Totem: A Fault-Tolerant Multicast Group Communication System

L. E. Moser, P. M. Melliar-Smith,
D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulos
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106

Abstract

When Totem delivers multicast messages, it invokes operations in the same total order throughout the distributed system. The result: consistency of replicated data and simplified programming of applications.

Introduction

Many applications can benefit from distributed systems based on multiple computers interconnected by a communication network. Distributed systems use inexpensive high-performance computers and can be configured closely to the application. Information can be replicated on several processors to improve performance and to provide fault tolerance. However, programming distributed applications is difficult, particularly when replicated information must remain consistent as it is updated in the presence of faults. Many messages may be required and recovery from faults may introduce delays, making real-time performance objectives difficult to achieve.

Ordered multicast group communication systems are a useful infrastructure on which complex distributed applications can be built. Isis [4], Horus [18], Trans/Total [12, 15], Transis [6], Amoeba [8] and Delta-4 [17] are other examples of such systems.

The Totem system, developed at the University of California, Santa Barbara, provides reliable totally ordered multicasting of messages over local-area networks (LANs), and exploits the hardware broadcasts of such networks to achieve high performance. (See the sidebar, "Why Totem?") Total ordering of messages simplifies the programming of fault-tolerant distributed applications. If distributed operations are derived from the same messages in the same total order, consistency of replicated information is easier to maintain. Simplified programming results in fewer programming errors and increased reliability for the application.

Totem is intended for complex applications in which both fault tolerance and real-time performance are critical. Such complex applications are typically built as asynchronous event-driven distributed systems. The types of applications that can benefit from Totem's totally ordered message delivery service include many of the computer systems that are most important to our society, for example, air traffic control, industrial automation, transaction processing, banking, stock market trading, intelligent highway, medical monitoring and replicated database systems.

The characteristics that make Totem suitable for complex applications, particularly soft real-time applications, include

- High throughput and low predictable latency
- Rapid detection of, and recovery from, faults
- System-wide total ordering of messages, even for systems in which the network can partition and remerge, and for systems in which process groups can intersect
- Scalability to larger systems based on multiple LANs, interconnected by gateways, within the same geographical area.

With Totem, correctness of message ordering and configuration changes is ensured, even in the presence of multiple faults. Yet, excellent performance is achieved.

Totem Services

The Totem system provides reliable totally ordered multicasting of messages to processes within process groups over a single LAN or over multiple LANs interconnected by gateways. Totem provides this delivery service in the presence of various types of communication and processor faults, including message loss, network partitioning, and processor crash, omission and timing faults, but not completely arbitrary faults.

The structure of the Totem system as a hierarchy of protocols is shown in Figure 1. With reference to this hierarchy, we say that a message is *received* from the next lower layer of the hierarchy and is *delivered* to the next higher layer. When messages are

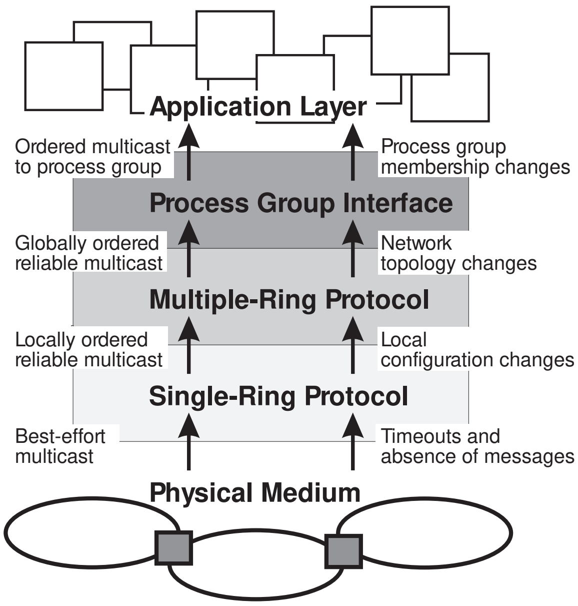

Figure 1: The Totem system hierarchy.

received, they may not be in the correct order and, thus, may need to be reordered before being delivered to the next higher layer.

The bottom layer of the Totem system hierarchy is a best-effort multicast service, which typically uses UDP to exploit the high-performance hardware broadcasts of LANs. The single-ring protocol converts the best-effort multicasts into the service of reliable totally ordered delivery of messages on a single LAN and, in addition, provides fault detection, recovery and configuration change services. The multiple-ring protocol uses the single-ring protocol and provides global total ordering of messages, as well as network topology maintenance services. The multiple-ring protocol, using information from the process group interface above it, forwards messages through the gateways to the LANs on which they are required. The process group interface delivers messages to the application processes in the appropriate process groups and provides process group membership services. The Totem system can also operate with the process group interface directly on top of the single-ring protocol.

The Totem system provides two reliable totally ordered message delivery services, as requested by the originator of the message, called agreed and safe:

- *Agreed delivery* guarantees that, when a processor delivers a message, it has already delivered all prior messages originated by processors in its current configuration and timestamped within the duration of that configuration.

- *Safe delivery* further guarantees that before a processor delivers a message, it has determined that every other processor in its current configuration has received the message. Safe delivery is useful, for example, in transaction processing systems where a transaction must be committed by all of the processors or none of them.

Both of these services deliver messages in a single system-wide total order (linear sequence) that respects Lamport's causal order [10]. A processor may not need to deliver all of the messages and, in the presence of faults, it may not be able to deliver all of them. When a processor fails or the network partitions, it may be impossible to determine which messages were delivered in which order by the processor before it failed, or whether messages were delivered by processors in other components of the partitioned network. Delivery of messages in a consistent system-wide total order is not easy when faults can occur.

Extended virtual synchrony [14] ensures that the above agreed and safe delivery guarantees are honored within every configuration, even if faulty processors are repaired or if a partitioned network remerges. (See the sidebar, "Virtual Synchrony and Extended Virtual Synchrony.") When a fault occurs, a transitional configuration with a reduced membership is introduced, all members of which are able to honor the delivery guarantees. If the network partitions, processors in different components of the partitioned network may deliver different messages, but they never deliver the same messages in different orders. For many applications, this is a very important property.

Consider, for example, a commercial enterprise in which purchases received over the Internet before the close of business in New York are handled by the New York office and after are handled by the San Francisco office. If the system were to partition at the critical moment (without extended virtual synchrony), it is possible that in New York the close of business message is ordered before the purchase message, while in San Francisco it is ordered after the purchase message. Both offices would then regard the other office as being responsible for the purchase.

Interestingly, extended virtual synchrony can be guaranteed only if messages are born-ordered, meaning that the relative order of any two messages is determined directly from the messages, as broadcast by their sources. The Totem system uses born-ordered messages, but some other multicast group communication systems do not.

The Totem Single-Ring Protocol

The Totem single-ring protocol [1, 3, 13] provides reliable totally ordered delivery of messages using a logical token-passing ring superimposed on a LAN, such as an Ethernet. The token circulates around the ring as a point-to-point message, with a token retransmission mechanism to guard against token loss. Only the processor holding the token can broadcast a message. The token, shown in Figure 2, provides total ordering of messages, rapid detection of faults, and effective flow control.

Message Ordering

In the Totem single-ring protocol, a sequence number field in the token, called *seq*, provides a single sequence of message sequence numbers for all messages broadcast on the ring, and thus a total order on messages. When a processor broadcasts a new message, it increments the *seq* field of the token and gives the message that sequence number. Other processors recognize missing messages by detecting gaps in the sequence of message sequence numbers, and request retransmissions by inserting the sequence numbers of the missing messages into the retransmission request list of the token. If a processor has received a message and all of its predecessors, as indicated by the message sequence numbers, it can deliver the message as an agreed message.

The all-received-upto field, or *aru*, of the token enables a processor to determine, after a full token rotation, a sequence number such that all processors on the ring have received all messages with lower sequence numbers. A processor can deliver a message as a safe message if the sequence number of the message is less than or equal to this sequence number. When a processor delivers a message as safe, it can reclaim the buffer space used by the message because it will never need to retransmit the message subsequently.

One might think that the continuously circulating token would result in increased overhead and reduced performance. The performance of other ordered multicast protocols is limited by input buffer overflow under high loads, causing message loss and retransmission. In Totem, the token provides accurate information on how many messages were transmitted during the prior token rotation. Using this information, Totem's flow control mechanism limits transmissions to ensure that input buffers seldom overflow, allowing Totem to operate at higher throughput than other protocols.

The token also provides information about the aggregate message backlog of the processors on the ring, allowing a fairer allocation of bandwidth to processors than simpler schemes such as FDDI. The Totem flow control mechanism provides excellent protection against fluctuations in processor loading but is vulnerable to competition for

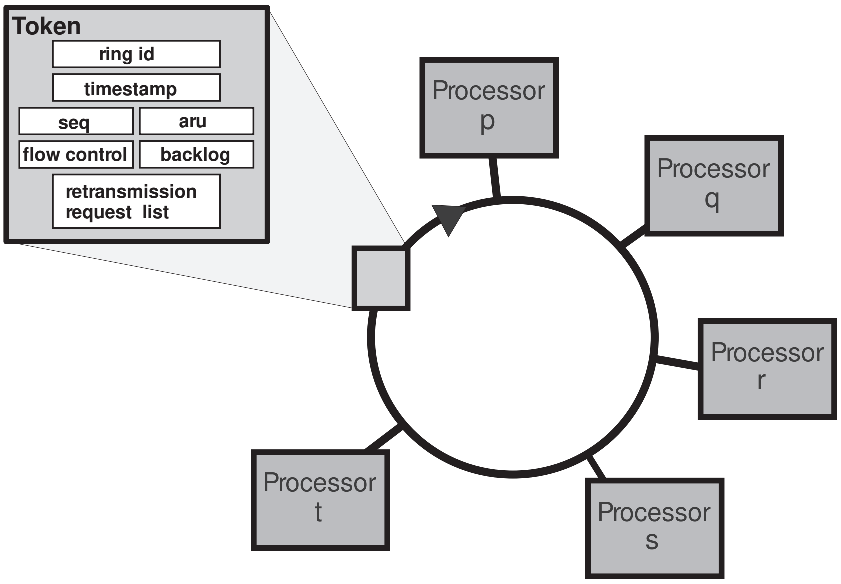

Figure 2: The token of the Totem single-ring protocol being passed from one processor to the next.

the input buffers from unanticipated network traffic on the LAN. Under high loads, Totem incurs relatively little variation in the latency to message delivery, an important factor for real-time applications.

Local Configuration Services

The Totem single-ring ordering protocol is integrated with a membership protocol that provides a membership or *configuration* service for a single LAN, including addition of new and recovered processors and deletion of faulty processors. Faulty processors are detected by timeouts. New or restarted processors are detected by the appearance of messages on the LAN from processors that are not members of the current ring. Like Transis [6], Totem handles network partitioning and remerging of components of a partitioned network.

The Totem single-ring membership protocol ensures

- Consensus – Every member of a configuration agrees on the membership of that configuration.
- Termination – Every processor installs some configuration with an agreed membership within a bounded time unless it fails within that time.

Subject to these consensus and termination requirements, the membership protocol aims to form as large a membership as possible.

The well-known Fischer, Lynch and Paterson impossibility result [7] has demonstrated that, in a completely asynchronous system, it is impossible for processors to reach consensus in the presence of even a single crash failure. Chandra and Toueg [5] have shown, however, that consensus is possible in an asynchronous system that is subject to faults if an unreliable failure detector is provided. We employ this strategy. The failure detector of Totem uses timeouts, and may exclude a slow processor from the membership, even though it has not actually failed.

The Totem single-ring membership protocol achieves consensus in bounded time, even if further faults occur, by reducing the membership until consensus is reached and by using timeouts that bound the time spent in any state of the membership protocol. The protocol can terminate in a singleton membership; however, with an appropriate choice of timeouts and with judicious use of randomization, the probability of a singleton membership is very small.

After reaching consensus on the membership, the membership protocol constructs a new ring on which the ordering protocol can resume operation, generates a new token, and recovers messages that had not yet been received when the fault occurred. To install a new regular configuration, the protocol delivers two Configuration Change messages, rather than the one message that might have been expected. The first Configuration Change message introduces a transitional configuration of reduced size that excludes the faulty or inaccessible processors. Delivery of this message indicates that the agreed and safe delivery guarantees now apply only to the smaller transitional configuration. Within the transitional configuration, the remaining messages of the old configuration are delivered. After these messages are delivered, the second Configuration Change message is delivered, introducing the new regular configuration.

The Totem Multiple-Ring Protocol

The Totem multiple-ring protocol [1, 2, 13] operates over multiple LANs interconnected by gateways. Imposed on each LAN is a logical token passing ring on which the single-ring protocol operates. The multiple-ring protocol provides essentially the same services, with the same properties, as the single-ring protocol. In particular, the message ordering service provides agreed and safe delivery, and the topology maintenance service provides consensus and termination for changes in the topology.

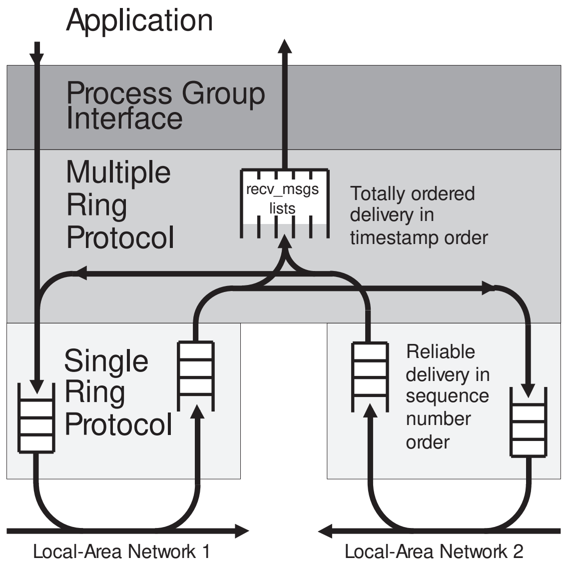

Figure 3: The operation of the Totem multiple-ring protocol at a gateway.

Message Ordering

To achieve a global total order of messages over all rings, the Totem multiple-ring protocol uses Lamport timestamps[1] and delivers messages in timestamp order. Messages with the same timestamp are delivered in the order of their source ring identifiers. Delivery of messages in timestamp order guarantees global consistency of message ordering. However, before a processor can deliver a message in timestamp order, it must know that it will never subsequently receive a message with a lower timestamp.

Messages are generated with increasing timestamps and sequence numbers on each individual ring. The gateways forward messages in sequence number order from one ring to the next, as shown in Figure 3. When a gateway broadcasts a forwarded message, it gives the message a new sequence number for the next ring so that the message can be reliably delivered on that ring. The timestamp of the message, however, remains unchanged. The single-ring sequence numbers (which contain no gaps), together with forwarding of messages in sequence number order, ensure that there are no missing messages.

[1]The Lamport timestamp of a message is obtained from a local Lamport clock. If a processor receives a message whose timestamp exceeds the value of its clock, the processor advances its clock to a value greater than the timestamp.

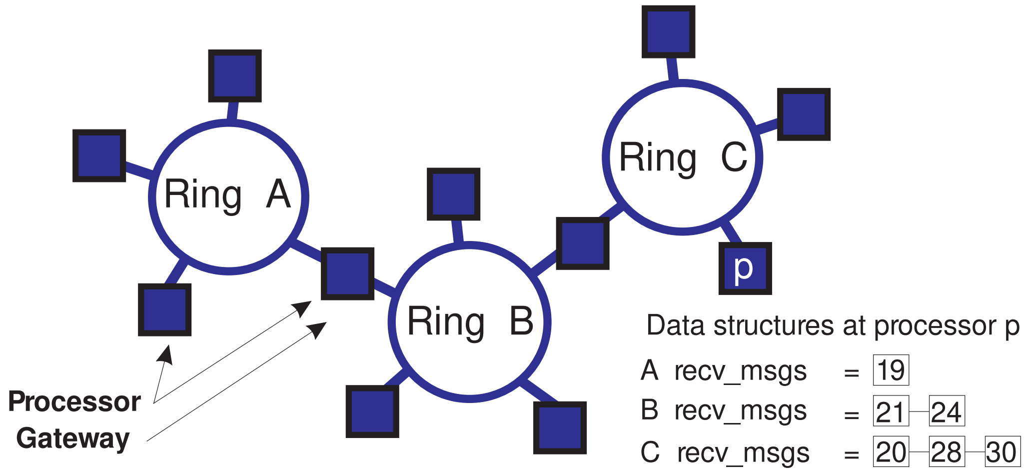

Figure 4: The *recv_msgs* lists for the Totem multiple-ring protocol.

For each ring from which it might receive a message, a processor maintains a *recv_msgs* list of messages originated on that ring and received from the single-ring protocol, as shown in Figure 4. A processor can deliver a message as an agreed message, and remove it from the *recv_msgs* list, if the message has the lowest timestamp of all messages in the *recv_msgs* lists and if none of the *recv_msgs* lists is empty. Because messages from the same source ring are forwarded in the order of their sequence numbers, which is also the order of their timestamps, a processor then knows that it will never receive a message with a lower timestamp from that ring.

The gateways periodically broadcast messages, called Guarantee Vector messages, for the rings to which they are attached. The Guarantee Vector messages ensure that a processor can continue to deliver messages as agreed messages even if, for some ring, no processor on that ring has recently originated a message. The Guarantee Vector messages also report which messages have been received on a ring from each of the other rings and, thus, allow a processor to determine which messages can be delivered as safe messages.

Network Topology Maintenance

In the Totem multiple-ring protocol, each gateway maintains a data structure, called *topology*, listing the rings within its connected component and the gateways that interconnect them. The topology of the network may be completely arbitrary. Since the gateways have knowledge of the network topology, they can adapt the message routing strategy to the current topology. A processor that is not a gateway needs to know only the rings from which it can expect to receive messages, rather than the full topology of the network.

If messages are originated on a ring of which a processor is unaware, it will not wait for such messages and may prematurely deliver other messages with higher timestamps. Similarly, if a ring becomes inaccessible and the processor is not informed, it will wait for a message from that ring and message ordering will stop.

Processor faults and network partitioning are detected by the single-ring protocol, which generates a Configuration Change message to report the change in the local ring. Each gateway on the ring analyzes the Configuration Change message to determine its effect on the topology. The multiple-ring protocol then generates and broadcasts a Topology Change message that reflects the change. In particular, if a gateway finds that a ring has become inaccessible, the gateway removes the ring from its topology and notifies the other processors and gateways using a Topology Change message. This removal of the ring ends the need to wait for messages from that ring and allows messages from other rings to be ordered. Similarly, a Configuration Change message and its consequent Topology Change message can report that a ring is being added to the topology.

A topology change must have the same effect for each of the processors and gateways that were previously able to, and can still, communicate with each other. Although the processors and gateways may learn of a topology change at different physical times, they must still agree on a common logical time for the topology change and also on the set of messages delivered before the topology change. To accomplish this, Configuration Change and Topology Change messages are timestamped, and are delivered in timestamp order along with the regular messages.

The Totem Process Group Interface

An application running on top of Totem (and also on top of other group communication systems) is structured as a collection of process groups. Each process group is a set of processes that cooperate to perform a particular task of the application. A process can be a member of several intersecting process groups, and a process group can span several rings, as shown in Figure 5. Each message is addressed to one or more process groups and is delivered to the processes that are members of those process groups.

The Totem process group interface [11] provides the services of sending and receiving messages addressed to process groups and of creating, joining and leaving process groups.

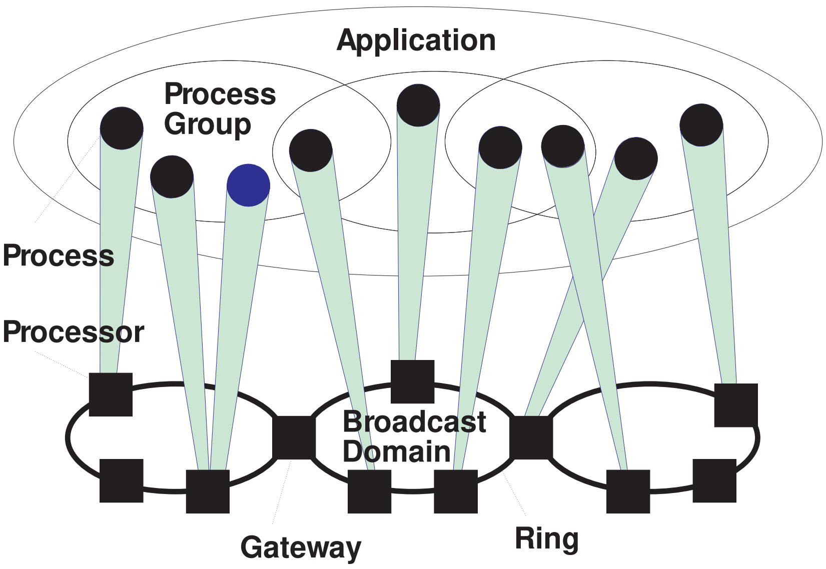

Figure 5: The relationship between rings and process groups in the Totem system.

For each application process, the interface establishes a socket through which the process communicates with Totem and which the process can poll to determine whether messages are pending.

As the process group interface passes messages from the application processes down to the multiple-ring protocol, it fragments large messages and combines small messages into larger messages (packets) of a convenient size for transmission. On receiving messages from the multiple-ring protocol, the process group interface reassembles the messages and enqueues them on the sockets of the processes that are members of the groups to which the messages are addressed. The process group interface receives messages in the correct order and, thus, does not need to be concerned with message ordering.

On each processor, the process group interface maintains the current membership of any process group of which at least one process on that processor is a member. When a process joins or leaves a group, this fact is disseminated throughout the network to all members of the group by the process group membership protocol.

Maintaining the consistency of message ordering when a process can be a member of several intersecting process groups, or when a process communicates with other processes outside its group, is an interesting problem. Ordering messages independently within each process group can lead to inconsistencies. Consider three processes, p, q and r, all of which are members of two process groups, G and H. Process p multicasts message m_1

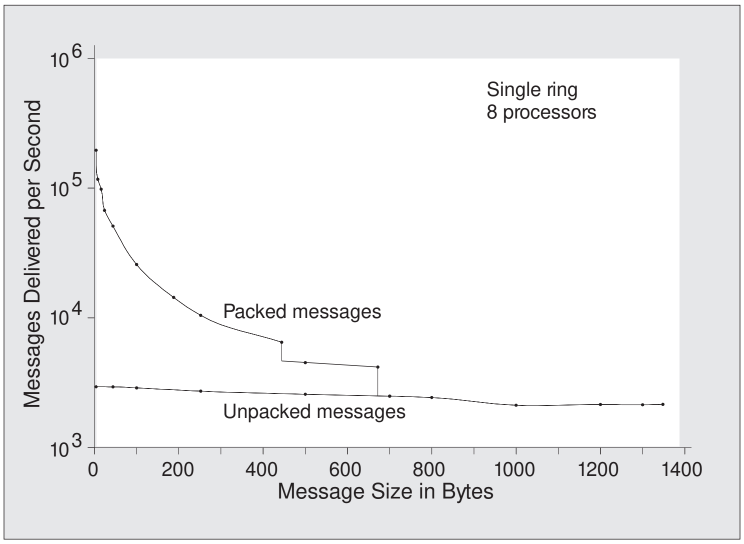

Figure 6: The throughput of the Totem single-ring protocol as a function of message size.

to group G. On receiving m_1, q multicasts message m_2 to group H. Clearly, m_1 causally precedes m_2 but, if messages are ordered only within groups, m_2 might be delivered to process r before m_1. The only effective method known to us for ensuring consistency in the presence of multiple intersecting process groups is to impose a single global total order on all messages for all process groups in the system, the strategy adopted by Totem.

Performance

The Totem system has been implemented in the C programming language on Sun Microsystems IPCs running SunOS 4.1, and on Sun SPARCstation 20s running Solaris 2.4, over 10 Mbit/s and 100 Mbit/s Ethernet. It uses the Ethernet hardware broadcast capabilities and standard UNIX facilities, in particular UNIX UDP sockets, to broadcast messages and to transfer the token. The implementation has been ported to several other types of machines.

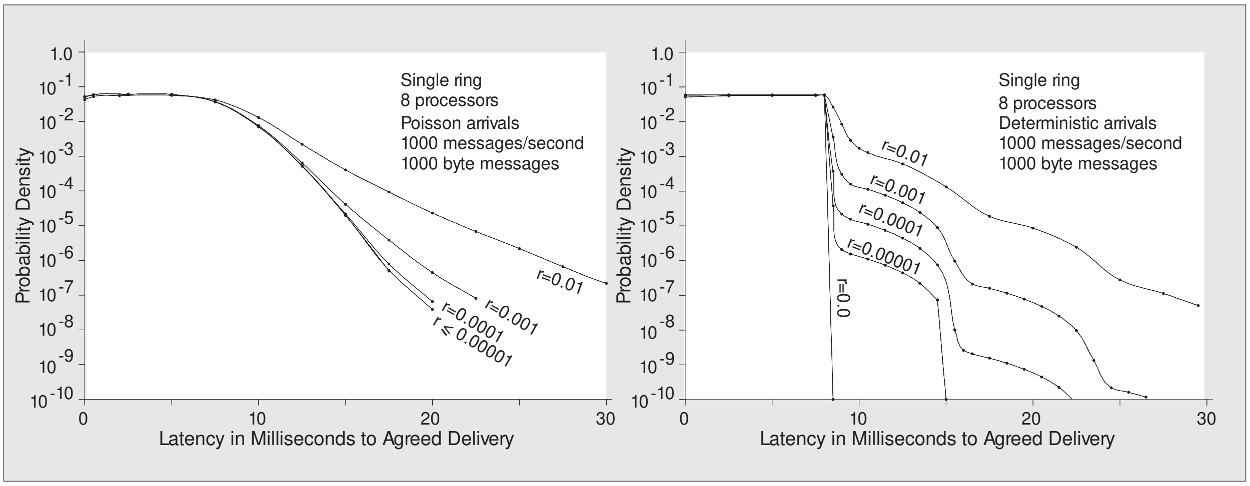

Figure 7: For the Totem single-ring protocol, the graph at the left shows the probability density function for the latency from message origination to agreed delivery for various values of r, the message loss probability, and for Poisson arrivals. The graph at the right shows the corresponding probability density function for deterministic arrivals.

Single Ring

We have measured the throughput (number of messages that an individual processor delivers into the total order per second) of the Totem single-ring protocol on our network of eight Sun SPARCstation 20s running Solaris 2.4 over a 100 Mbit/s Ethernet. Each processor was ready to broadcast at all times and the extraneous load on the processors and on the Ethernet was minimal. The flow control parameters were adjusted to maximize throughput.

As Figure 6 shows, the highest throughput results from packing small messages into larger messages (packets) within the application process. For small messages, the primary determinant of throughput is the cost of packing, rather than the cost of transmission and ordering. For the highest throughput, the processors are saturated and no cycles are left for the application. Realistic applications must operate with substantially fewer messages per second than are shown in Figure 6.

Detection of a processor crash requires at most 50 milliseconds and recovery after detection typically requires less than 20 milliseconds for this eight processor network. Further faults during recovery may lengthen this time, but it remains bounded.

For real-time applications, the latency from origination to delivery of a message is also important. To investigate the tail of the latency distribution, we have developed

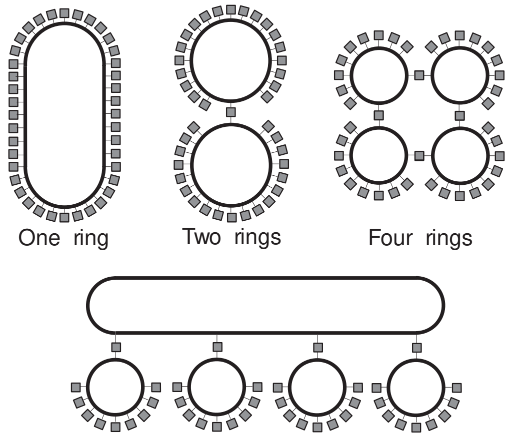

Figure 8: The four topologies used to calculate the latencies shown in Figure 9.

an analytic model [16]. The graph at the left of Figure 7 shows the probability density function for the latency to agreed delivery with approximately 1000 Poisson arrivals per second on a ring of eight processors with 1000 byte messages. The graph at the right of Figure 7 shows the corresponding probability density function for a deterministic arrival process. For a modern LAN, under normal conditions and with good flow control, the probability of message loss is very small, typically less than $r = 0.00001$.

As is evident, for low message loss rates, the deterministic arrival process has a lower probability of incurring a longer latency to agreed delivery than the Poisson arrival process. The Poisson arrival process allows messages to bunch together, slowing the token rotation and resulting in higher latencies for all messages in the bunch. If the nature of the system is such that the generation of messages tends to be Poisson rather than deterministic, it is necessary to operate at lower message generation rates to avoid high latencies.

We are currently extending the probability density function analysis to include processor faults and multiple rings. These probability density functions provide the predictability needed for real-time applications. (See the sidebar, "What is Real Time?")

Multiple Rings

In general, token-based protocols scale poorly to large systems, but Totem can operate on multiple rings with a filtering mechanism at each gateway. Messages addressed to a process group are forwarded along one (or more) spanning trees, but only if needed to

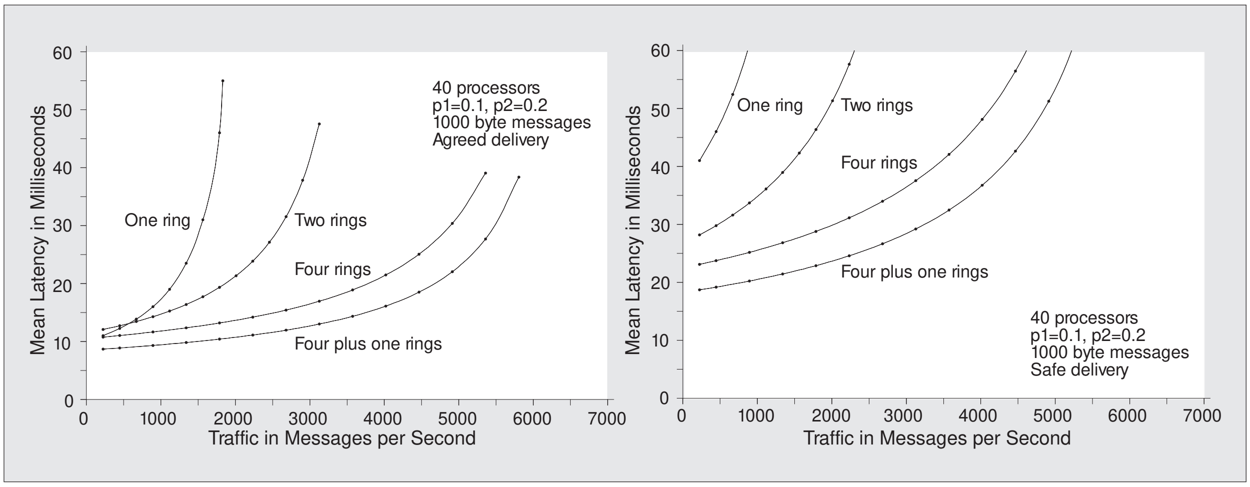

Figure 9: For the Totem multiple-ring protocol and for the four topologies shown in Figure 8, the graph at the left shows the mean latency to agreed delivery, while the graph at the right shows the mean latency to safe delivery, when relatively few messages must be forwarded to the other rings.

reach members of the process group. Thus, Totem exploits process group locality and scales logarithmically, rather than linearly, to larger networks.

To investigate the performance of the multiple-ring protocol with more processors than we currently have in our laboratory, we developed an analytic model. For the four topologies illustrated in Figure 8, each containing 40 processors, we considered the probability that a message must be forwarded through the network. In this analysis, p_1 represents the probability that a message originated in one half of the network must be forwarded to the other half; p_2 represents the probability that, within one half of the network, a message must be forwarded from one quarter to the other.

The graphs in Figure 9 show the mean latencies to agreed and safe delivery for various traffic levels (total number of messages generated in the network per second) and for each of the four topologies with $p_1 = 0.1$ and $p_2 = 0.2$. With these low probabilities, process group locality is good and relatively few messages must be forwarded through the gateways. The multiple-ring topologies show substantially lower latencies than a single ring with the same number of processors at the same traffic level, and are capable of sustaining substantially higher traffic levels with reasonable latencies. Even when $p_1 = 1.0$ and $p_2 = 1.0$, the multiple-ring topologies exhibit improved performance over a single ring, particularly for safe messages.

Conclusion

The Totem system enables fault-tolerant applications in distributed systems to maintain the consistency of replicated information by providing reliable totally ordered multicasting of messages. A hierarchy of protocols delivers messages to processes within process groups over a single LAN or over multiple LANs interconnected by gateways. The message ordering strategy of Totem employs timestamps to define a consistent total order on messages system-wide and sequence numbers to ensure reliable delivery of messages. Hardware broadcasts, multiple rings, filtering of messages, and process group locality enable Totem to achieve high throughput and low predictable latency.

Acknowledgments

We wish to thank Yair Amir, Thomas Archambault, Wesley Chun, Paul Ciarfella, Erling Fledsberg, Bei-Jing Guo, Michael King, Priya Narasimhan, Vandana Rao, Michael Santos and Efstratios Thomopoulos for their contributions to the development of the Totem system. We also wish to thank David Powell for inviting us to write this paper, as well as the other authors of the papers in this special issue for their comments.

This research was supported by the National Science Foundation, Grant Number NCR-9016361, and by the Advanced Research Project Agency, Contract Number N00174-93-K-0097.

More Information

The homepage for the Totem system is http://beta.ece.ucsb.edu/totem.html.

References

[1] Agarwal, D. A. *Totem: A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks.* PhD Dissertation. Department of Electrical and Computer Engineering. University of California, Santa Barbara (August 1994).

[2] Agarwal, D. A., Moser, L. E., Melliar-Smith, P. M., and Budhia, R. K. A reliable ordered delivery protocol for interconnected local-area networks. In *Proceedings of the International Conference on Network Protocols* (November 1995, Tokyo, Japan) 365–374.

[3] Amir, Y., Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., and Ciarfella, P. The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems* 13, 4 (November 1995) 311-342.

[4] Birman, K. P., and van Renesse, R. *Reliable Distributed Computing with the Isis Toolkit.* IEEE Computer Society Press, Los Alamitos, CA, 1994.

[5] Chandra, T. D., and Toueg, S. Unreliable failure detectors for reliable distributed systems. To appear in *Journal of the ACM.*

[6] Dolev, D., and Malki, D. The Transis approach to high availability cluster communication. *Communications of the ACM*, this issue.

[7] Fischer, M. J., Lynch, N. A., and Paterson, M. S. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32*, 2 (April 1985) 374–382.

[8] Kaashoek, M. F., and Tanenbaum, A. S. Group communication in the Amoeba distributed operating system. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems* (May 1991, Arlington, TX) 222–230.

[9] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., and Zainlinger, R. Distributed fault-tolerant real-time systems: The Mars approach. *IEEE Micro* 9, 1 (February 1989) 25–40.

[10] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (July 1978) 558–565.

[11] Lingley-Papadopoulos, C. A. *The Totem Process Group Membership and Interface.* MS Thesis. Department of Electrical and Computer Engineering. University of California, Santa Barbara (August 1994).

[12] Melliar-Smith, P. M., Moser, L. E., and Agrawala, V. Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems 1*, 1 (January 1990) 17–25.

[13] Melliar-Smith, P. M., Moser, L. E., and Agarwal, D. A. Ring-based ordering protocols. In *Proceedings of the IEE International Conference on Information Engineering* (December 1991, Singapore) 882–891.

[14] Moser, L. E., Amir, Y., Melliar-Smith, P. M., and Agarwal, D. A. Extended virtual synchrony. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems* (June 1994, Poznan, Poland) 56–65.

[15] Moser, L. E., Melliar-Smith, P. M., and Agrawala, V. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems 5*, 5 (May 1994) 459–473.

[16] Moser, L. E. and Melliar-Smith, P. M. Probabilistic bounds on message delivery for the Totem single-ring protocol. In *Proceedings of the 15th IEEE Real-Time Systems Symposium* (December 1994, San Juan, Puerto Rico) 238–248.

[17] Powell, D., ed. *Delta-4: A Generic Architecture for Dependable Distributed Computing* (1991) Springer-Verlag, Berlin, New York.

[18] van Renesse, R., Birman, K. P., and Maffeis, S. Horus: A flexible group communications system. *Communications of the ACM*, this issue.

About the Authors:

Louise E. Moser is an associate professor in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. **Author's Present Address:** Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106; email: moser@ece.ucsb.edu

P. M. Melliar-Smith is a professor in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. **Author's Present Address:** Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106; email: pmms@ece.ucsb.edu

Deborah A. Agarwal is a staff scientist at the Ernest Orlando Lawrence Berkeley National Laboratory. **Author's Present Address:** Ernest Orlando Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, MS 50B-2239, Berkeley, CA 94720; email: DAAgarwal@lbl.gov

Ravi K. Budhia is a candidate for the Ph.D. in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. **Author's Present Address:** Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106; email: ravi@alpha.ece.ucsb.edu

Colleen A. Lingley-Papadopoulos is a software engineer at Tandem Computers. **Author's Present Address:** Tandem Computers Incorporated, 19333 Vallco Parkway, LOC 3-22, Cupertino, CA 95014-2599; email: LINGLEY_AMY@tandem.com

Why Totem?

Totem – a North American Indian word for a small natural object serving as the emblem of a family or group, and passed from one generation to the next, just as the code of the Totem system has passed between successive generations of students at UCSB. The name Totem derives from the **tot**al order and **tem**poral predictability of the Totem system, while its hierarchical structure resembles a totem pole.

The best totem of all is the spirit and enthusiasm of the students.

Virtual Synchrony and Extended Virtual Synchrony

In group communication systems, the delivery and subsequent processing of multicast messages can alter related or replicated data items, maintained by several processes. If the messages are received in different orders by different group members, the data at those processes might become inconsistent. Moreover, if processes fail, or if they leave or join the process groups dynamically, different processes can have different views of the process group membership, which again might result in inconsistent data.

The virtual synchrony model [4], introduced for Isis, orders group membership changes along with the regular messages. It ensures that failures do not result in incomplete delivery of multicast messages or holes in the causal delivery order. It also ensures that, if two processes proceed together from one view of the group membership to the next, then they deliver the same messages in the first view. Virtual synchrony does not constrain the behavior of faulty or isolated processes. Faulty processes, if they recover, are regarded as new processes. In a primary partition strategy such as that of Isis, if the system partitions, one component of the partition (the primary component) continues to operate, if any. Processes in other components are deemed to be faulty.

Processors and processes do, however, recover after failure with stable storage intact, and networks do remerge after partitioning. In different components of a partitioned network, processes can operate concurrently without being able to communicate with each other. Thus, the message delivery guarantees provided by a process can refer only to its local component, which suffices for messages delivered only in that component. However, while the network is becoming partitioned or while a process fails, some messages might be delivered in more than one component of the network.

The extended virtual synchrony model [14], introduced for Totem, extends the model of virtual synchrony to systems in which processes can fail and recover and in which the network can partition and remerge. Even in such systems, the message delivery guarantees are strictly honored. The same messages may be delivered in two or more components of a partitioned network, but the message ordering is consistent in all of them. Moreover, some processes may not have received a message and, thus, the other processes are told which processes are known to have received it. Extended virtual synchrony does not solve all the problems of recovery in a fault-tolerant distributed system, but it does avoid inconsistencies that make recovery unnecessarily difficult.

What is Real Time?

Traditionally, the design of real-time systems has been dominated by the synchronous hard real-time paradigm, which is appropriate for embedded real-time systems [9]. All operations in the system are performed according to a preplanned schedule based on the worst-case workload and worst-case processor performance. The classical hard real-time paradigm aims to provide absolute guarantees that every real-time deadline will be met.

In the real world, however, there are no absolute guarantees; there is a probability, small but non-zero, that all of the processors within the system will fail. The best that we can do is to determine the probability that a deadline will be missed and ensure that the probability is small enough. For civilian airline flight control, a probability of 10^{-16} of missing a deadline is small enough, based on an acceptable failure rate of 10^{-10} per hour and 10^6 deadlines per hour. Other applications allow higher rates of missing deadlines.

Complex real-time systems contain many sources of variability. There are variations in processor performance resulting from caches, cycle stealing and interrupt handling; there are variations in the execution of programs caused by special cases and by heuristic algorithms; there are variations resulting from fault recovery; and there are variations in the workload. A preplanned worst-case design for a complex real-time system is necessarily a conservative design with adverse effects on performance. Such systems are, therefore, seldom built as preplanned synchronous systems. Instead, they are built as event-driven asynchronous soft real-time systems that provide a high probability, rather than an absolute guarantee, that real-time deadlines will be met.

Both hard and soft real-time paradigms are necessarily probabilistic. For soft real-time systems, we estimate the probability that the system will generate the required results before the deadlines. For hard real-time systems, we estimate the probability that the system will generate the results at all. These probabilities are not easy to calculate; more research is required in this area.

In simple real-time systems, only one or a few operations are pending at any time, and processing and communication latencies are important factors in ensuring that real-time deadlines are met. In complex real-time systems, many operations may be pending and the time that these operations spend in queues is an important factor in determining whether real-time deadlines are met. The lengths of the queues and the time the operations spend in them are determined by the throughput. Consequently, complex real-time systems are better served by mechanisms designed for high throughput and predictable latency, rather than by mechanisms that try to achieve the lowest possible latency at the price of decreased throughput.