbers up to the root concurrently, therefore effectively reducing the latency of the protocol.

5 Analysis by simulation

In Sections 3 and 4, we have calculated the number of rounds and the number of messages processed by each member for the basic and structured protocols. Comparisons are made for these protocols without considering the importance of the network environment. The total number of messages on all hops in the system and round-trip time can only be measured in a given network environment. The main purpose of our simulation is to measure these indices in a number of network topologies.

5.1 The underlying network

One of the settings in which stability protocols are used is a clustered environment where thousands of machines are connected via LANs in a small geographic span, like on a campus. We have conducted our simulation in this environment. We simulate the stability protocols in a single group of multicast servers running on distinct processors in a point-to-point network.

We use the ns [12] simulator to explore the behavior of these protocols. In our simulation, each link is bi-directional and each direction has a bandwidth of 100 Mbps (Fast Ethernet, FDDI and ATM can achieve this bandwidth). The time a message spends on the wire to travel one link is $t_0 = u/v$, where u is the message size in bytes and v is the bandwidth. Message propagation delay on a LAN is in the order of microseconds, therefore negligible. The time a message spends in a router is 1 millisecond[3]. The time needed for a host to send message in microseconds follows the formula $t_s(u) = 100 + 2(94 + 35u/4000 + 50u/1000) + 50 = 338 + 47u/400$ [1, 11]. The time needed for a host to receive a message is normally about 10% higher than the sending time since

[3]Typical value for router processing time is 1 to 10 milliseconds with today's technology.

z	1	2	3	4
$b=2$	47	63		
$b=3$	202	283	364	
$b=4$	597	853	1109	1365

Table 3: Group Size (height $p = 5$)

interrupts need to be handled to receive a message [1]. Therefore, we set $t_r(u) = 1.1 \times t_s(u)$. We set the header size of each message to $h = 32$ bytes which is enough for most transport protocols [1, 18].

A real network can have a chain, star, or tree topology. The tree structure is a generalization of both a chain and a star. In general, the structure of clustered machines in a LAN setting is a tree, so is the structure of a group with a large number of members spanning large geographic areas on the Internet. Therefore, as the underlying network used in our simulation, we pick a b-ary tree with height $p = 5$ that is complete up to level $p-1$. The degree z of the nodes at level $p-1$ varies from 1 to b. In a clustered environment, 5 is a reasonable height for the tree. A child node is connected to its parent via a bi-directional link. There is one group member on each node of the tree. A leaf node serves only as a host, whereas a root or an internal node serves both as a host and as a router. For a large group size n, the probability that a particular node in a random labeled tree has a degree of at most four approaches 0.98 [16], therefore we construct trees with fixed degrees of 2 to 4. Table 3 lists the sample tree structure and the corresponding group size n used in the simulation which can be calculated by

$$n(b,p,z) = 1 + b + b^2 + \cdots + b^{p-1} + zb^{p-1} = \frac{b^p-1}{b-1} + zb^{p-1}.$$

5.2 Complexity indices

To measure message complexity, we use the total number of messages on all hops in the network. A point-to-point message is counted w times if it travels w hops to get to its destination. A multicast message is counted as the total number of hops the original message and its copies travel in the network to reach all the destinations.

One index to measure processing load is *maximum queue size* over all the processes which indicates how large the buffer needs to be to hold messages used in a certain protocol.

A more accurate index to measure time complexity is *round-trip time* which is defined as the time for one execution of the protocol.

Our simulation is designed to satisfy all the assumptions in Section 2. IP multicast is provided by the ns [12] simulator. We simulate the sending of a data message to $n-1$ group members as sending point-to-point messages along the multicast spanning tree which coincides with the underlying tree network. Alternatively, this could be done by sending $n-1$ point-to-point messages. We choose the former method because IP multicast is becoming commonplace.

5.3 Analytical results

5.3.1 Total number of messages on all hops in the system

Given the tree structure of the underlying network, analytical results can be derived for the total number of messages on all hops. In the following section, when a tree is used in the analysis of the three basic protocols, it is the underlying tree network; when a tree is used in the two structured protocols, it is both the underlying network and the tree structure employed by the protocols. In real applications, these two trees do not necessarily coincide with each other. We make this assumption here to simplify the analysis and the simulation. In the future we plan to conduct more simulation without this assumption.

In **CoordP**, every node other than the root sends an ACK message to the root. An ACK message originated from depth k needs to travel k hops to reach the root, hence it is counted as k messages. There are b^k nodes with depth k, so there are kb^k messages originated from level k. There are zb^{p-1} nodes with depth p, so there are pzb^{p-1} messages originating from the bottom level. The total number of ACKs in **CoordP** is therefore

$$F_a = \sum_{k=0}^{p-1}(kb^k) + pzb^{p-1} = \frac{b-b^p}{(1-b)^2} - \frac{(p-1)pb^p}{1-b} + pzb^{p-1}.$$

The multicast of the START and INFO messages each counts for $n-1$ messages in the system, therefore, the total number of messages on all hops is $F_a + 2n - 2$.

In **S_CoordP**, each ACK message only travels one hop to the parent of the originator, hence there are $n-1$ ACK messages on all the hops and the total number of messages is $3n-3$.

Each member in **FullDist** multicasts an INFO message in the group, which turns out to be $n-1$ messages on all hops. Since there are n members, the total number of messages on all hops is $n(n-1)$.

Protocol	START	ACK	INFO	Total (exact)	Total
CoordP	$n-1$	F_a	$n-1$	F_a+2n-2	$O(n\log n)$
S_CoordP	$n-1$	$n-1$	$n-1$	$3(n-1)$	$O(n)$
FullDist	0	0	$n(n-1)$	$n(n-1)$	$O(n^2)$
S_CoordP	$n-1$	$n-1$	$n-1$	$3(n-1)$	$O(n)$
Train	1	F_d-1	F_d	$2F_d$	$O(n)$
S_Train	$n-1$	F_g	$n-1$	F_g+2n-2	$O(n)$

Table 4: Total number of messages on all hops in the system

In **Train**, the number of hops as a message traverses the tree from node 1 to 2, ..., to n, then back to node 1 is

$$F_d = 2\left[\frac{b^2(b^{p-1}-1)}{(b-1)^2} - \frac{b(p-1)}{b-1} - \frac{(p-1)(p-2)}{2}\right] + \left[2 + 2zb^{p-1} + \frac{2b(b^{p-1}-1)}{b-1} - 2p\right].$$

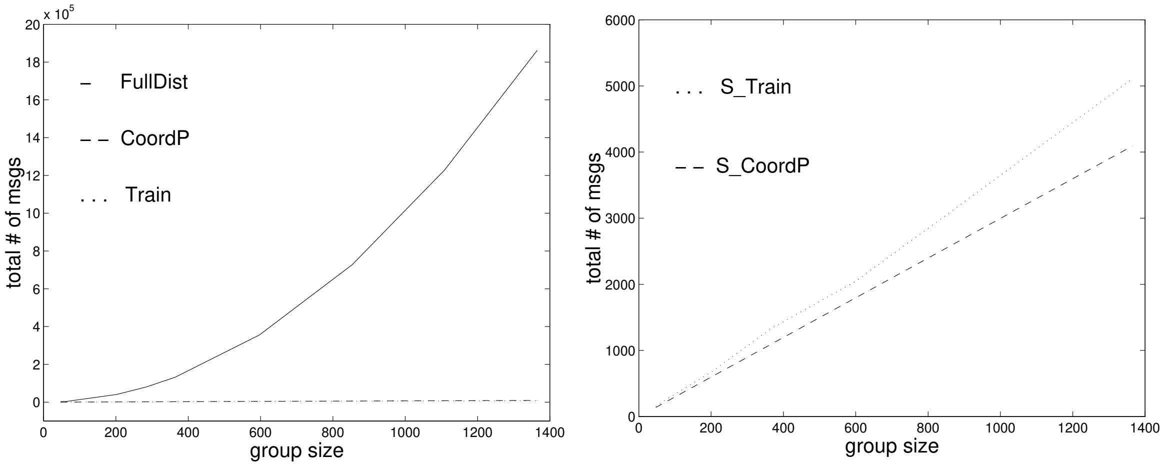
Messages travel around the circle twice, therefore, the total number of messages on all hops is $2F_d$. (Note that the root node is numbered as member 1. The nodes with depth k are numbered from left to right starting from $1+b+b^2+\cdots+b^{k-1}+1$, when k is odd, and numbered from right to left when k is even.)

In **S_Train**, the number of ACKs generated from one subgroup is $2(b-1)+1$ and since there are total of $\sum_{k=1}^{p-2} b^p$ subgroups above the bottom level, there are b^{p-1} subgroups at the bottom level and each group generates $2(z-1)+1$ ACKs, the number of ACKs is

$$F_g = [2(b-1)+1]\sum_{k=0}^{p-2} b^k + [2(z-1)+1]\,b^{p-1} = \frac{(2b-1)(b^{p-1}-1)}{b-1} + (2z-1)b^{p-1}.$$

And the total number of messages on all hops is F_g+2n-2.

The results for each protocol are presented in Table 4 and Figures 6 and 7. These results have also been verified by our simulation. The structured protocols reduce the total number of messages on all hops from their respective basic protocols by at least a factor of $\log n$. We expect better scalability and round-trip time from the structured protocols based on this information.



(A): the three basic protocols (B): the two structured protocols

Figure 6: Total number of messages on all hops

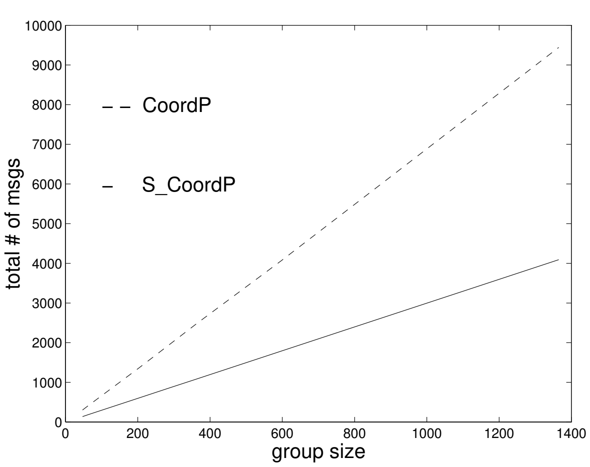
5.4 Simulation results

5.4.1 Maximum queue size over all the nodes in the system

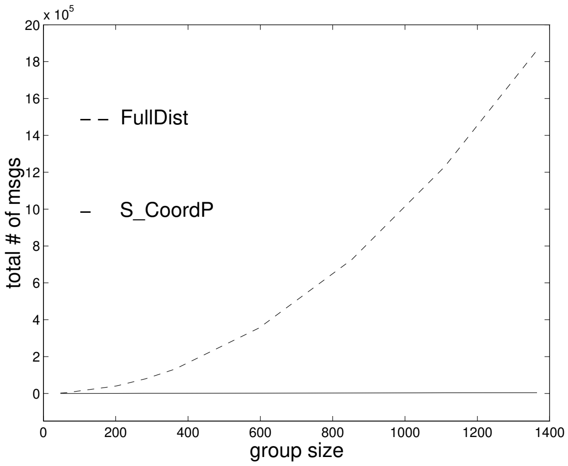
The maximum queue size recorded during the simulation is reported in this section. This number is an indicator of processing load at each node. This index is different from the total number of messages processed by a node, since the maximum queue size reflects how many messages are accumulated at each node at any moment.

Figure 8 shows that out of the three basic protocols, **FullDist** has the largest value for maximum queue size, followed by **CoordP**, both of which are increasing with the group size, while the number for **Train** stays flat at 1. For any internal node in protocol **CoordP**, the ACK messages from all of its descendants need to go through it to reach the root. Therefore their maximum queue size increases with n. The number for **FullDist** also increases with n simply because of the increased number of messages in the system. The number for **Train** stays at 1 since each node only needs to handle 2 messages.

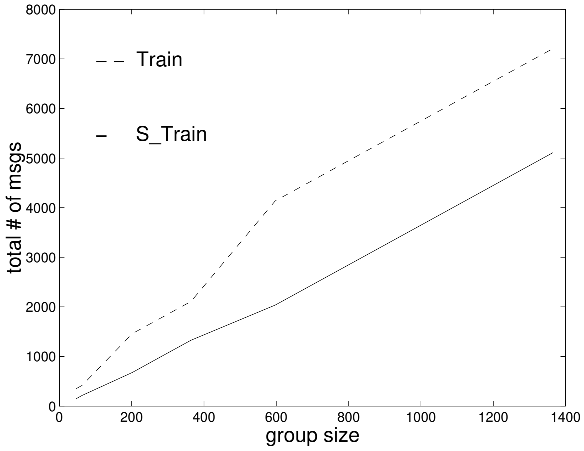
The maximum queue size stays constant at 1 in both structured protocols which is the result of the number of messages handled by each node being a small number. For **S_CoordP**, each node only needs to handle up to b ACK messages, with b varying from 2 to 4 in our simulation. For **S_Train**, each node needs to handle up to 4 messages.



(A): **CoordP** and **S_CoordP**



(B): **FullDist** and **S_CoordP**



(C): **Train** and **S_Train**

Figure 7: Total number of messages on all hops for the basic and correspoinding structured protocols

5.4.2 Round-trip time (RTT)

Round-trip time (RTT) is the time for one execution of the protocol. For a group member it is defined as the interval between the time it starts participating in one execution of the protocol and the time it receives the stability information of all the other members. At each member, it is measured as the time between receiving the START and the INFO message. We present the RTT measured at the first member (also the coordinator or the root) since the RTT at other members follows the same trend and is only slightly larger than that of the first member because the INFO message normally is $4n$ bytes and the START message is only 1 byte in most of the protocols. Another time index that reveals the property of the stability protocols is message stability time, also called time-to-stable (TTS). TTS is a function of RTT and the frequency

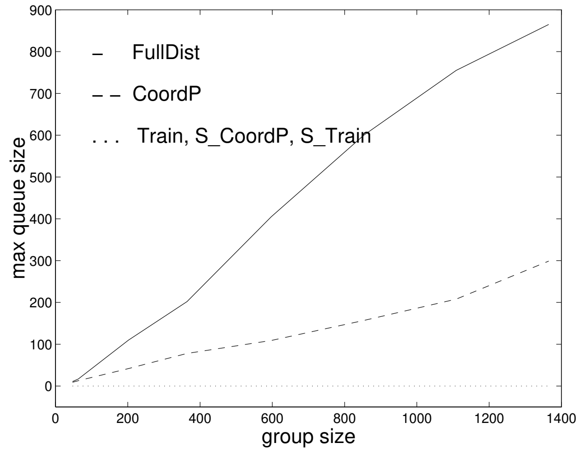
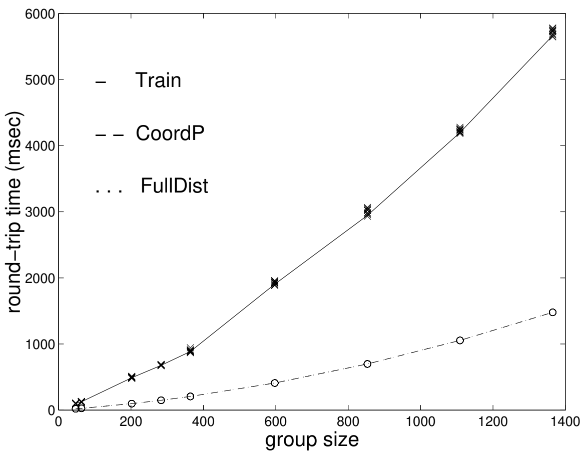


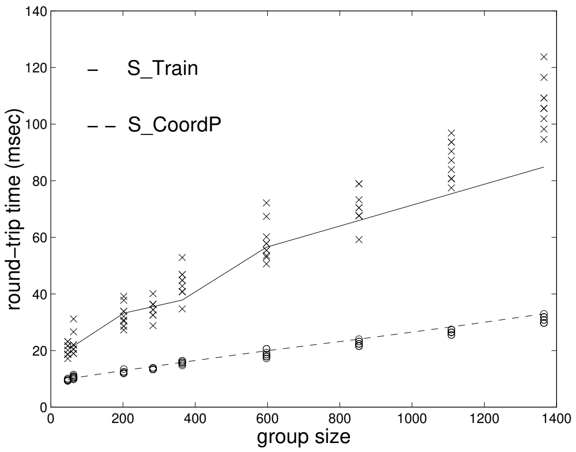
Figure 8: Maximum queue size over all the nodes for the five protocols

and mechanism to trigger the stability protocols. As a first step, we study RTT. The analysis of TTS is presented in Section 6.

RTT for the three basic protocols is shown in Figure 9(A). **Train** has the longest RTT, which is the result of the $2n$ message rounds. **CoordP** and **FullDist** both show increase of RTT as n increases, but since all of them have constant multicast message rounds, their increment is not as significant as **Train**. It is interesting to notice that **CoordP** and **FullDist** have almost identical RTT.



(A): the three basic protocols



(B): the two structured protocols

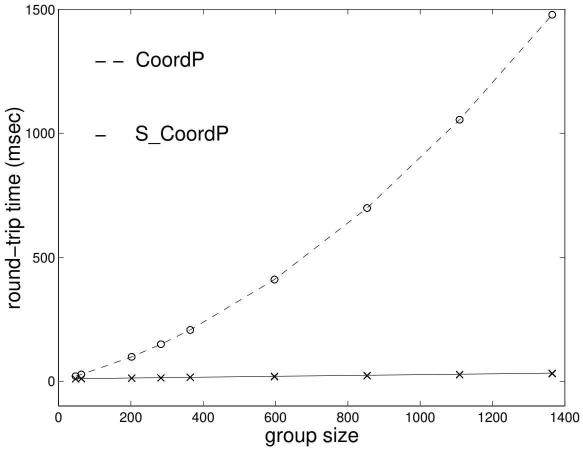
Figure 9: Round-trip time

Figure 9(B) displays the RTT for the two structured protocols. The trend is similar to Figure 9(A): **S_Train** shows significant increase of RTT when n is large, whereas the increase in **S_CoordP** is not as significant. This is because the number of messages rounds is $p+2$ in **S_CoordP**, and $bp+2$ in **S_Train**.

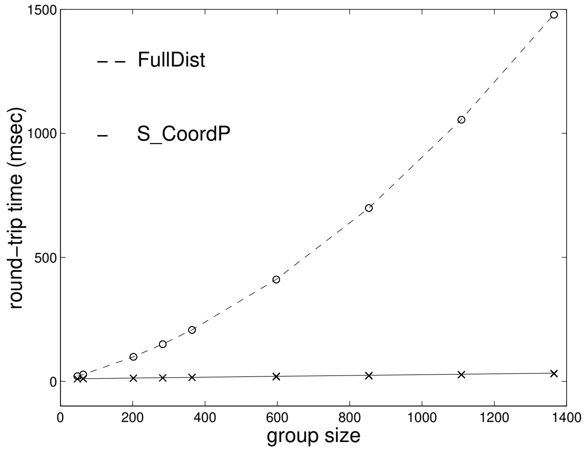
We have been studying the properties of various message stability protocols given the assumption that the underlying network topology is a near-complete tree. To confirm our belief that this is a reasonable approximation of the real network environment, and our analysis based on this assumption is valuable, we also measured RTT for each protocol in a set of random network environments. Given group size n, mean degree b for all the nodes and height p, we constructed a random tree as follows. One of the n nodes is assigned as the root, the remaining $n-1$ nodes are assigned to each level above the lowest level with probability $\frac{b^k}{n-1}$ to be in level k with $0 < k < p$. Since n is not necessarily equal to $1 + b + b^2 + \cdots + b^p = \frac{b^{p+1}-1}{b-1}$, we assigned the probability of a node being placed on level p to be $\frac{n - \frac{b^{p+1}-1}{b-1}}{n-1}$. After assigning each node to a level, we connected every node in level k randomly to a node in level $k-1$. Clearly, this is not a completely random tree. But we expect that this degree of randomness is appropriate to simulate the real clustered network environment. In such a setting, there are more computers connected to lower levels of the tree than to upper levels. For each given n in Table 3, we generated 10 random trees and plotted the corresponding RTTs in Figures 9 and 10. We can see that the RTTs for random trees are close to that of their corresponding near-complete trees.

Figure 10(A) shows the RTT for **CoordP** and **S_CoordP**. There are 3 rounds in **CoordP**, but $p+2$ rounds in **S_CoordP**. Contrary to what the message rounds suggest, we find that the RTT for **CoordP** increases dramatically as n increases, while the RTT for **S_CoordP** stays almost flat. In **CoordP**, for any node in the tree, ACK messages from all its descendants need to travel through the node in order to reach the root. The closer a node is to the root, the more ACK messages it needs to handle. This scheme will overload the root node, its neighbors and the links connecting them as n becomes large. As a result, ACK implosion will appear at the root and those nodes close to it. On the other hand, in **S_CoordP**, each ACK message travels only 1 hop to its parent. There is only one ACK message on any hop in the tree network. Each node only needs to handle up to b ACK messages, and, when the degree of the tree b is small (2 to 4 in our simulation), this will not cause implosion at any node. The RTT for **S_CoordP** is essentially the time spent for the START message to travel p hops down the tree plus the time for the ACKs to travel p hops up the tree to reach the root. Since there is no contention for the links, this time is proportional to the tree height p. As we set the tree height to be 5 in our simulation, we see a flat line for the RTT of **S_CoordP**. When $n = 1000$, **S_CoordP** offers 70 times improvement in RTT over **CoordP**. When $n = 20000$ in a tree with height 7, this improvement becomes 2100 times.

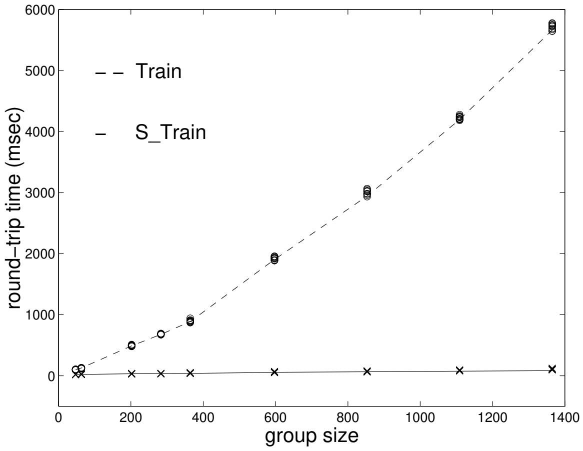
Figure 10(B) shows RTT for **FullDist** and its corresponding structured protocol **S_CoordP**.



(A): **CoordP** and **S_CoordP**



(B): **FullDist** and **S_CoordP**



(C): **Train** and **S_Train**

Figure 10: Round-trip time for the basic and corresponding structured protocols

We see a sharp increase in RTT for **FullDist** because of implosion at every member when n is large. RTT for **S_CoordP** stays flat because the reduced number of messages each member needs to handle eliminates the implosion problem.

The results for **Train** and its structured version **S_Train** are presented in Figure 10(C). There is no message implosion problem in either protocol. The RTT is proportional to the number of hops the ACK message from the n-th member needs to travel to reach the root. For **Train**, this number is $H_d = F_d$, whereas for **S_Train**, it is $H_h = p(2(b-1)+1)$, where $2(b-1)+1$ is the number of hops needed to travel through one level of the tree. Notice $H_d = O(n)$ and $H_h = O(\log n)$. This explains why **S_Train** scales significantly better than **Train**. When $n = 1000$, **S_Train** offers 35-fold improvement in RTT over **Train**. When $n = 20000$ in a tree with height 7, this improvement becomes 500-fold.

6 Stability triggering mechanism

In this section, we address the optimization issue for triggering stability tracking protocols. Assume that all members are sending messages at the same rate r messages per millisecond, and that each member has an output buffer of size g messages and an input buffer of size ng messages. Notice that when each member is sending at the same rate, during the time each sender sends g messages, it could at most receive ng messages from all the n members. The corresponding round-trip time (in milliseconds) for the stability protocol is expressed as $f = f(r)$, a function of data message rate r messages per millisecond. This is a very naive assumption. But we do know that the time for one round of stability protocol to complete is influenced by traffic load generated by data messages. We can say that $f(r)$ is a complicated function of which r is an important input variable.

The trigger works as follows. When the output buffer at the root node reaches X messages, the stability protocol is triggered. During the time period $f(r)$ of a round-trip, the number of data messages sent out will be $Q = rf(r)$. Then the number of messages in the root's output buffer will be $X + Q$, where X is the number of existing messages before the trigger, and Q is the number of newly sent messages since the trigger.

Suppose αX ($0 \leq \alpha \leq 1$) messages are stable at the end of a round of stability test, in other words, $1-\alpha$ is the failure ratio which stands for the percentage of data messages lost[4]. Then αX out of the X messages can be released from the top of the output buffer. If $(1-\alpha)X + Q < X$, then the root node waits until the number of messages in its output buffer reaches X. If $(1-\alpha)X + Q \geq X$, then the root node starts the next round of stability test. When the data messages fill up the buffer, all the senders stop sending data messages.

In this triggering mechanism, the message number X in the root node's output buffer should be chosen, given the measured round-trip time $f(r)$ for different protocols. If X is too large, multicast group members will constantly be forced to stop sending data messages since their buffers are full. If X is too small, the stability protocol will be triggered too often and the total number of messages in the system will increase unnecessarily. Therefore an optimal size for X should be determined according to the following set of constraints enforced on X by the buffer size:

[4]We do not assume the FIFO property is provided by the underlying protocols here. When FIFO is assumed, $\alpha = 1$, and it is a special case of the following discussion.

- The first round of stability test starts when there are X messages in the buffer, therefore,

$$X \leq g \tag{1}$$

  is a must.

- At the end of the first round, there are $X + rf$ messages in the buffer, so

$$X + rf \leq g \tag{2}$$

  must be satisfied.

- After the first round, αX messages can be released from the sender's output buffer. If $(1-\alpha)X + rf < X$, then the sender waits until X is filled to start the next round, so $X + rf \leq g$ is required. If $(1-\alpha)X + rf \geq X$, the second round is trigger immediately, rf messages are newly sent during the second round, in order to prevent buffer overflow,

$$(1-\alpha)X + rf + rf = (1-\alpha)X + 2rf \leq g \tag{3}$$

  is needed. In the worst case when $\alpha = 0$, Equation 3 becomes $X + 2rf \leq g$ which makes Equation 2 redundant.

Assume each member sends total of K messages in the test, and the sending rate is constant r when the output buffer is not full. The optimization problem can be described as follows. Given output buffer size g, message sending rate r, and round-trip time $f = f(r)$, choose X to minimize total number of stability tests $T(X) = \frac{K}{\alpha X}$, or equivalently to maximize X, subject to the following two constraints:

$$X + rf \leq g \tag{i}$$

$$(1-\alpha)X + 2rf \leq g \tag{ii}$$

(i) and (ii) can be simplified to

$$X \leq g - rf \tag{iii}$$

$$X \leq \frac{g - 2rf}{1-\alpha} \tag{iv}$$

which result in the optimal $X = max(g - rf, \frac{g-2rf}{1-\alpha})$. When given some special values of α, the optimal X becomes the following:

- When $\alpha = 0$, $X = max(g - rf, g - 2rf) = g - rf$.

- When $\alpha = 0.5$, $X = max(g - rf, 2(g - 2rf))$.
- When $\alpha = 1$, $2rf \leq g$ is satisfied by the assumption, hence $X = g - rf$.

In general,

- When $\alpha = 0$ or 1, the optimal $X = g - rf$.
- When $0 < \alpha < 1$, the optimal $X = max(g - rf, \frac{g-2rf}{1-\alpha})$.

Given the total number of messages K sent by each member, failure rate $(1 - \alpha)$ and the optimal X, the minimum number of tests needed is $T(X) = \frac{K}{\alpha X}$. If a smaller time-to-stable is needed, X has to be reduced, which results in an increase of $T(X)$.

6.1 Time-to-stable (TTS)

In a real network environment, messages will have a variable delay in the routers depending on the queue size at each router. Messages might get lost in the network because of buffer overflow at some intermediate nodes, and later get retransmitted. These non-deterministic factors will add more delay to the round-trip time and time-to-stable. Therefore, to use the stability triggering algorithm efficiently in practice, users should measure the $f(r)$ in the current system, then set X, Q, and g accordingly. Our simulation results are useful in the sense that they provide a lower bound in round-trip time $f(0)$ under the given tree shaped network structure. It also provides some insight on how to choose stability tracking algorithms under different circumstances.

7 Conclusion and future work

Message stability tracking protocols play an important role in distributed systems, distributed data base management systems, and parallel computing systems. The three basic protocols commonly used do not scale as the group size increases. By adding a tree structure to the basic protocols, we have derived two structured protocols with significant increase of scalability.

There are some techniques commonly used to improve the performance of stability protocols. Messages for stability protocols can be piggybacked on data messages to reduce traffic load in the network and the time processors spend handling messages received; only the changed sequence numbers need to be exchanged among group members in order to reduce the message size; some senders can be grouped together sharing one series of sequence numbers to reduce the size of the sequence number array (or stability matrix) kept at each member, therefore the size of ACK and INFO messages. In the extreme case, when the stability protocol is combined with a total ordering protocol, only 4 bytes of information is needed from each member, and the message size is reduced to a minimum. These techniques can be applied to both the basic and the structured protocols to further improve their performance.

We have simulated a set of stability protocols in a clustered LAN environment. Different network characteristics affect the behavior of these protocols. We are currently studying these protocols in a WAN environment.

References

[1] R. Ahuja, S. Keshav, and H. Saran. Design, implementation, and performance of a native mode ATM transport layer. *IEEE/ACM Transactions on Networking*, 4(4):502–515, August 1996.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *Digest of Papers, The 22nd IEEE International Symposium on Fault-Tolerant Computing Systems*, pages 76–84, July 1992.

[3] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 9(12):36–53, December 1993.

[5] R. Carr. The Tandem global update protocol, June 1985.

[6] F. Cristian. Asynchronous atomic broadcast. *IBM Technical Disclosure Bulletin*, 33(9):115–116 ???, February 1991.

[7] F. Cristian and S. Mishra. The pinwheel asynchronous atomic broadcast protocols. In *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, 1995.

[8] S. Deering. Host extensions for ip multicasting. Technical Report RFC 1112, August 1989.

[9] K. Guo, W. Vogels, and R. van Renesse. Structured virtual synchrony: Exploring the bounds of virtually synchronous group communication. In *Proceedings of the 7th ACM SIGOPS European Workshop*, September 1996.

[10] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An efficient reliable broadcast protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[11] J. Kay and J. Pasquale. The importance of non-data touching processing overheads in TCP/IP. In *Proceedings of ACM SIGCOMM'93*, San Francisco, CA, September 13-17 1993.

[12] S. McCanne and S. Floyd. Ns (network simulator). Available via http://www-nrg.ee.lbl.gov/ns, 1995.

[13] J. M. Mellor-Crummey and M. L. Scott. Synchronization without contention. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, CA, April 1991.

[14] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, April 1996.

[15] S. Mullender, editor. *Distributed Systems*. ACM Press, Addison-Wesley, 1993. page 104.

[16] E. Palmer. *Graphical Evolution: An Introduction to the Theory of Random Graphs.* John Wiley & Sons, 1985. page 99.

[17] M. Simmons, R. Koshela, and I. Bucher, editors. *Instrumentation for Future Parallel Computing Systems.* ACM Press, 1989.

[18] R. van Renesse. Masking the overhead of protocol layering. In *Proceedings of the ACM SIGCOMM'96*, pages 96–104, Stanford University, California, USA, August 1996.

[19] R. van Renesse, K. P. Birman, and S. Maffeis. Horus, a flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.