

# A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks

D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith, R. K. Budhia

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

## Abstract

We present the Totem multiple-ring protocol, a novel reliable ordered multicast protocol for multiple interconnected local-area networks. The protocol exhibits excellent performance and maintains a consistent network-wide total order of messages despite network partitioning and remerging, or processor failure and recovery with stable storage intact. The Totem protocol is designed for fault-tolerant distributed systems, which replicate data to guard against failures and must ensure that replicated data remain consistent despite failures. The network-wide total order of messages provided by Totem simplifies the maintenance of consistency of replicated data and, thus, eases the development of fault-tolerant distributed systems.

## 1 Introduction

The Totem protocol has been designed to support fault-tolerant distributed systems. In such systems, both the processes executing application tasks and the data must be replicated to protect against failures. Inconsistencies in the replicated data can arise if processes receive and process the same messages in different orders. Prevention of such inconsistencies can be simplified by a reliable ordered multicast protocol that delivers messages in the same total order to all processes in the system. With a multicast protocol that maintains a consistent total order of messages, programming the application is easier than with a causally ordered or unordered multicast protocol.

The design of an efficient totally ordered multicast protocol is, however, difficult if the protocol must maintain a consistent global total order of messages when the network partitions and remerges or when a processor fails and recovers with stable storage intact. In a failing processor, or in an isolated component of the network, a naive ordered multicast protocol may deliver messages in an order that is different from the order determined elsewhere in the network. These inconsistencies may not be apparent immediately but, when a processor is repaired and readmitted to the system or when the partitioned network is remerged,

the inconsistencies may become manifest and recovery may be difficult.

It is also difficult for an ordered multicast protocol to maintain a consistent message order when messages are multicast only to members of particular process groups. Provided that the groups do not overlap and communication is restricted to the members in a group, a protocol that orders messages independently in each of the groups will not cause inconsistencies. However, if the process groups overlap or a process communicates with processes outside its group, then inconsistencies in the message order can arise.

These problems can be solved by an ordered multicast protocol that places a consistent network-wide total order on messages, provided that it has good performance. The poor performance of previous ordered multicast protocols has caused system designers to use protocols for partially ordered delivery of messages, or protocols for totally ordered delivery of messages within a process group or within a network component. The risk of inconsistent message ordering, which is the price of such compromises, can be eliminated by a network-wide total ordering protocol that achieves high efficiency and low overheads.

The Totem protocol maintains consistency of message ordering by imposing a total order on messages network-wide. The protocol is currently being elaborated to achieve high throughput and low latency by avoiding the need for all messages to be transmitted to, and ordered by, all processes in the network. Messages destined for processes in a particular process group are forwarded only to the parts of the network containing members of that group; nonetheless, they are delivered in a consistent total order network-wide. When the system is structured so that the process groups exhibit a high degree of locality, few messages need to be forwarded across the entire network and excellent performance can be achieved while strict consistency is maintained.

The Totem protocol hierarchy is shown in Figure 1. The bottom layer of the hierarchy is a best-effort multicast service, typically provided by the Unix UDP protocol using the physical broadcast capability of the Ethernet. The Totem single-ring protocol converts the best-effort multicast service into a service that provides reliable totally ordered delivery of messages within a single local-area network, as well as failure

---

This work was supported by NSF grant NCR-9016361 and by ARPA grant N00174-93-K-0097. The current address of D. A. Agarwal is Lawrence Berkeley National Laboratory, Berkeley, CA 94720.

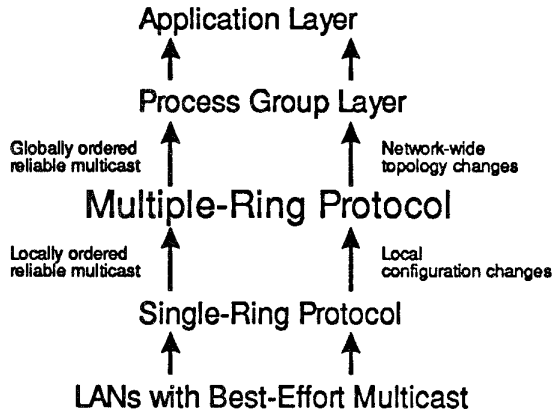


Figure 1: The Totem protocol hierarchy.

detection, recovery, and membership services. The Totem multiple-ring protocol provides similar services for larger networks composed of multiple local-area networks interconnected by gateways. The process group layer, above the Totem multiple-ring protocol layer, delivers messages to those application processes that are members of the appropriate process group, and maintains the process group membership.

In this paper we do not discuss the Totem single-ring protocol or the physical layer below it; descriptions of the single-ring protocol can be found in [1, 3, 10]. Rather, we provide a high-level description of the Totem multiple-ring protocol. A detailed description of the multiple-ring protocol, including proofs of correctness and pseudo-code for the implemented protocol, can be found in [1]. The Totem multiple-ring protocol is the first protocol of which we are aware to combine sequence numbers and timestamps to provide a global total order of messages across multiple interconnected local-area networks in the presence of network partitioning and remerging, and processor failure and recovery. While the basic idea of the Totem protocol is novel, elegantly simple, and efficient, the implementation proved surprisingly difficult, particularly in maintaining a consistent message order through topology changes and in maintaining a consistent network topology.

## 2 Background

Several protocols have been developed to provide reliable ordered delivery of messages. Unfortunately, most of these protocols do not scale well to large networks. Protocols that provide efficient reliable ordered delivery across multiple interconnected local-area networks are needed.

The Isis system [4, 5] has been used in a variety of applications to provide partial and total ordering of messages within process groups. Isis uses the same reliable ordered delivery protocol regardless of the size of the network. Isis includes timestamp vectors in messages multicast within a group to preserve causality

relationships and uses a protocol similar to that of Chang and Maxemchuk [7] to determine a total order of messages within the group. The Isis protocols were designed for a system that does not allow remerging of a partitioned network or rejoining of processors that recover with stable storage intact. The protocols of Horus [16], the successor of Isis, have better performance and incorporate our notion of extended virtual synchrony [12] to allow for recovery from such failures.

The Trans and Total protocols [11, 13] provide partial and total ordering of messages broadcast on a local-area network. The Trans protocol uses positive and negative acknowledgments that are piggybacked on messages to create a partial order on messages. The Total protocol converts this partial order on messages into a total order without exchange of additional messages. The Lansis and Toto protocols of the Transis system [2] take a similar approach. The Lansis protocol is a variation of the Trans protocol that temporarily withholds acknowledgments pending the delivery of messages. The Toto protocol differs from the Total protocol in that it uses the exchange of additional messages to provide deterministic rather than probabilistic guarantees of message delivery.

In the Amoeba system [8], messages are sent point-to-point to a central coordinator that assigns the message a sequence number and then broadcasts the message. A process acknowledges receipt of messages by placing the highest message sequence number received without gaps in its next message. Messages are broadcast within process groups, and each group has its own independent central site. Messages for different groups are not ordered with respect to one another.

Takizawa [14, 15] has also investigated ordered group communication protocols on a single high-speed channel and over larger networks consisting of groups of processors connected by gateways. Preacknowledgments are used to ensure that buffers do not overflow and to define a total order on messages. The RMP protocol [17] also provides reliable ordered multicasts over larger networks, but lacks the performance of UDP over a single local-area network.

The Totem single-ring protocol [1, 3, 10] was designed to achieve lower computational cost and higher throughput than the Trans and Total protocols. It provides totally ordered message delivery and membership services using a logical token-passing ring imposed on a local-area network with broadcast communication. Consistency of message ordering is maintained despite network partitioning and remerging, or processor failure and recovery with stable storage intact. The Totem multiple-ring protocol described here is built on top of the Totem single-ring protocol.

## 3 Model

We model a network as multiple local-area networks interconnected by gateways. Each gateway connects a pair of local-area networks, but a pair of local-area networks is connected directly by at most one gateway. The gateways forward messages between local-

area networks. Like any other processor, a gateway can transmit messages from, and deliver messages to, the application processes.

Processors and gateways may incur fail stop, timing, or omission faults. There are no malicious faults. Failure of a processor or gateway is represented in our model as a partition that isolates the failed processor or gateway from the rest of the network. Messages may be lost or corrupted on the local-area network, but a local-area network protocol is assumed to handle this and deliver each message or detect a failure.

The network may become partitioned so that processors in one component are unable to communicate with processors in another component. Partitioned components of a local-area network are assumed to operate as separate local-area networks while the partition exists. Communication among separated components may subsequently be reestablished.

A *configuration* or *ring* is a set of processors on a local-area network that are able to communicate with each other by broadcasting messages. A *topology* is a set of *rings* interconnected by gateways such that there is a path between any two rings in the topology. A *process group* is a set of processes, possibly on processors on different rings. A process may be a member of several process groups, and a ring may contain processes from different process groups.

## 4 Services

The services provided by the Totem multiple-ring protocol are described below in terms of topologies. A local-area network protocol, such as the Totem single-ring protocol, is assumed to provide similar services for configurations in each local-area network. Proofs of correctness that the multiple-ring protocol satisfies these properties are given in [1].

### 4.1 Topology Maintenance Services

The Totem multiple-ring protocol provides membership and topology services across multiple interconnected local-area networks using the services provided by the Totem single-ring protocol. Each change in the membership within a local-area network due to processor failure or recovery, or network partitioning or remerging, is reported by a Configuration Change message generated by the single-ring protocol. Topology Change messages are generated by the multiple-ring protocol as a result of a configuration change. Topology Change messages and Configuration Change messages are delivered to the process group layer, which forwards information about process group membership to the application; such information is necessary to enable the application to take an appropriate course of action. The multiple-ring protocol provides the following topology maintenance properties.

#### Uniqueness of Topologies

Each topology identifier is unique; moreover, at any time a processor is a member of a ring in at most one topology.

#### Consensus

If two processors are members of rings in the same topology, if neither receives a Configuration Change message that disconnects it from the other, and if one of the processors installs a new topology with a given set of rings, then the other processor decides on the same set of rings for its next topology.

#### Termination

If a topology ceases to exist for any reason, such as processor failure or network partitioning, then every processor in that topology will install a new topology or will fail before doing so. The minimum topology is a ring that contains only a single processor.

#### Topology Change Consistency

Processors that are members of the same topology deliver the same Topology Change message to initiate the topology. Moreover, if two processors install a topology  $T_2$  directly after  $T_1$ , then the processors deliver the same Topology Change message to terminate  $T_1$  and initiate  $T_2$ .

## 4.2 Reliable Ordered Delivery

Reliable totally ordered message delivery is provided by the Totem single-ring protocol for messages on a local-area network and by the Totem multiple-ring protocol across multiple local-area networks. We distinguish between receipt and delivery of a message as follows. A message is *received* from the next lower layer in the protocol hierarchy, and a message is *delivered* to the next higher layer.

Messages are delivered within a topology, and processors that deliver messages in the same topology deliver those messages in the same total order. The total order on messages originated within a topology respects the causal order defined by Lamport [9]. Partitioning of the network can result in different sets of messages being delivered in different components of the network and, therefore, in different topologies.

Regular messages are originated by the application for delivery to the application, and are ordered by timestamp and source ring identifier. Topology Change messages are ordered with respect to regular messages as follows. The Topology Change message that initiates a topology is delivered prior to the first regular message delivered in that topology. The Topology Change message, delivered to terminate a topology, is delivered after all of the regular messages delivered in that topology. The Totem multiple-ring protocol provides the following reliable ordered delivery properties.

#### Reliable Delivery

Reliable delivery requires that each regular message has a unique timestamp and source ring identifier (and thus a unique identifier) and that each message is delivered once only. In addition, a processor must deliver its own messages, as well as all of the other messages

originated in its current topology, unless a topology change occurs. Moreover, if two processors are both members of consecutive topologies, then those processors must deliver the same set of messages originated in the first topology.

#### Delivery in Agreed Order

Delivery in agreed order for topology  $T$  requires reliable delivery, delivery in causal order, and delivery in the total order for topology  $T$ . In particular, if processor  $p$  delivers a message in topology  $T$ , then  $p$  must have delivered all messages that precede that message in the total order for topology  $T$ .

#### Delivery in Safe Order

Delivery in safe order for topology  $T$  requires delivery in agreed order for topology  $T$  and also that, when processor  $p$  delivers message  $m$  in topology  $T$  and the originator of  $m$  requested safe delivery, then  $p$  has determined that each processor in  $T$  has received  $m$ , and will deliver  $m$  in safe order or will fail before doing so.

Safe delivery is useful because an application may need to know that all processes have received the message. Moreover, it allows a processor to reclaim the buffer space used by the message. The processor then knows that the message has been received by the other processors and will not need to be retransmitted. However, delivery of messages in safe order incurs additional latency.

#### Extended Virtual Synchrony

Extended virtual synchrony [12] extends the concept of virtual synchrony introduced in [5]. Virtual synchrony ensures that processors that are members of the same consecutive configurations deliver the same sequence of messages and configuration changes, but does not constrain the behavior of failed or isolated processors. Extended virtual synchrony requires, in addition, that the total order for each topology is a subset of a global total order on all messages generated in the system. It extends the concept of virtual synchrony to systems in which the network can partition and remerge and to systems in which failed processors can be repaired and recover with stable storage intact.

Extended virtual synchrony allows two processors in different components of a partitioned network to deliver different sets of messages, but it does not allow them to deliver messages in inconsistent orders. In particular, if processor  $p$  delivers message  $m_1$  before  $p$  delivers message  $m_2$ , then processor  $q$  must not deliver message  $m_2$  before  $q$  delivers message  $m_1$ . Extended virtual synchrony also requires the properties of delivery in agreed and safe order to be satisfied. Thus, if processor  $p$  delivers message  $m$  as safe in topology  $T$ , then every processor in  $T$  must have received  $m$ , and must deliver  $m$  unless it fails. In the presence of failed processors or a partitioned network, safe delivery is achieved by introducing a new topology with a reduced membership, all members of which are able to honor the safe delivery guarantee.

## 5 Reliable Ordered Delivery

We first describe the Totem multiple-ring protocol without considering configuration changes that may occur during its execution. The difficult task of dealing with configuration changes is considered in Section 6.

The basic strategy of the Totem multiple-ring protocol is to deliver messages and topology changes in timestamp order. Timestamp order is very attractive, as it guarantees global consistency of message ordering. The ability of a processor to deliver a message in timestamp order depends, however, on that processor's knowing that it has already received and delivered all relevant messages whose timestamps are less than the timestamp of the message to be delivered. This knowledge is not readily obtained in a network subject to message loss, and we are aware of no other implemented reliable ordered multicast protocol that is based on delivery of messages in timestamp order.

The Totem multiple-ring protocol exploits the Totem single-ring protocol for this knowledge. The single-ring protocol provides reliable delivery of messages in sequence number order. On any one ring, messages are generated with increasing sequence numbers and timestamps. A gateway forwards messages, when necessary, in sequence number order. When a gateway broadcasts a forwarded message on its other ring, it provides the message with a new sequence number so that the message can be reliably delivered in sequence number order on that ring. The timestamp of a forwarded message remains unchanged to ensure that the message is delivered in the same total order on all rings. Both sequence numbers and timestamps have been used by previous protocols, for example [6], but not in conjunction to achieve total ordering of messages in multiple interconnected local-area networks.

The Totem single-ring protocol delivers a message that was originated on a remote ring only if it has already delivered all relevant messages originated on that ring with smaller sequence numbers. The Totem multiple-ring protocol delivers a message only if the following two conditions are satisfied: (1) the message has the lowest timestamp of any undelivered message, and (2) the processor or gateway executing the multiple-ring protocol has received at least one message with a higher timestamp/source ring identifier originated on each of the other rings.

### 5.1 Data Structures and Messages

The Totem multiple-ring protocol employs the following data structures and messages. The data structures and messages used by the Totem single-ring protocol are not enumerated here, but are described in [1].

#### Data Structures

Each processor or gateway maintains the following data structures to track the messages received and to enable message ordering by the multiple-ring protocol.

- *my\_timestamp*: The highest message timestamp known to this processor.
- *ring\_table*: An array with an entry for each ring in the topology containing
  - *ring\_id*: A unique ring identifier.
  - *recv\_msgs*: A list of messages that were originated by a processor on the ring and that have not yet been delivered to the process group layer.
  - *max\_timestamp*: The highest timestamp of a message in *recv\_msgs* for the ring.
  - *min\_timestamp*: The lowest timestamp of a message in *recv\_msgs* for the ring.
- *can\_msgs*: A list containing the lowest entry in *recv\_msgs* for each ring, sorted by timestamp.
- *my\_guarantee\_vector*: A vector whose length is the number of rings in the topology. Each vector component contains the highest timestamp of any message received from the corresponding ring. A gateway maintains such a vector for each of the two directly attached rings.
- *guarantee\_array*: An array with one row for each ring in the topology. The rows are the guarantee vectors received from the gateways on the other rings and the local guarantee vector. Each column of the array corresponds to messages originated on a particular ring.

The Totem multiple-ring protocol handles the following types of messages, as well as the membership and topology change messages described in Section 6.

#### Regular Message

Each regular message contains data, as well as the following fields used by the Totem multiple-ring protocol:

- *timestamp*: The message timestamp.
- *src\_ring\_id*: The identifier of the ring on which the message was originated.
- *src\_sender\_id*: The identifier of the processor that originated the message.

The first two fields constitute the identifier of the message, and are used for message ordering. If two messages have the same timestamp, then the *src\_ring\_ids* are used to order the messages. The uniqueness of the *src\_ring\_ids* through processor failure and recovery, or network partitioning and remerging, is maintained by the use of stable storage on disk.

#### Guarantee Vector Message

Each Guarantee Vector message originated by a gateway contains

- *guarantee\_vector*: The current *my\_guarantee\_vector* for the ring on which this message was originated.

Guarantee Vector messages are forwarded throughout the network, but are not delivered to the application processes.

## 5.2 The Ordering Protocol

A processor or gateway executing the Totem single-ring protocol stores new messages received from the application in a FIFO buffer until it can broadcast them on the local-area network. A gateway also places messages that were forwarded from the other ring attached to the gateway into this same buffer.

When a processor or gateway receives a message or the token, it sets its local variable *my\_timestamp* to the larger of *my\_timestamp* and the *timestamp* field in the message or token. For each new message it broadcasts, a processor increments *my\_timestamp* and sets the *timestamp* field in the message to *my\_timestamp*; this is done when it removes the message from the buffer and before it broadcasts the message on the ring. Before releasing the token, a processor sets the *timestamp* field in the token to *my\_timestamp*. A forwarded or retransmitted message retains the timestamp and source ring identifier it was given when it was originated. Each new message broadcast by a processor has a higher timestamp than the timestamp of any message previously received or broadcast and, thus, the timestamps are Lamport timestamps [9].

A processor or gateway writes its *my\_timestamp* to stable storage periodically. When a membership change occurs (which happens very infrequently), a processor or gateway writes its *my\_ring\_id* (its local ring identifier) to stable storage. Our implementation aims to minimize the cost of writing to stable storage. When a processor or gateway recovers, it initializes its *my\_timestamp* and *my\_ring\_id* to values that are greater than the timestamp and ring identifier that it last used as indicated by the stable storage. This ensures that causality and total ordering are maintained through recovery of failed processors and remerging of partitioned networks.

#### Reliable Delivery

The primary mechanism used to achieve reliable delivery of messages is the single sequence of message sequence numbers provided by the Totem single-ring protocol. The single-ring protocol uses a token circulating around a logical ring superimposed on the local-area network. A processor (or gateway) must hold the token to broadcast a message. A sequence number field, *seq*, in the token provides total ordering of messages for all processors on the ring. When a processor receives the token, it checks that it has received all messages with sequence numbers less than *seq*. If not, it requests that a missing message be retransmitted by including the sequence number of the message in a retransmission request field, *rtr*, of the token. The message is retransmitted by the first processor around the ring that follows the processor requesting retransmission and that has the message. An all-received-up-to field, *aru*, of the token allows a processor to determine whether all processors on the ring have received all messages with sequence numbers less than or equal to the *aru*.

Messages broadcast on a ring are delivered in sequence number order by the Totem single-ring protocol to the Totem multiple-ring protocol executing at a gateway. The gateway forwards the messages required by processors on the other side of the network in sequence number order. In doing so, it broadcasts the messages on its other ring, giving them a new sequence number but not a new timestamp. The forwarded messages are interleaved with messages originated on the other ring and with messages forwarded onto that ring by the other gateways.

### Delivery of Messages in Agreed Order

The key to delivery of messages in agreed order by the Totem multiple-ring protocol is that messages originated on a ring are forwarded from one ring to another in sequence number order.

For each of the rings in its *ring.table*, each processor or gateway maintains a *max.timestamp*. The *max.timestamp* records the highest timestamp of any message received by the processor or gateway and originated on that ring. All messages from that ring with lower timestamps have already been received, since the messages are forwarded in sequence number order. If a gateway receives a message with a timestamp less than the *max.timestamp* of the message's source ring, it discards the message. This mechanism allows a gateway to identify redundant copies of messages forwarded over different routes.

Messages are delivered to the application processes by the Totem multiple-ring protocol in the order of their timestamps. Messages with the same timestamp are delivered in the order of their source ring identifiers. To deliver messages in agreed order, a processor first determines the lowest entry in *cand.msqs*. If the lowest entry in *cand.msqs* is a Guarantee Vector message, it is discarded. Otherwise, if the lowest entry message in *cand.msqs* has requested agreed delivery and the *recv.msqs* lists for all of the rings are non-empty, then the message is delivered into the total order. The *recv.msqs* lists must be non-empty because, if the *recv.msqs* list for some ring were empty, then the next message from that ring might have a lower timestamp than the messages already received from the other rings.

A gateway periodically generates Guarantee Vector messages for the rings to which it is attached, typically once per token rotation. The Guarantee Vector messages have a dual purpose: (1) they ensure that a processor is not delayed for long by the lack of a message in the *recv.msqs* list of some ring, and (2) they inform a processor of the messages that have been delivered as agreed or safe on the local ring by the single-ring protocol. This reduces the latency of the protocol and allows delivery of messages in safe order network-wide, as described below.

### Delivery of Messages in Safe Order

Delivery of a message in safe order by the Totem multiple-ring protocol requires information about whether the message has been received by all of the

relevant processors in the topology. When the single-ring protocol delivers a message in safe order, all of the processors and gateways on the local ring have already received the message.

A processor executing the multiple-ring protocol maintains a variable, called *my.guarantee.vector*, in which it records the messages that have been delivered on its local ring. A gateway maintains two such vectors, one for each of the rings to which it is attached; it periodically transmits a Guarantee Vector message containing a guarantee vector through the network. The guarantee vectors form the rows of the *guarantee.array* and ensure that a processor or gateway has received all of the messages from the other rings in the topology up to a particular timestamp.

When a processor or gateway receives a Guarantee Vector message, it compares the guarantee vector in the message with the corresponding row of its *guarantee.array*. If any component of the new guarantee vector is greater than the corresponding component of the row, it replaces the row with the new guarantee vector.

A message can be delivered in safe order only if the following condition is met. All entries in the column of the local *guarantee.array* for the message's source ring must be at least equal to the timestamp of the message. This ensures that all of the processors and gateways have received the message. The reliable delivery property of the single-ring protocol, the forwarding of messages by the gateways, the periodic broadcasting of Guarantee Vector messages, and the mechanisms of the membership and topology maintenance protocol ensure that a processor or gateway will not wait forever to deliver the message in safe order.

### 5.3 Example

As an example, consider the network shown in Figure 2, where the rings are represented by circles and the processors and gateways by squares. Processor *p* has already ordered all messages up through the message with timestamp 17, and the *recv.msqs* lists for rings *A*, *B* and *C* are as shown. If the message with timestamp 19 from ring *C* contains a request for agreed delivery, then processor *p* orders that message next.

If, however, the message with timestamp 19 from ring *C* contains a request for safe delivery, then processor *p* cannot order that message because some of the entries in the column of the *guarantee.array* for ring *C* are less than 19. Processor *p* does not yet know that the message has been received by all of the processors in the network.

Once the message with timestamp 19 from ring *C* is ordered, processor *p* cannot order any other message, in particular the message with timestamp 21 from ring *B*, until it receives the next message from ring *C*. Otherwise, there may be a message from ring *C* with timestamp 20 that it has not yet received.

## 6 Membership and Topology Changes

The message ordering algorithm described above depends on knowledge of the network topology. If mes-

Data structures at processor p

A *recv\_msgs* = 25, 28, 30  
*min\_timestamp* = 25  
*max\_timestamp* = 30

B *recv\_msgs* = 21, 24  
*min\_timestamp* = 21  
*max\_timestamp* = 24

C *recv\_msgs* = 19  
*min\_timestamp* = 19  
*max\_timestamp* = 19

	A	B	C
A	30	24	19
B	7	14	8
C	10	12	17

guarantee\_array at p

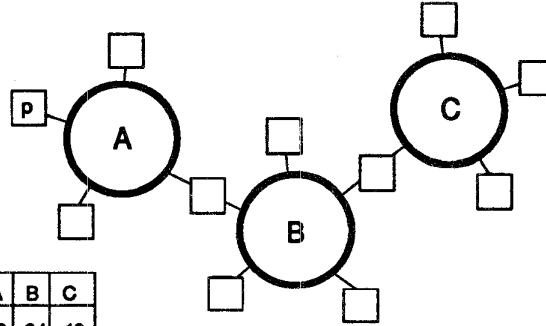


Figure 2: An example with rings A, B and C indicated by circles. The processors and gateways are drawn as squares. The data structures at processor p are also shown. A row of the *guarantee\_array* corresponds to the guarantee vector received from a gateway on the ring.

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

Each gateway maintains a data structure in which to record its current view of the network topology, a graph with rings represented as nodes and gateways as edges. Gateways use this data structure for several purposes, including to decide which messages should be forwarded. In the event of a topology change, the processors receive the necessary topology information from the gateways. The application receives information about the changes in the memberships of process groups from the process group layer.

Processor failure and network partitioning, and the addition of a new processor or gateway, are detected by the single-ring protocol, which generates a Configuration Change message to report the change to the multiple-ring protocol. The gateways analyze the Configuration Change message to determine its effect on the network topology. When a ring becomes inaccessible, a processor or gateway generates a Topology Change message and removes that ring from its *ring\_table*, ending the need to wait for messages from that ring which will never arrive and allowing messages from other rings to be ordered.

It is important that a topology change should have the same effect throughout the set of processors that were previously able to, and can still, communicate with one another. Even though the processors learn of the topology change at different physical times, all should agree on a common logical time for the topology change and on the sets of messages delivered before and after the topology change. Consequently, all Configuration Change and Topology Change messages are timestamped and are delivered to the process group

layer, along with other messages, in timestamp order. Only with great care is it possible to maintain, network-wide, the extended virtual synchrony guarantees of Section 4.

## 6.1 Topology Maintenance Messages

The Configuration Change and Topology Change messages maintained by the Totem multiple-ring membership and topology maintenance protocol are described below. The difference between these two types of messages is that a Configuration Change message signals a change to a new ring within the local-area network, whereas a Topology Change message deletes rings from, and adds rings to, the topology of the network as a whole.

### Configuration Change Message

A Configuration Change message is generated by the single-ring protocol to report a change in the membership of a ring for any reason, such as processor failure or recovery, or network partitioning or remerging. It informs processors and gateways of the membership of a new ring, and marks the beginning of the delivery of messages for the new ring. The Configuration Change message is forwarded by the gateways and is delivered by both the single-ring and multiple-ring protocols after all messages delivered on the old ring. It is originated as an agreed message, and contains a timestamp, the source ring identifier of the configuration it initiates, and the membership of that configuration.

### Topology Change Message

A Topology Change message is generated by the multiple-ring protocol as a result of a configuration change generated by the single-ring protocol. Gateways forward Topology Change messages, which are delivered in order along with other messages by the

single-ring protocol. The local view of the topology is updated when the Topology Change message is delivered by the multiple-ring protocol. The Topology Change message is originated with a request for agreed delivery, and contains the identifiers of the rings and of the gateways, added to or deleted from the topology, if any.

## 6.2 Handling of Configuration Changes

We now describe the handling of a single configuration change without considering additional configuration changes during the operation of the membership and topology maintenance protocol. The handling of additional configuration changes is somewhat more complex, and is described in detail in [1], but all messages are still processed in timestamp order.

A topology change is signaled by the receipt of a Configuration Change message generated by the single-ring protocol. When a processor or gateway receives a Configuration Change message, it adds the new ring to the *ring\_table*, inserts the Configuration Change message into *recv\_msgs* for the new ring, and sets the corresponding *min\_timestamp* and *max\_timestamp* to the timestamp of the Configuration Change message.

### Merging Disconnected Components

When two or more components of a partitioned network are rejoined, the gateways in one component have no topology information about the other components, and cannot obtain it from the purely local information in the Configuration Change message. Thus, a gateway attached to the ring involved in the configuration change determines the topology of the newly connected rings by combining its own topology information with that received from the other gateways. When a gateway finds that a Configuration Change message for one of its attached rings is the lowest entry in *cand\_msgs*, it forwards the new topology information through the network in a Topology Change message.

A gateway or processor delays delivery of the Configuration Change message to the process group layer until it has received a Topology Change message with the same timestamp and source ring identifier. This is necessary because the Configuration Change message only provides information about the ring involved in the configuration change; information about the effect of the configuration change on the topology of the network as a whole is provided by the Topology Change message. A message with a timestamp earlier than that of the Configuration Change or Topology Change message and originated in one component of the network is not forwarded to other components of the network when they merge.

### Disconnection of Rings

If a configuration change causes a ring to become disconnected, no further messages will be received from that ring and message ordering will stop. We consider two cases. When a processor or gateway determines that it is unable to deliver messages because

the *recv\_msgs* list is empty for some ring, and it has received but not delivered a Configuration Change message deleting that ring, it generates a Topology Change message to delete the ring. The timestamp of the Topology Change message is the timestamp of the last message received from the ring and the Topology Change message is ordered immediately after that last message. This is necessary because there may have been a message generated on that ring before the ring became disconnected, which will never reach the processor due to the absence of a forwarding path. For consistency requirements, the Topology Change message must indicate that the ring was disconnected before the message was generated.

A processor or gateway may also determine that it is unable to deliver messages because the message with the lowest timestamp in *cand\_msgs* requested safe delivery, the Guarantee Vector message from some ring does not show that the message is safe on that ring, and a Configuration Change message indicates that the ring has become disconnected. Again, the processor must generate a Topology Change message to delete the ring but, in this case, the Topology Change message must be timestamped and delivered immediately before the message requesting safe delivery.

A gateway determines that a ring has become disconnected when it receives a Configuration Change message that deletes the last remaining connection to the ring. Making this determination is what adds interest to the handling of multiple concurrent Configuration Change messages.

## 7 Implementation and Performance

The Totem single-ring and multiple-ring protocols have been implemented using the C programming language on a network of Sun workstations connected by a 10 Mbit/s Ethernet. To test the Totem multiple-ring protocol within our limited physical resources, multiple rings were run on one Ethernet. Each such ring had its own socket numbers for broadcasting and for transmitting the token. Although this allows multiple rings to coexist on a single Ethernet, it reduces the effectiveness of the Totem single-ring flow control algorithm and increases the message loss rate.

The measurements of the Totem multiple-ring protocol presented here were made for two rings, operating over separate Ethernets, connected by a gateway. Each ring had two other processors. A Sun SPARCstation 20 was used as the gateway and four Sun 4/PCs were used as the processors. The gateway was allowed to transmit twice as many messages per token rotation as each of the individual processors.

The Totem single-ring protocol with three processors (two IPCs and one SPARCstation 20) on a ring, transmitting 1024 byte messages, achieved a throughput of 768 agreed messages per second. With the Totem multiple-ring protocol running on the ring, the throughput dropped to 636 ordered messages per second. This translates to an additional 0.27 milliseconds per message for the Totem multiple-ring protocol. With two rings and the gateway forwarding all

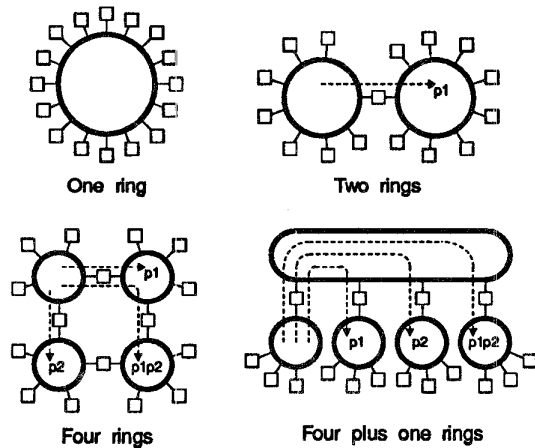


Figure 3: The four topologies considered. Parameters  $p_1$  and  $p_2$  determine the number of messages communicated from the source ring of a message to the other rings.

messages between them, a throughput of 631 ordered messages per second was achieved.

To investigate the performance of the Totem multiple-ring protocol with more processors than we currently have in our laboratory, we developed a simple analytic model. For the four topologies illustrated in Figure 3, we considered the effects of locality and the preferential assignment of processes in the same process groups to processors on the same rings. The probability that a message must be communicated to another ring is represented by two parameters,  $p_1$  and  $p_2$ . Intuitively,  $p_1$  represents the probability that a message originated in one half of the network must be forwarded to the other half, while  $p_2$  represents the probability that it must be forwarded to the other quarter of its half.

Figure 4 shows the latencies to delivery in agreed and safe order for various traffic levels and for each of the four topologies. Process group locality is good with  $p_1 = 0.1$  and  $p_2 = 0.2$ . The multiple-ring topologies show substantially lower latencies than a single ring at the same traffic level with the same number of processors. They are capable of sustaining substantially higher traffic levels with reasonable latencies.

## 8 Conclusions and Future Work

The Totem multiple-ring protocol uses a hierarchical approach to provide reliable totally ordered delivery of messages across a network formed from local-area networks interconnected by gateways. The protocol uses a novel strategy that employs timestamps to provide total ordering and sequence numbers from the single-ring protocol to ensure reliable delivery. The sequence numbers determine whether all messages with lower timestamps have been received. The strategy is computationally inexpensive and results in excellent performance, as demonstrated by the measurements from our implementation and the results of our analysis.

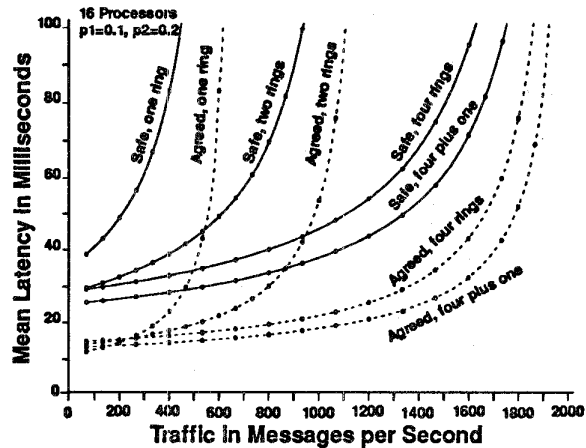


Figure 4: The latency of the Totem multiple-ring protocol for the four topologies when relatively few messages must be forwarded to the other rings.

The total order imposed on message delivery simplifies the maintenance of consistency of replicated data in fault-tolerant distributed systems and, thus, reduces the risk of programming errors in the application. A total ordering protocol with high throughput and low latency provides application programmers with a viable alternative to causally ordered and unordered delivery protocols.

The Totem multiple-ring protocol maintains strict consistency of message ordering despite network partitioning and remerging, or processor failure and recovery with stable storage intact. The global total order of messages ensures consistency of message ordering even when processes are members of multiple intersecting process groups. The multiple-ring protocol also maintains network topology information, based on membership information obtained for each ring from the single-ring protocol and forwarded through the network by the gateways.

Although the message ordering and topology maintenance mechanisms of the Totem multiple-ring protocol have been implemented and operate reliably and efficiently, there are still many issues that remain to be addressed. These include more effective flow control and better coupling between the routing and process group mechanisms. Currently, we are implementing the filtering mechanism at the gateways so that messages are forwarded only when necessary to reach the members of the process group. This will improve the performance of Totem across multiple interconnected local-area networks, while still providing strict consistency of message delivery.

## References

- [1] D. A. Agarwal. *Totem: A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks*. PhD Thesis, University of California, Santa Barbara, August 1994.

- [2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 76–84, Boston, MA, July 1992.
- [3] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. Fast message ordering and membership using a logical token-passing ring. In *Proceedings of the 13th International IEEE Conference on Distributed Computing Systems*, pages 551–560, Pittsburgh, PA, May 1993.
- [4] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.
- [5] K. P. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.
- [6] V. G. Cerf and R. E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, (5):647–648, May 1974.
- [7] J. M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.
- [8] M. F. Kaashoek and A. S. Tanenbaum. Group communication in the Amoeba distributed operating system. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 222–230, Arlington, TX, May 1991.
- [9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [10] P. M. Melliar-Smith, L. E. Moser, and D. A. Agarwal. Ring-based ordering protocols. In *Proceedings of the IEE International Conference on Information Engineering*, pages 882–891, Singapore, December 1991.
- [11] P. M. Melliar-Smith, L. E. Moser, and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):17–25, January 1990.
- [12] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 56–65, Poznan, Poland, June 1994.
- [13] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):459–473, May 1994.
- [14] T. Tachikawa and M. Takizawa. Selective total-ordering group communication on single high-speed channel. In *Proceedings of the 1994 International Conference on Network Protocols*, pages 212–219, Boston, MA, October 1994.
- [15] M. Takamura and M. Takizawa. Large-scale group communication protocol on high-speed channel. In *Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing*, pages 254–261, San Francisco, CA, August 1994.
- [16] R. van Renesse, T. M. Hickey, and K. P. Birman. Design and performance of Horus: A lightweight group communications system. Technical Report 94-1442, Cornell University, Department of Computer Science, August 1994.
- [17] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems*, Dagstuhl, Germany, September. Springer Verlag, Lecture Notes in Computer Science 938.