

Performance engineering of the Totem group communication system

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1998 Distrib. Syst. Engng. 5 78

(<http://iopscience.iop.org/0967-1846/5/2/003>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.212

The article was downloaded on 20/02/2012 at 07:58

Please note that [terms and conditions apply](#).

Performance engineering of the Totem group communication system*

R K Budhia[†], L E Moser[‡] and P M Melliar-Smith[§]

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA

Received 15 December 1997

Abstract. Group communication systems simplify the development of fault-tolerant distributed applications; however, concerns exist about the performance and overheads associated with such systems. This paper presents measurements of the performance of the Totem group communication system, operating on a single local-area network (LAN) and on multiple LANs interconnected by gateways. The Totem system runs in user space with standard commercial off-the-shelf (COTS) software and hardware. For 1 kbyte messages, a throughput of over 5000 messages per second has been measured for Totem on a single LAN, and an aggregate throughput of 10 000 messages per second has been measured for Totem on three LANs. At 2000 messages per second, the message latency is less than 3 ms. The paper also discusses some of what has been learned in engineering a group communication system for high performance.

1. Introduction

Many critical distributed applications require fault tolerance, but fault tolerance is difficult to program because information replicated on multiple computers must be kept consistent despite asynchrony and faults. Multicast group communication systems provide a foundation upon which fault-tolerant distributed applications can be built. By placing a total order on the delivery of messages, these systems make it easier to maintain the consistency of replicated information.

In the past, a disadvantage of group communication systems has been their poor performance because

- Messages must be delivered to, and acknowledgments must be obtained from, many destinations potentially leading to many messages.
- If several sources transmit in rapid succession to the same destinations, their receiver input buffers may overflow and messages may be lost.
- The need to deliver messages in order may delay some messages until other messages are delivered.

Improved designs of group communication systems, and improved hardware, have reduced the effects of these problems. While the hardware broadcast mechanisms of local-area networks (LANs) reduce the number of messages transmitted, clever acknowledgment mechanisms reduce the

number of acknowledgments, and effective flow control mechanisms ensure that very few messages are lost.

Nonetheless, careful design of the algorithms and fine tuning of the implementation are required to achieve high performance under realistic conditions. With modern high-clock-rate processors and a slow 10 Mbit/s Ethernet LAN, even a naive group communication system can achieve high utilization of the communication medium. A fast processor, carrying no application load, can process messages faster than a 10 Mbit/s Ethernet can deliver them and, thus, there is no risk of message loss in the input buffers. The performance of the protocol is determined by the media access control mechanisms of the LAN, and the mechanisms of the protocol are largely irrelevant. Therefore, it is more appropriate to evaluate a group communication system using a heavily loaded processor or, alternatively, using a communication medium that is faster than the processor, such as a 100 Mbit/s Ethernet. The efficiency of the protocol implementation and the effectiveness of the flow control mechanisms can then be evaluated more effectively. When the group communication system is used in a real application, adequate processor cycles must remain available for the application.

The Totem group communication system is a suite of multicast protocols that are designed to support complex high-performance fault-tolerant distributed applications. For such applications, many activities are processed concurrently and each activity must spend a significant amount of time waiting in queues for resources, processing or communication. The most important performance parameters are throughput (high throughput reduces queueing delays) and latency (delay) from message

* Research supported in part by DARPA Grant Nos N00174-93-K-0097 and N00174-95-K-0083.

[†] E-mail address: budhia_ravi@ntos.tandem.com

[‡] E-mail address: moser@ece.ucsb.edu

[§] E-mail address: pmms@ece.ucsb.edu

origination by an application process to message delivery to an application process under high load. Storage requirements for message buffering are also important. However, with the large inexpensive memories of computers (which are increasing in size at least as rapidly as processors and LANs are increasing in speed), storage requirements are rarely a limiting factor.

When multicasting 1 kbyte messages, Totem readily achieves a throughput of over 5000 messages/s on a single LAN and an aggregate throughput of 10 000 messages/s on three LANs. To the best of our knowledge, no prior publication of a reliable totally ordered group communication system has given performance measurements that exceed 1000 1 kbyte messages/s when run in user space with standard commercial off-the-shelf (COTS) software and hardware. Several systems (see the related work) are reported to achieve about 1000 1 kbyte messages/s using fast processors and a 10 Mbit/s Ethernet.

Moreover, Totem achieves a latency of 2.664 ms at a throughput of 1997 messages per second. Prior publications of other reliable totally ordered group communication systems exhibit similar latency but only under zero load. At realistic loads, their latencies are much larger and none achieves 2000 messages/s at any latency. However, the lack of a common hardware platform and similar operating conditions makes it difficult to compare the performance of different group communication systems, and implementing such a system is a major task. Therefore, we do not give any further detailed comparisons of the performance of various group communication systems; rather, we present the results we have obtained for Totem with various settings of the parameters and various configurations.

The remainder of this paper is organized as follows. In section 2 we highlight the relevant aspects of the Totem system, in particular, the key mechanisms for message ordering in the single-ring and multiple-ring protocols. In section 3 we present the performance results obtained for each of the protocols in terms of throughput and latency. Section 4 describes the methodology used for measuring the latency, and section 5 discusses factors that determine the performance of group communication systems. Section 6 presents related work, and section 7 concludes the paper.

2. The Totem system

The Totem system is a suite of multicast group communication protocols with a hierarchical structure that operates over a single LAN or over multiple LANs interconnected by gateways. The Totem single-ring protocol provides reliable totally ordered message delivery and membership services on a single LAN. The same services are provided by the multiple-ring protocol for multiple LANs.

The Totem process group layer allows messages to be addressed to specific process groups and to be delivered to the processes that are members of those groups. Typically, a distributed application consists of many, possibly intersecting, process groups. Each process group may span several processors and several LANs. Messages need only be transmitted to LANs hosting

relevant processes. Moreover, processors that do not host a particular process group, but receive messages broadcast to that process group, filter such messages before delivering them to the application. The multiple-ring protocol exploits such knowledge to improve the overall performance.

In Totem, messages are delivered to a process in a system-wide total order, but processes receive only those messages that are relevant to it. A system-wide total order allows processes in intersecting process groups to communicate without inconsistencies in the message order. Consistency of message delivery is guaranteed even when processes or processors fail and recover, and when networks partition and re-merge.

Totem provides two levels of message delivery, called *agreed* and *safe*. A processor can deliver a message in agreed order only if it has already delivered all prior messages in the total order. A processor can deliver a message in safe order only if it can deliver the message in agreed order and it determines that the message has been received by every other processor to which the message was addressed and multicast. When a processor determines that a message has become safe on its local ring, it can reclaim the buffer space used by that message, because it will never need to rebroadcast the message subsequently. Several other group communication systems describe a message as *stable*, with a meaning generally similar to safe delivery in Totem. However, the term *stable* is also used for several other, entirely distinct, concepts within distributed systems.

We now give a brief description of the Totem single-ring and multiple-ring protocols; more details can be found in [1, 2, 5, 19].

2.1. The Totem single-ring protocol

The Totem single-ring protocol uses a logical token-passing ring imposed on a broadcast domain to achieve reliable totally ordered delivery of messages. The token is a special message transmitted point-to-point that contains reliable delivery and total ordering information. The fields of the token include:

- seq*—the highest sequence number of any message broadcast on the ring (a high-water mark)
- aru*—a sequence number used to determine whether all processors have received all messages up to the message with this sequence number (a low-water mark).

To broadcast a message, a processor must possess the token. On receipt of the token, a processor first rebroadcasts any messages requested in the token. It then broadcasts its own new messages, incrementing the *seq* field of the token to obtain the message sequence numbers. Lastly, the processor updates the token and transmits it to the next processor on the ring.

If a processor has not received all messages up to the message with the sequence number contained in the *aru* field of the token, it reduces the *aru* accordingly. The processor that most recently lowered the *aru* can raise it again, except when *seq* and *aru* are equal. If the token makes a complete rotation without the *aru* being lowered, a processor can be certain that all processors on the ring

have received all messages up to that sequence number, i.e. that those messages are safe on the ring.

Gaps in the message sequence numbers denote missing messages; the retransmission of those messages is requested in the token. A message is delivered by the single-ring protocol to the multiple-ring protocol in sequence number order as soon as the conditions for delivery in agreed or safe order are met.

Totem's flow control mechanism limits the number of messages that a processor can broadcast when it holds the token. The flow control mechanism is based on two limits: the number of messages that can be broadcast by any one processor during a single token visit and the total number of messages that can be broadcast by all processors during a single token rotation. This mechanism ensures that the input buffers at the destinations seldom overflow. Moreover, when there are no messages broadcast on the ring, the rate at which the token circulates is reduced. Further details about Totem's flow control mechanism can be found in [1, 5].

2.2. The Totem multiple-ring protocol

Because a single token ring does not scale well, Totem includes a multiple-ring protocol that operates on top of the single-ring protocol over multiple LANs interconnected by gateways. The multiple-ring protocol scales much better.

In addition to the fields of the token for the single-ring protocol, the token now contains:

timestamp—a timestamp, obtained from a local Lamport clock

The timestamp is used to determine the system-wide total order on messages. Thus, when a processor receives a message or the token whose timestamp exceeds the value of its local Lamport clock, the processor advances its clock to a value greater than that timestamp. When a processor broadcasts a message or forwards the token, it uses the value of its Lamport clock as the timestamp of the message or token and increments the clock.

Messages are delivered by the multiple-ring protocol to the process group layer in timestamp order, ensuring the same delivery order at all processors system-wide. Before a processor can deliver a message in timestamp order, it must know that it has already received all relevant messages, from all of the connected rings, with timestamps less than the timestamp of the message. The multiple-ring protocol exploits the reliable delivery of the single-ring protocol for that knowledge.

On each ring, messages are generated with increasing sequence numbers and timestamps. The gateways forward messages in sequence number order from one ring to the next. Messages that are not needed in the direction of the forwarding are filtered by the gateways. A forwarded message is given a new sequence number for the next ring, so that it can be reliably delivered on that ring. The timestamp, however, remains unchanged.

Each processor maintains a *my_guarantee_vector* with an entry for each ring in the network, containing the highest timestamp of all of the messages received from

that ring. The message with the lowest timestamp among all of the undelivered messages can be delivered in agreed order, provided that its timestamp is less than all of the timestamps in that vector, because no messages with lower timestamps will be received subsequently. A gateway periodically generates a *Guarantee Vector message* for each ring to which it is attached, containing a copy of that ring's *my_guarantee_vector*.

A message *m* can be delivered in safe order if a *Guarantee Vector message* has been received, from every other ring, reporting receipt of a message from the source ring of *m* with a timestamp at least equal to the timestamp of *m*.

2.3. Implementation

The code for the Totem group communication protocols is written in the C programming language, and runs in user space on Sun workstations, running SunOS 4.x and Solaris 2.x with standard Unix System V calls, on 10 and 100 Mbit/s Ethernets. The code uses standard System V calls, standard network interfaces, and does not preempt the regular process scheduling of the operating system. The protocols have also been ported to HPs, running HP_UX, and to PCs, running Windows NT and is easily ported to other platforms.

3. Performance results

The measurements given here are for Totem running on Sun 1170 and Sun 2200 workstations, running Solaris 2.5.1, on 100 Mbit/s Ethernets. The three topologies for which the measurements were taken are shown in figure 1. The single-ring topology consists of nine Sun 1170 processors on a single 100 Mbit/s Ethernet. The two-ring topology consists of eight Sun 1170 processors, with four processors on each of two 100 Mbit/s Ethernets, and a Sun 2200 as the gateway connecting the two Ethernets. The three-ring topology consists of nine Sun 1170 processors spread across three 100 Mbit/s Ethernets, with two Sun 2200s as the gateways. All of the Sun 1170 processors generate and receive messages concurrently, except when the measurements were taken for a single sender. In this case all of the processors receive messages, but only one of them generates messages. The Sun 2200 gateways receive and forward, but do not generate, messages. All messages are transmitted to, and ordered by, all processors on the same LAN. The probability that a message must be forwarded through a gateway to an adjacent LAN is determined by a parameter.

For the measurements, a message size of 1 kbyte was used; the message size is the actual payload in an Ethernet packet, and does not include either the 46 bytes of UDP/IP/Ethernet packet header or the 68 bytes of Totem header. All of the performance results are based on the transmission of 100 000 messages. During a typical run, 100 to 1000 messages out of the 100 000 messages were lost and had to be retransmitted. These losses were probably due to unavoidable competing traffic on the Ethernet and

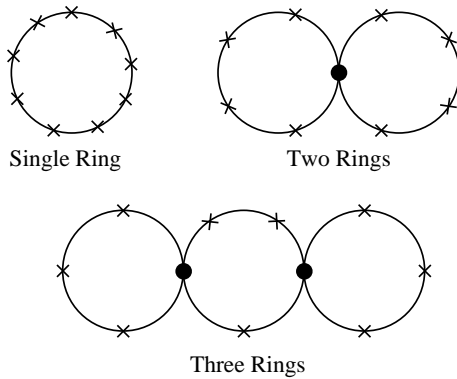


Figure 1. The three topologies used for the measurements.

to unavoidable competing activities within the operating system.

All of the measurements were performed for a deterministic message generation process. Each processor broadcasts messages at the maximum rate allowed by the flow control mechanisms, and has a full window size of messages to broadcast each time it receives the token. No measurements were made for a Poisson message generation process because the overheads and inaccuracies of the Unix timer delay mechanism preclude the use of such a process, even at 1000 messages/s.

We now give performance measurements for the Totem single-ring and multiple-ring protocols in terms of throughput, latency and useful utilization, defined as follows:

- *Throughput*—the total number of new messages broadcast and delivered in the network
- *Latency*—the time from origination of a message by the application until delivery, in total order, to the application at another processor
- *Useful utilization*—the percentage of the bandwidth of the network that is used for user data, discounting message headers.

3.1. The Totem single-ring protocol

3.1.1. Throughput Figure 2 shows the throughput and utilization for the single-ring protocol as a function of message size. The throughput is the total number of new messages delivered by the single-ring protocol at each processor. A different window size was used for each message size to obtain the best throughput for that message size. The window sizes ranged from 10 for 1400 byte messages, to 22 for 100 byte messages. The throughput measured was 5364 messages/s for 1000 byte messages, and 6951 messages/s for 100 byte messages. The useful utilization of the Ethernet, discounting message headers, is also shown in the graph. The maximum useful utilization was 55.76% for 1400 byte messages.

Figure 3 shows the throughput in number of ordered messages per second, as a function of message size, when message packing is used. Here, instead of each small message being sent separately with its own header,

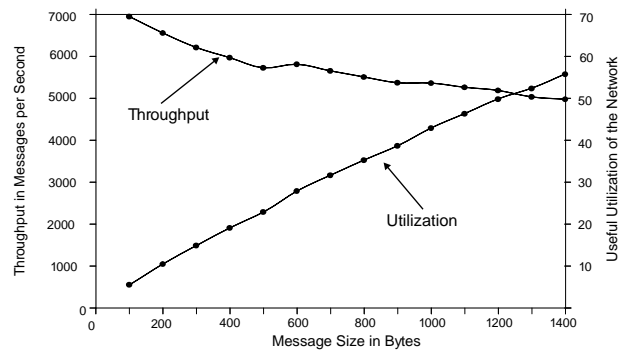


Figure 2. Throughput and utilization as a function of message size for the single-ring topology.

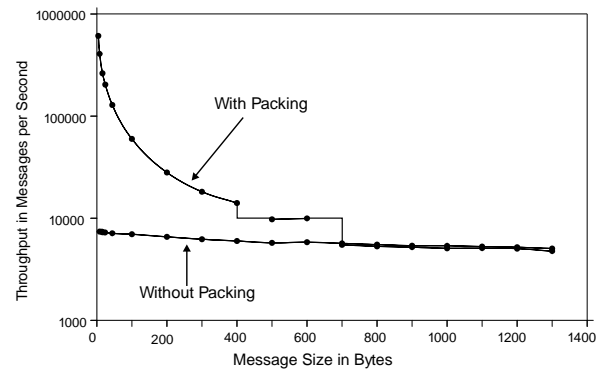


Figure 3. Throughput as a function of message size with and without message packing for the single-ring topology. Note the logarithmic scale on the vertical axis.

several small messages are packed into one large message with a single message header and two byte separators between them. Many applications use messages of 100 to 200 bytes. For 100 byte packed messages, a throughput of 59 577 messages/s was measured and, for four byte packed messages, a throughput of 607 576 messages/s was measured. The throughput with packing is compared with the throughput for unpacked messages in the figure. Note the logarithmic scale on the vertical axis. For small messages, different window sizes were used when transmitting packed messages and when transmitting unpacked messages, as explained in [20]. Even though the raw window size for packed messages is less than the corresponding window size for unpacked messages, the effective window size is greater, yielding better throughput for packed messages.

The effect of varying the window size on the throughput is shown in figure 4. These results are for all of the processors multicasting 1 kbyte messages. As the window size increases, the throughput also increases until a window size of 13 is reached. The decrease in throughput when the window size increases beyond 13 is probably due to message loss from input buffer overflow and consequent increase in the number of retransmissions. It is evident that any window size between 7 and 14 yields good throughput.

Figure 5 shows the throughput as a function of the number of processors for all processors sending messages and for only one processor sending messages, for 1 kbyte

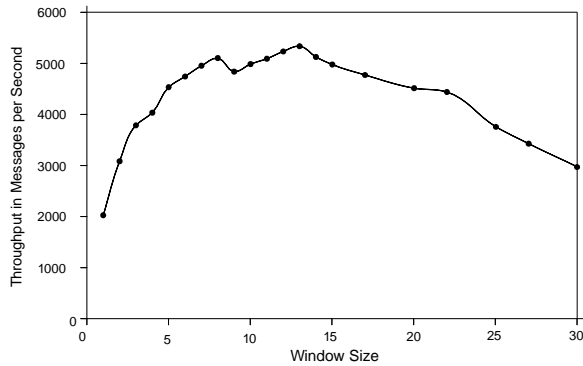


Figure 4. Throughput as a function of window size for the single-ring topology.

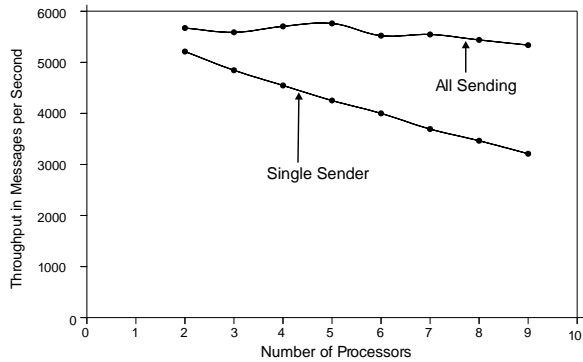


Figure 5. Throughput as a function of the number of processors for single and multiple senders for the single-ring topology.

messages. There is a slight degradation in going from multiple senders to a single sender. The optimal window sizes for multiple senders and a single sender vary by only one or two. Consequently, with a single sender, efficiency drops as the token must pass through an increasing number of non-senders.

3.1.2. Latency Figure 6 shows the mean latencies to agreed and safe delivery, as well as the token rotation time, as a function of the number of processors on the ring for a window size of one. The mean latency is the average time from origination of a message by the application until delivery, in total order, to the application at another processor, and includes the time waiting for the token to arrive before the message can be sent. The token rotation time includes the time to broadcast messages at each of the processors, as well as the time for the token to traverse the ring.

The mean latency for agreed delivery on a single ring is approximately one message transmission latency plus half a token rotation. The mean latency for safe delivery on a single ring is approximately one message transmission latency plus two token rotations (half a rotation for the sender to receive the token, half a rotation for the receiver to receive the token, and one more rotation for the message to become safe). The latency to safe delivery is important not only because an application may need safe delivery for critical operations, such as committing transactions, but also

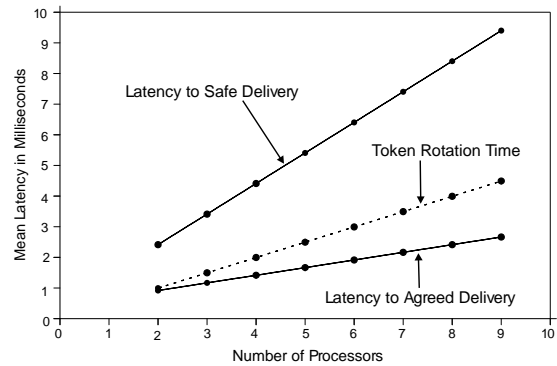


Figure 6. Latencies as a function of the number of processors for a window size of one for the single-ring topology.

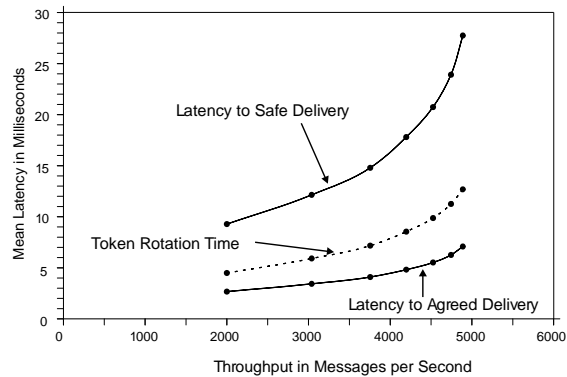


Figure 7. Latencies as a function of throughput for nine processors with all processors sending for the single-ring topology.

because a message may need to be retransmitted and thus cannot be discarded from the buffers until it becomes safe. The latency to safe delivery multiplied by the throughput determines the required buffer sizes.

Figure 7 shows the same quantities but as a function of throughput and measured for a ring of nine processors. The throughput is varied by varying the window size. Note that the latency to agreed delivery remains reasonable even up to very high loads. For the graphs shown in figures 6 and 7, 1 kbyte messages were used and all processors on the ring were broadcasting messages.

3.1.3. Token rotation time We also measured the token visit time for an unloaded system. This is the time for a processor to receive the token, process it, and then transmit it to the next processor on the ring. It includes two context switches between the kernel and the Totem code, the time required by Totem to process the token, and the time spent by the token on the Ethernet. This time was measured to be $240 \mu s$, which is independent of the number of processors on the ring. Thus, the token rotation time on an unloaded system is $240 * N \mu s$, where N is the number of processors on the ring.

3.2. The Totem multiple-ring protocol

3.2.1. Throughput Figure 8 shows the throughput for the single-ring, two-ring and three-ring topologies, for 1 kbyte messages, as a function of the probability that any particular message is forwarded by the gateway. The throughput is the total number of new messages multicast to, and ordered by, all of the processors to which they were multicast. For the multiple-ring topologies, a gateway does not generate any new messages of its own; it merely forwards the other processors' messages from one ring to the next. For the single-ring topology, all of the processors send and receive messages. The forwarding probability applies only to the multiple-ring topologies, and not to the single-ring topology. Different window sizes are used for different forwarding probabilities to obtain the best throughput at that forwarding probability.

A throughput of 7993 1 kbyte messages/s was obtained for the two-ring topology and a forwarding probability of 0.1, which exceeds that of the single ring for all forwarding probabilities less than 0.5. For the three-ring topology and a forwarding probability of 0.1, a throughput of 10 086 1 kbyte messages/s was obtained. This is higher than that for both the single-ring and two-ring topologies.

3.2.2. Latency Figures 9 and 10 compare the mean latencies to agreed and safe delivery, respectively, for the three topologies, as a function of throughput, for a forwarding probability of 0.1. It is evident that the improvement in latency for multiple-ring topologies is more pronounced for higher loads. As in figure 6, the throughput is varied by varying the window size. The advantage of the multiple-ring topologies is evident, especially for safe delivery.

With the small number of processors available in our laboratory, the advantage of multiple-ring topologies is most evident for low forwarding probabilities. At higher forwarding probabilities, the benefits of the multiple-ring topologies become more evident for larger numbers of processors. Figures 11 and 12 show the probability density functions for the latencies to agreed and safe delivery, calculated using a detailed analytic model [23]. Results are given for 40 processors on a single ring, on two

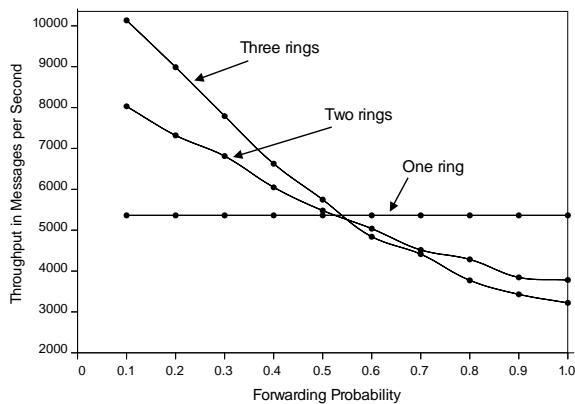


Figure 8. Throughput for the three topologies as a function of the forwarding probability.

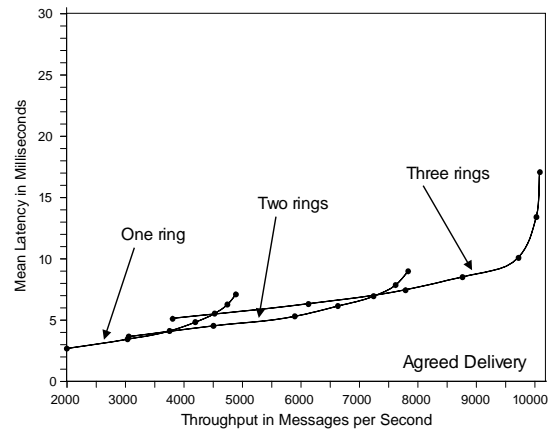


Figure 9. Latencies to agreed delivery for the three topologies as a function of throughput.

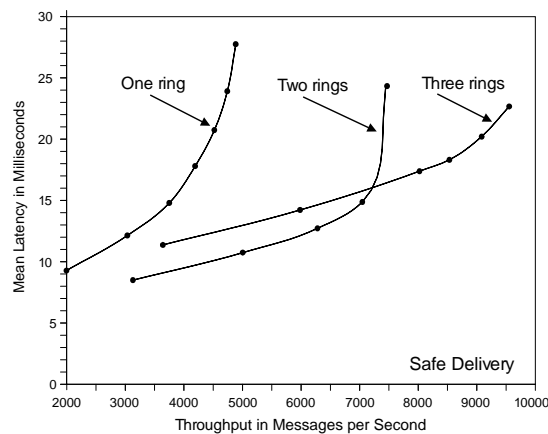


Figure 10. Latencies to safe delivery for the three topologies as a function of throughput.

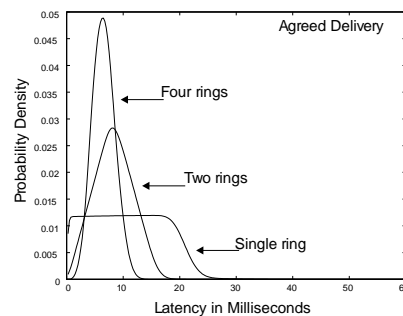


Figure 11. The pdfs for the latency to agreed delivery for 40 processors in the three topologies.

rings with a gateway, and on four rings in a ring-of-rings topology with four gateways, for a forwarding probability of 0.6. The message generation process is Poisson, and the aggregate throughput is 2000 new messages/s. As these graphs indicate, with multiple-ring topologies, both the mean latency and the variation in the latency are reduced, a characteristic of value in soft real-time systems.

The performance gain of the multiple-ring topologies over the single-ring topology is quite evident from these graphs. These graphs support the design philosophy that

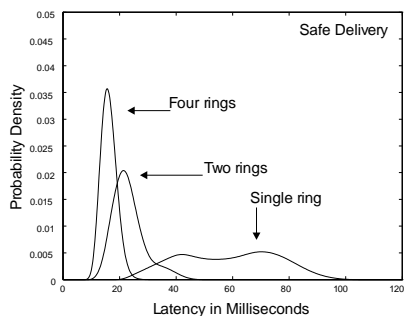


Figure 12. The pdfs for the latency to safe delivery for 40 processors in the three topologies.

we espouse, i.e. structure the application intelligently, so that the processes within a process group are assigned to processors on the same ring, and exploit the principle of locality, so that few messages need to be forwarded between rings.

4. Methodology for measuring the latency

Accurate measurement of one-way latencies in a distributed system is a difficult and challenging task. The very fact that extra processing must be performed to measure the latency distorts it! Moreover, messages are originated in, and delivered to, different processors, and the clock skews between the different processors affect the measurements. Round-trip latencies, on the other hand, are easy to measure, but less useful for the design of distributed applications.

In many systems, the clock skew between the different computers in the system does not vary significantly over time. A group of workstations, connected by a single LAN such as an Ethernet, or multiple interconnected LANs that are confined to the same geographical area, is an example of such a system. For such systems, we can eliminate the clock skew in the latency measurement through calculation.

In the first method of measuring the latency, a processor, say p_1 , appends its current clock value, say `transmit_time`, to the message being transmitted. The destination processor, say p_2 , reads its own clock immediately after receiving the message. The difference between the two clock readings is the message delivery latency L_i plus the clock skew S . Summing these latencies over M messages, processor p_1 measures

$$L_1 = \sum_M (L_i + S)$$

and processor p_2 measures

$$L_2 = \sum_M (L_i - S).$$

If M is large enough, we can estimate the mean latency L as

$$L = \frac{L_1 + L_2}{2 * M}.$$

For more than two processors in the system, each of them maintains an array of latencies for messages received from the other processors. These arrays allow us to

determine the mean latency in the system as a whole. In our system, this method allows us to measure latencies with an error of less than $10 \mu s$.

Complications can arise when measuring the communication latency for systems with variable clock skews. Systems in which the processors do not have reasonably stable clocks, systems with computers spread far apart geographically, and systems in which the latency must be measured over a long period of time, all fall into this category. For such systems, the clock skews can be eliminated provided that only the readings of a single processor's clock are compared.

In the second method of measuring the latency (as in the first method), a processor p_1 appends the `transmit_time` to the message being transmitted. Each processor also records the `transmit_time` of the last message received from each of the other processors in a `transmit_time_array`, as well as the time when that message was received in a `local_delay_array`. Whenever a processor p_2 broadcasts a new message, the `transmit_time_array` and the `local_delay_array` are also appended to the message. This information allows any other receiving processor, such as p_1 , to calculate the delay at p_2 between the receipt of a message and the transmission of the next message. Note that the transmission of the two arrays with each message adds overhead and may decrease the accuracy of the measured latency.

More specifically, suppose that processor p_1 broadcasts message m_1 at time T_1 , which is received by processor p_2 at time T_2 . Processor p_2 then broadcasts message m_2 at time T_3 , which is received by p_1 at time T_4 . We can estimate the mean one-way communication latency L by

$$L = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}.$$

Note that the clock values T_1 and T_4 are read by p_1 and that the clock values T_2 and T_3 are read by p_2 . Only the differences in the clock values read by the same processor are computed, thereby preventing problems that might otherwise arise due to clock skews and variations in the clock skews. In our system, this method allows us to measure latencies with an error of less than $15 \mu s$.

5. Factors that determine performance

We now discuss some of what we have learned while engineering Totem to achieve high performance.

First, a low message loss rate is essential for high performance. Message loss and resulting retransmissions waste network bandwidth and processor cycles, and delay the ordering of messages. At high load there are two primary causes for message loss on the Ethernet: collisions on the Ethernet and input buffer overflow at the destinations. Input buffer overflow occurs when messages are broadcast faster than a receiver can process them.

In Totem, collisions on the Ethernet are prevented by allowing only the processor holding the token to broadcast messages. Even then, flow control is required to avoid input buffer overflows. The rotating token and its window mechanism provide each processor with

accurate information about other processor's broadcasts and thus about their demands on buffer space. In Totem, an appropriate choice of the window size can almost completely eliminate input buffer overflow. An alternative flow control strategy is presented in [4]. Such strategies allow multiple processors to broadcast concurrently, and thus operate with incomplete information about the broadcasts of other processors. This makes it more difficult for the flow control algorithm to prevent buffer overflows under heavy loads. The simple flow control algorithm used in Totem yields high throughput and reasonably low latencies under heavy loads.

Some of the better performance figures that have been reported by other researchers use large window sizes, in the range of 20–30, even for 1 kbyte messages. Large window sizes permit increased throughput at the expense of increased latency. Totem achieves excellent performance, even for small window sizes, for 1 kbyte messages; for a window size of two, a throughput of over 3000 messages/s and a latency of 3.43 ms were obtained.

Effective buffer management requires that every processor knows that every other processor has received a message, so that it will not need to retransmit the message and thus can delete the message from its input buffers. Without such information, both reliable operation and buffer management become impossible. It is essential that every processor should transmit, quite frequently, at least an acknowledgment of the messages it has received. This is achieved by the circulating token in the Totem single-ring protocol and by the Guarantee Vector messages in the Totem multiple-ring protocol. The token also provides a heartbeat mechanism to allow rapid detection of faults.

High performance depends critically on a high-quality network interface. Currently, we are using the SunSwift FEPS 100 Mbit/s interface, which is quite good. Our experience with prior interface cards indicates that a better network interface can yield a factor of two or three improvement in performance.

In the past, much attention has been given to buffer management and message copying. Elaborate communication kernels have been designed to minimize the costs of these activities. Recent operating systems have, however, optimized commonly used buffer management system calls, such as *bcopy* and *malloc*. The time spent in these calls is now quite small and is not a significant factor in the performance of Totem. Rather, the performance of Totem appears to be determined almost entirely by context switching. Context switches from user space into the kernel and then back to user space are made whenever a message is received. Because the token is processed in user space, each token requires four context switches, two to receive the token and two to transmit it. In contrast, an ordinary message requires two context switches. For example, the time to broadcast a 1 kbyte message is approximately 110 μ s, which is slightly less than half the token visit time of 240 μ s. Unfortunately, faster processors do not necessarily provide proportionately faster context switching.

Contrary to popular belief, today's group communication systems are limited by the processor, rather than by

the communication medium. Network performance has recently been improving more rapidly than processing capabilities. Today, 100 Mbit/s Ethernet exceeds the capability of expensive workstations, even when those workstations are devoted 100% to communication. Because most users will wish to allocate the bulk of their processing resources to the application, rather than to the group communication protocol, it is unlikely, for several years to come, that processors will become fast enough to render inadequate the 1 Gbit/s Ethernet now being developed.

It is certainly possible to build systems of greater generality than Totem that can be tailored to the specific needs of individual applications, but such an approach does not necessarily lead to higher performance, even for applications that require a lesser service than total ordering of messages. Protocols like those of Totem, that have been carefully engineered and optimized to provide a single service very efficiently, can easily outpace more general protocols whose design and interfaces must retain more complexity to maintain that generality. Simpler algorithms with fewer options operate better.

Further performance results and discussion of performance engineering of group communication systems can be found in [8].

6. Related work

Reliable ordered group communication protocols can be classified as symmetric or asymmetric, depending on whether all nodes play the same role or some nodes are distinguished from others. The most symmetric protocols are those based on a causal order, discussed below. Clearly asymmetric are the static sequencer protocols, such as that of the Amoeba system [15, 16], in which messages are transmitted point-to-point to a central sequencer which multicasts them in order. Kaashoek and Tanenbaum have provided extensive performance measurements for Amoeba. Disadvantages of the central sequencer approach are that all messages are transmitted twice and that the sequencer may be a bottleneck and a single point of failure.

Part way between symmetric and asymmetric protocols are the rotating sequencer protocols. Here there are two major subclasses, the sequencing acknowledgment strategy and the token strategy. In the sequencing acknowledgment strategy, first described by Chang and Maxemchuk [9], processors broadcast messages at will, and unordered. One processor acts as a sequencer. It determines an order on the messages it has received and broadcasts a message that communicates that order to the other processors. Periodically, the role of the sequencer moves from one processor to the next in a predetermined order. Sequencer-based protocols exhibit low latency at low loads but suffer from flow control problems at high loads.

Several reliable ordered multicast protocols have been developed using the sequencer approach. While the Amoeba protocol does not rotate the sequencer, the RMP protocol [14] of Jia, Kaiser and Nett and the RMP protocol [26] of Whetten and Kaplan do rotate the sequencer. Performance measurements have been provided for both of those protocols. Another alternative is the Pinwheel

protocol [11] of Cristian and Mishra. Extensive simulations have been undertaken by Cristian, de Beijer and Mishra to compare four reliable ordered multicast protocols [10].

The use of a circulating token to sequence ordered multicasts was first described by Rajagopalan and McKinley [21]. This technique has been extensively developed in the Totem single-ring protocol. In these protocols, only the token holder is allowed to broadcast messages, and to determine the order of messages that it broadcasts. Token-ring ordered multicast protocols can provide high throughput, good flow control, and rapid detection of faults. However, the latency increases linearly with the size of the ring.

Some interesting variations of the token-based protocols are van Renesse's Total protocol [24,25] and the On-demand protocol [3] of Alvarez, Cristian and Mishra. The token moves in response to requests for the token, avoiding visits to sites with nothing to transmit so as to reduce the latency. Buffer management is, however, more difficult. Measurements have been provided for both the Total protocol and the On-demand protocol.

Another strategy is developed in the Newtop protocol [13] of Ezhilchelvan, Macedo and Shrivastava and the Hybrid protocol [22] of Rodrigues and Verissimo. These protocols are designed for systems in which some communication links are much slower than others. Typically, a rotating sequencer circulates through sites connected by high-speed links. Sites connected by low-speed links transmit their messages point-to-point to one of the sequencer sites, where they are ordered and multicast. Rodrigues and Verissimo have provided simulation results for their Hybrid protocol. Totem also uses a hybrid approach in its use of both single-ring and multiple-ring protocols.

Ordered multicast protocols based on the formation of a causal order are elegant, but suffer from high computational costs. The few performance measurements that are available clearly indicate that they are eclipsed by the sequencer and token-based approaches. Birman's Isis system [6,7] derives the causal order from a vector clock, which is effective for small groups. The Consul protocol [17] of Mishra, Peterson and Schlichting, and also the Trans protocol [18] of Melliar-Smith, Moser and Agrawala and its derivative, the Lansis protocol [4,12] of Amir, Dolev, Kramer and Malki, form the causal order from acknowledgments embedded in messages. The Total total ordering protocol implemented on top of Trans, and the Toto protocol implemented on top of Lansis, have the interesting characteristic that they continue to order messages even in the presence of faulty processors. All of the other ordered multicast protocols cited above block in the presence of processor faults until a membership algorithm has detected the fault and removed the faulty processor.

7. Conclusion

Fault-tolerant distributed applications are becoming steadily more important. As the price of computers is reduced and the performance of LANs is increased, the primary

limitation on the widespread use of fault-tolerant distributed applications is the difficulty and cost of programming them. Multicast group communication systems, particularly those that provide reliable totally ordered message delivery, can reduce that cost. Unfortunately, some prior totally ordered group communication systems have exhibited poor performance, which has limited their use in real applications.

The Totem group communication system has been designed for high performance and low overheads. Totem's totally ordered multicast messages make the programming of distributed systems easier, and remove many of the prior concerns about performance. It should now be possible for real applications to benefit from the reduced programming costs made possible by group communication systems. The lessons learned in engineering Totem to achieve high performance will also be applicable to other group communication systems.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive comments, which have greatly improved this paper. The authors also wish to thank P Narasimhan for the graphs of the throughput for the Totem single-ring protocol with and without packing given in figure 3, and E Thomopoulos for the graphs of the probability density functions for the latencies for the Totem single-ring and multiple-ring protocols given in figures 11 and 12.

References

- [1] Agarwal D A 1994 Totem: a reliable ordered delivery protocol for interconnected local-area networks *PhD Dissertation* Dept of Electrical and Computer Engineering, University of California, Santa Barbara, CA
- [2] Agarwal D A, Moser L E, Melliar-Smith P M and Budhia R K 1998 The Totem multiple-ring ordering and topology maintenance protocol *ACM Trans. Comput. Syst.* **16** (2)
- [3] Alvarez G A, Cristian F and Mishra S 1998 On-demand asynchronous atomic broadcast *Proc. 5th IFIP Int. Working Conf. on Dependable Computing for Critical Applications (Urbana-Champaign, IL, 1995)* (Los Alamitos, CA: IEEE Computer Society Press) pp 68–78
- [4] Amir Y, Dolev D, Kramer S and Malki D 1992 Transis: a communication subsystem for high availability *Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing (Boston, MA, 1992)* (New York: IEEE) pp 76–84
- [5] Amir Y, Moser L E, Melliar-Smith P M, Agarwal D A and Ciarfella P 1995 The Totem single-ring ordering and membership protocol *ACM Trans. Comput. Syst.* **13** 311–342
- [6] Birman K P and Clark T 1994 Performance of the Isis distributed computing toolkit *Technical Report* 94-1432, Dept of Computer Science, Cornell University, Ithaca, NY
- [7] Birman, K P and van Renesse R 1994 *Reliable Distributed Computing Using the Isis Toolkit* (Los Alamitos, CA: IEEE Computer Society Press)
- [8] Budhia R K 1997 Performance engineering of group communication protocols *PhD Dissertation* Dept of Electrical and Computer Engineering, University of California, Santa Barbara, CA

- [9] Chang J and Maxemchuk N 1984 Reliable broadcast protocols *ACM Trans. Comput. Syst.* **2** 251–73
- [10] Cristian F, de Beijer R and Mishra S 1994 A performance comparison of asynchronous atomic broadcast protocols *Distrib. Syst. Engng* **1** 177–201
- [11] Cristian F and Mishra S 1995 The pinwheel asynchronous atomic broadcast protocols *Proc. 2nd Int. Symp. on Autonomous Decentralized Systems (Phoenix, AZ, 1995)* (Los Alamitos, CA: IEEE Computer Society Press) pp 215–21
- [12] Dolev D, Kramer S and Malki D 1993 Early delivery totally ordered multicast in asynchronous environments *Proc. 23rd IEEE Int. Symp. on Fault-Tolerant Computing (Toulouse, France, 1993)* (Los Alamitos, CA: IEEE Computer Society Press) pp 544–53
- [13] Ezhilchelvan P D, Macedo R A and Shrivastava S K 1995 Newtop: A fault-tolerant group communication protocol *Proc. 15th IEEE Int. Conf. on Distributed Computing Systems (Vancouver, Canada, 1995)* (Los Alamitos, CA: IEEE Computer Society Press) pp 296–306
- [14] Jia W, Kaiser J and Nett E 1996 RMP: fault-tolerant group communication *IEEE Micro* **16** (2) 59–67
- [15] Kaashoek M F and Tanenbaum A S 1991 Group communication in the Amoeba distributed operating system *Proc. 11th IEEE Int. Conf. on Distributed Computing Systems (Arlington, TX, 1991)* (Los Alamitos, CA: IEEE Computer Society Press) pp 882–91
- [16] Kaashoek M F and Tanenbaum A S 1996 An evaluation of the Amoeba group communication system *Proc. 16th IEEE Int. Conf. on Distributed Computing Systems (Hong Kong, 1996)* (Los Alamitos, CA: IEEE Computer Society Press) pp 436–47
- [17] Mishra S, Peterson L L and Schlichting R D 1993 Consul: a communication substrate for fault-tolerant distributed programs *Distrib. Syst. Engng* **1** 87–103
- [18] Melliar-Smith P M, Moser L E and Agrawala V 1990 Broadcast protocols for distributed systems *IEEE Trans. Parallel Distrib. Syst.* **1** 17–25
- [19] Moser L E, Melliar-Smith P M, Agarwal D A, Budhia R K and Lingley-Papadopoulos C A 1996 Totem: a fault-tolerant multicast group communication system *Commun. ACM* **39** 54–63
- [20] Narasimhan P, Moser L E and Melliar-Smith P M 1996 Message packing as a performance enhancement strategy with application to the Totem protocols *Proc. IEEE Conf. on Global Telecommunications: GLOBECOM'96 (London, 1996)* vol 1 (New York: IEEE) pp 649–53
- [21] Rajagopalan B and McKinley P K 1989 A token-based protocol for reliable, ordered multicast communication *Proc. 8th IEEE Symp. on Reliable Distributed Systems (Seattle, WA, 1989)* (Washington, DC: IEEE Computer Society Press) pp 84–93
- [22] Rodrigues L E T, Fonseca H and Verissimo P 1996 Totally ordered multicast in large-scale systems *Proc. 16th IEEE Int. Conf. on Distributed Computing Systems (Hong Kong, 1996)* (Los Alamitos, CA: IEEE Computer Society Press) pp 503–10
- [23] Thomopoulos E, Moser L E and Melliar-Smith P M 1996 Latency analysis of the Totem protocols *Technical Report #96-17*, Dept of Electrical and Computer Engineering, University of California, Santa Barbara, CA
- [24] van Renesse R, Hickey T M and Birman K P 1994 Design and performance of Horus: a lightweight group communication system *Technical Report 94-1442*, Dept of Computer Science, Cornell University, Ithaca, NY
- [25] van Renesse R, Birman K P and Maffei S 1996 Horus: a flexible group communication system *Commun. ACM* **39** 76–83
- [26] Whetten B and Kaplan S 1995 A high performance totally ordered multicast protocol *Proc. Int. Workshop on Theory and Practice in Distributed Systems (Dagstuhl Castle, Germany, 1994)* ed K P Birman *et al* (Berlin: Springer) pp 33–57