The Totem Redundant Ring Protocol

R. R. Koch, L. E. Moser, P. M. Melliar-Smith *
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{ruppert, moser, pmms}@alpha.ece.ucsb.edu

Abstract

Group communication protocols greatly simplify the design of fault-tolerant distributed systems. Most of those protocols focus on node redundancy rather than on network redundancy. The Totem Redundant Ring Protocol allows the use of multiple redundant local-area networks. The partial or total failure of a network remains transparent to the application processes. The distributed system remains operational while an administrator reacts to an alarm raised by the Totem Redundant Ring Protocol. The user can choose between active, passive and active-passive replication of the network.

1. Introduction

Group communication protocols [1, 2, 4, 12] must provide reliable delivery, ensured either by an underlying reliable protocol or by the group communication protocol itself. Properties such as virtual synchrony [4] and extended virtual synchrony [16] ease the maintenance of consistency of replicated data. Systems that are connected by a wide-area network [13, 20, 22] have a good chance of remaining operational if parts of the network fail. Local-area networks (LANs), on the other hand, often employ a single switch or a hub. If that component fails, no node can communicate with any other node and the system partitions into singletons. Systems that follow the primary component model [4] shut down all nodes, while other systems keep the nodes up, even though they cannot do useful work when the communication links are severed.

To allow a distributed system to tolerate network faults, the network itself must be replicated. Although replicated wide-area networks are not practical, LANs can be replicated cheaply. However, the mere presence of a redundant network does not overcome network faults. A special protocol must be employed to coordinate redundant networks.

*This research has been supported by DARPA/ONR Contract N66001-00-1-8931 and MURI/AFOSR Contract F49620-00-1-0330.

Such a protocol can be used in distributed applications with high availability requirements, such as financial, avionic, or military applications, that are based on clusters of computers, instead of dedicated hardware. The range of applications that can benefit from a redundant network protocol extends from real-time radar image analysis to back-end servers for financial applications. Other applications include more general fault tolerance infrastructures, such as AQuA [8] or Eternal [18], which build on a group communication protocol.

To enable the use of redundant networks in a fault-tolerant distributed system, we have developed the Totem Redundant Ring Protocol (Totem RRP), which is based on the Totem Single Ring Protocol (Totem SRP) [2]. The Totem SRP is a highly efficient group communication protocol for Ethernet-based LANs. Totem imposes a logical token-passing ring on the network. The token is used to achieve reliable delivery of messages, causal and total message ordering, flow control and fault detection. The Totem SRP also provides group membership services.

The Totem RRP provides the same services to application processes as the Totem SRP. However, the Totem RRP utilizes multiple networks to achieve resilience against partial or total network faults. Network faults remain transparent to the application processes, and the system remains operational as long as a single network is operational.

As shown in Section 8, the Totem RRP increases both reliability and throughput. This characteristic is important for building fault-tolerant distributed systems that handle heavy message loads, such as telecommunication switches and distributed real-time image processing systems, or reliable network storage devices.

2. The Totem Single Ring Protocol

The Totem Single Ring Protocol (SRP) is a group communication protocol designed for Ethernet-based LANs. The Totem SRP uses the native Ethernet broadcast service to broadcast messages efficiently. All data is sent in the form of packets using UDP.

Reliable message delivery and message ordering is achieved by imposing a logical token-passing ring on all participating nodes. A node is allowed to broadcast a message only if it holds the token. After sending the messages that have accumulated in its send queue, a node passes the token to the next node on the ring. For performance reasons, tokens are not broadcast. If the traffic is light, the token rotates very quickly. Unicasting the token implies that a node receives the token only once per rotation.

The strict sending schedule allows Totem to utilize an Ethernet far beyond the usual point of saturation. The requirement that only one node at a time can transmit data prevents collisions on the medium. With Totem, a throughput of more than 9,000 1 Kbyte msgs/sec has been achieved on a 100Mbit/sec Ethernet, which corresponds to a utilization of almost 90% (see Section 8).

When the application wishes to send a message, it passes the message to Totem, which stores the message in its send queue. The next time a node receives the token, it dequeues the messages from its send queue and broadcasts them in the order in which they were enqueued. Totem includes in each message header a unique sequence number. The token carries the sequence number seq of the last message broadcast on the ring. For each message that it broadcasts, a node increments seq and puts it in the message header. After broadcasting its messages, the node copies seq into the token and passes the token to the next node on the ring. The sequence number attached to each message imposes a global order on messages. Each node delivers the messages in the order of the sequence numbers included in the headers.

If a node misses a message, it detects a gap in the sequence numbers when it receives the next message or the token. If the gap still exists when the node receives the token, it puts a retransmission request into the token. The next token holder checks if it has a copy of the requested messages. If it does, it broadcasts those messages and removes the request from the token. Otherwise, it leaves the request in the token and forwards the token to the next token holder.

If two nodes A and B are missing the same message m, only a single retransmission will occur. Thus, if a node C retransmits m because it received A's request, B will receive m as well. As explained in Sections 5 and 6, this behavior simplifies the design of the Totem RRP.

In addition to ensuring reliable delivery, the token also serves as a fault detector. If a node has not received the token for a certain period of time, it starts the membership protocol. To avoid triggering the membership protocol because of token loss, a node periodically resends a copy of the last token it sent, as long as it has not received a message with a sequence number greater than that in the token. The reception of such a message indicates that the token has been received successfully by the next node on the ring. If a node receives a token with the same sequence number as the previous one, it recognizes that the token was retransmitted and ignores the token.[1]

3. Fault Model

The Totem protocol is designed to tolerate omission faults and node faults. To tolerate network faults, the distributed system must employ redundant communication channels. By connecting nodes through multiple networks, communication can be maintained as long as one network remains operational.

Throughout the rest of this paper, we assume N to be the number of redundant networks. We refer to the first network as n', the second network as n'', and so on. Messages (tokens) are denoted as m_s (t_s), where s is the sequence number of the message (token). To distinguish different copies of a message (token) that are sent over different networks, we mark them as follows: m'_s (t'_s) is the copy of m_s (t_s) sent via n', m''_s (t''_s) is sent via n'', etc.

In the case of redundant networks, the types of faults tolerated are:

- A node A is unable to send any data via a particular network n^x.
- A node A is unable to receive any data via a particular network n^x.
- A network n^x is unable to deliver any data from some subset of nodes to some other subset of nodes. These sets can overlap, and can even comprise the entire set of nodes.

Such types of faults do not result in membership changes because the affected nodes are still able to communicate through another network. Moreover, the network fault is hidden from the user application. However, the system can handle only so many network faults before it fails. Consequently, the Totem RRP monitors the behavior of the networks and raises an alarm if the network behavior deviates from normal behavior.

If the Totem RRP detects a network fault, it marks the network as faulty. The Totem network monitor, which operates entirely locally, is based on receiving messages and tokens; it does not send messages or probe connections. Once a monitor detects a network fault, the Totem RRP marks the network as failed and stops sending messages over it. At the same time, the Totem RRP issues a fault report to the user application process that is connected to it. Although

[1] If the token completes an entire rotation without any message being broadcast, the token sequence number remains unchanged. To prevent a node from regarding the new token as an identical copy of a token that it has seen previously, the token contains a *rotation_counter*, which the ring leader increments every time the token completes one rotation.

a node does not send via networks it has marked as faulty, it does accept messages that it receives via those networks, because the network fault might not yet have been detected by the network monitors of all other nodes. Consequently, those nodes keep sending messages and tokens via those networks.

A node's refusal to send via a particular network is interpreted as a fault by the monitors of the other nodes. If those monitors have not yet done so, they now mark the network as faulty and issue a fault report. The order in which the fault reports are issued and the content of those reports aids the user in diagnosing of the problem.

4. Replication Styles

For the Totem Redundant Ring Protocol, we have implemented three different styles of network replication:

- **Active replication:** In active replication, all messages and tokens are sent over all networks at the same time. All data is received multiple times. The bandwidth consumption increases linearly with the number N of networks. The maximum throughput equals the throughput of the slowest network. The system is able to mask the loss of a message on up to N-1 networks without any message retransmission delay.

- **Passive replication:** In passive replication, messages are sent alternately over one of the available networks. The bandwidth consumption equals the bandwidth consumption of an unreplicated system. In the fault-free case, the maximum throughput equals the sum of the throughputs of all networks. If one of the networks fails, the maximum throughput is reduced. If a message is lost, Totem must wait until the message has been retransmitted.

- **Active-passive replication:** This replication style is a mixture of active replication and passive replication. Every message or token is sent over K networks simultaneously ($1 < K < N$). The bandwidth consumption increases K-fold. The system is able to mask the loss of a message on up to K-1 networks without any message retransmission delay.

5. Active Replication

Using active replication, the Totem RRP sends every message over all N networks. For optimal performance, all networks should exhibit similar throughput and similar latency. The Totem RRP sends different copies of messages (tokens) in the same order: m'_s (t'_s) is sent first, m''_s (t''_s) is sent second, and so on.

In the fault-free case, UDP over IP over Ethernet preserves the sending order of messages if they are sent via the same network to the same recipient.[2] The FIFO behavior is violated only when a message is dropped. However, due to the asynchronous nature of the system, nodes may receive messages in an arbitrary order if the messages are sent via different Ethernets.

Assume that a sender sends a message m via multiple networks n^i, $1 \leq i \leq N$. Using active replication, the following inequalities hold for the timing of events for any x, y with $x < y$:

$$t_{send}(m^x) < t_{send}(m^y) \quad (1)$$
$$t_{send}(m^x) < t_{recv}(m^x) \quad (2)$$
$$t_{send}(m^y) < t_{recv}(m^y) \quad (3)$$

From inequalities (1) and (3), it follows that

$$t_{send}(m^x) < t_{recv}(m^y) \quad (4)$$

No relation between $t_{recv}(m^x)$ and $t_{recv}(m^y)$ exists.

Considering the relationship between a pair of messages, the following additional inequalities hold:

$$t_{send}(m_1^x) < t_{send}(m_1^y) < t_{send}(m_2^x) < t_{send}(m_2^y) \quad (5)$$
$$t_{recv}(m_1^x) < t_{recv}(m_2^x) \quad (6)$$
$$t_{recv}(m_1^y) < t_{recv}(m_2^y) \quad (7)$$

Again, no relationship between $t_{recv}(m_1^x)$ and $t_{recv}(m_2^y)$, or between $t_{recv}(m_1^y)$ and $t_{recv}(m_2^x)$ exists. Obviously, these inequalities also hold if we replace any of the messages m with a token t. For any pair of messages (tokens) and any pair of networks, six possible scenarios arise, as shown in Figure 1.

When using active replication, six additional requirements for the Totem RRP must be met in addition to the Totem SRP requirements:

A1: Each message must be delivered only once to the application. All duplicate messages must be suppressed. To keep the message delivery latency low, the Totem RRP must attempt to deliver a message when it is first received.

A2: A node can request a retransmission only if it has not received a message over any network. None of the scenarios shown in Figure 1 can trigger a retransmission for either Totem messages or tokens.

[2] This does not hold for wide-area IP networks because IP might choose different routes for different packets. Moreover, it does not hold for Ethernets if the messages are sent to different recipients via UDP. The sender s might send a packet m_2 to node r_2 before it sends a packet m_1 to node r_1, even if m_1 is passed to the UDP stack before m_2. A possible reason is that s might still be waiting for the ARP packet that resolves r_1's MAC address.

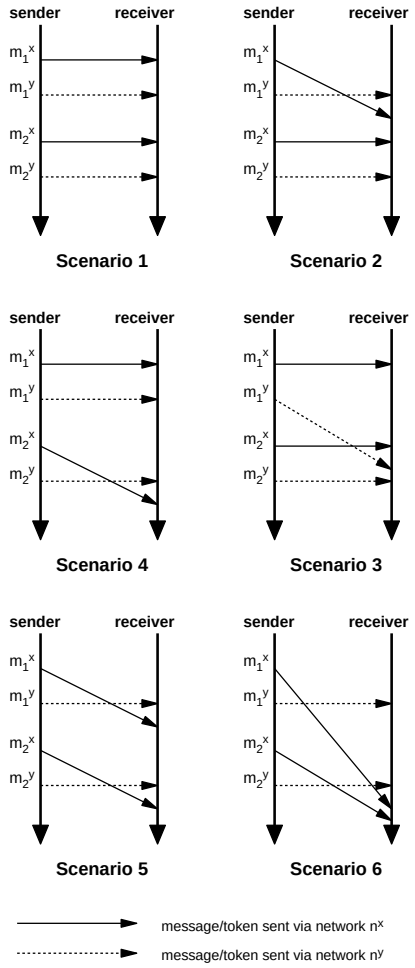

Figure 1. Timing of sending (receiving) messages sent via two networks.

A3: The networks must remain synchronized. If the networks are loaded differently, or if the networks have different speeds, the slower network must not fall behind.

A4: The Totem RRP must make progress even if a message or token is lost, or if a network fails.

A5: The Totem RRP must eventually detect a permanent failure of a network.

A6: The Totem RRP network fault detector must not be triggered by sporadic messages or token loss.

The algorithm given in Figure 2 transforms the Totem SRP into the Totem RRP for active replication. The algorithm forms a layer that resides between the Totem SRP and the networks. Each message and token is sent on all N networks.

When a message is received, it is passed directly to the Totem SRP. Identical copies of messages are destroyed by the Totem SRP. Because retransmitted messages are identical to the original messages, the Totem SRP implements a filter based on sequence numbers to eliminate duplicate messages. This mechanism also filters duplicate messages from different networks and, therefore, satisfies Requirement A1.

Unlike messages, tokens are passed to the Totem SRP only if they have been received via all non-faulty networks. This condition is necessary to satisfy Requirements A2 and A3. The Totem SRP tags retransmission requests to the token; thus, a node cannot request the retransmission of a message until it holds the token. Because of (6) and (7), all copies of messages that have been sent before a token has been sent, are received via all networks before all copies of the token are received. This means that the Totem SRP has received all outstanding messages before it processes the token and is able to issue a retransmission request (Requirement A2). Waiting for all copies of a token also prevents the networks from losing synchrony (Requirement A3), because the token is passed to the Totem SRP after the last copy of the token has been received.

The algorithm for active replication can cope with the loss of multiple copies of a message. If all copies are lost, the Totem SRP retransmission protocol resolves the problem. To guarantee progress when a token is dropped by some of the networks, or when some networks fail, the Totem RRP starts a token timer every time a new token is received over any of the networks. The token is passed to the Totem SRP when the token timer expires. All later copies of the token are ignored. This strategy implements Requirement A4. Note that, once the timer is running, it will never be restarted because a new token can arrive only after the current token has completed another rotation.

To monitor the health of the networks, the Totem RRP maintains a *problem_counter* for each network. If a token timer expires, the Totem RRP increments the *problem_counter* for all networks that did not deliver the token. If the *problem_counter* of a network exceeds a threshold, the Totem RRP declares the network to be faulty. This mechanism satisfies Requirement A5. To prevent the protocol from declaring an operational network as faulty simply because a number of token losses accumulated over time, a network's *problem_counter* is decremented periodically (not shown Figure 2.). This mechanism ensures that Requirement A6 is satisfied.

6. Passive Replication

Using passive replication, each node that runs the Totem RRP establishes connections over all of its networks. It sends a single copy of each message, and each token, over

```
boolean faulty[N]              = false
boolean recvLastToken[N]       = false
int     problemCounter[N]      = 0
token   lastToken              = {0, 0, ...}

sendMsg( m )
    for ( i = 1; i ≤ N; i++ )
        if ( faulty[i] = false )
            broadcast m via network n^i

sendToken( t )
    for ( i = 1; i ≤ N; i++ )
        if ( faulty[N] = false )
            send t to next token holder via network n^i

recvMsg( m, n^x )
    deliver m to Totem SRP

recvToken( t, n^x )
    if ( t.seq > lastToken.seq )
        lastToken = t
        for ( i = 1; i ≤ N; i++ )
            recvLastToken[i] = false
        recvLastToken[x] = true
        start token timer
    if ( t.seq = lastToken.seq )
        recvLastToken[x] = true
        for ( i = 1; i ≤ N; i++ )
            if ( recvLastToken[i] = false
                 ∧  faulty[i] = false )
                delete t
                return
            stop token timer
            deliver t to Totem SRP

tokenTimerExpired()
    for ( i = 1; i ≤ N; i++ )
        if ( recvLastToken[i] = false )
            problemCounter[i]++
    for ( i = 1; i ≤ N; i++ )
        if ( problemCounter[i] ≥ threshold )
            faulty = true
    deliver lastToken to Totem SRP
```

Figure 2. Algorithm for active replication.

one of the networks. A node assigns messages and tokens to the networks in a round-robin fashion.

As for active replication, passive replication requires the Totem protocol to satisfy the following additional requirements:

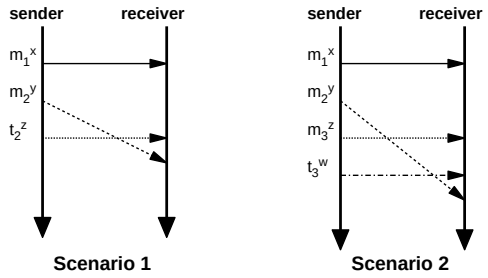

Figure 3. Out-of-order reception when using passive replication.

P1: A node can request a retransmission of a message only if the message has been dropped. None of the scenarios given in Figure 3 can trigger a retransmission of a delayed message.

P2: The networks must remain synchronized. If the networks are loaded differently, or if networks of different speeds are used, the slower network must not fall behind.

P3: The Totem RRP must make progress even if a message or token is lost, or if a network fails.

P4: The Totem RRP must eventually detect a permanent failure of a network.

P5: The Totem RRP network fault detector must not be triggered by sporadic messages or token loss.

These requirements are comparable to Requirements A2 to A6 for active replication. Because only a single copy of a message (token) is sent, there is no requirement for single message (token) delivery.

As in the case of active replication, we describe an algorithm that transforms the Totem SRP into the Totem RRP for passive network replication. The algorithm given in Figure 4 again forms a layer between the Totem SRP and the networks.

This algorithm sends a single copy of a message or a token. Received messages are passed to the Totem SRP directly. To satisfy requirement P1, the algorithm passes the token to the Totem SRP only if no message is missing. If there are outstanding messages, the token is stored in a token buffer and a timer is started. Upon expiration of the timer, the contents of the token buffer are passed to the Totem SRP. This step is necessary to make the algorithm comply to Requirement P3. The token timer is never restarted while it is active.

To improve performance (not necessary for correctness), the Totem RRP checks for a running token timer each time it receives a message. If the timer is running and no more messages are missing, *this* message is the reason that the

```
boolean faulty[N]  = false
int     lastSeq[N] = 0
token   lastToken  = {0, 0, ...}

sendMsg( m )
    do
        sendMessageVia = (sendMessageVia + 1)
                         mod N
    until ( faulty[sendMessageVia] = false )
    broadcast m via network n^sendMessageVia

sendToken( t )
    do
        sendTokenVia = (sendTokenVia + 1) mod N
    until ( faulty[sendTokenVia] = false )
    send t to next token holder via network n^sendTokenVia

recvMsg( m, n^x, s )
    deliver m to Totem SRP
    if ( anyMessagesMissing() = false
         ∧   tokenTimerRunning() = true )
        deliver lastToken to Totem SRP
        stop token timer
    messageMonitor(n^x, s)

recvToken( t, n^x )
    if ( anyMessagesMissing() = false )
        deliver t to Totem SRP
    else
        lastToken = t
        start token timer
    tokenMonitor(n^x)

tokenTimerExpired()
    deliver lastToken to Totem SRP
```

Figure 4. Algorithm for passive replication.

```
boolean recvCount[N] = false

monitor( n^x )
    recvCount[n^x] ++
    for ( i = 1; i ≤ N; i ++ )
    if ( max(recvCount[]) − recvCount[i] > threshold )
        faulty[i] = true
```

Figure 5. Network monitor module for passive replication.

token could not be delivered previously. Therefore, the timer is disabled and the contents of the token buffer are delivered. To provide fast recovery from message loss, the timer's timeout must be small. We chose a timeout of 10ms for our experiments.

As in active replication, the token resolves the issue of network desynchronization caused by differences in network speeds or loads. The Totem SRP must wait until it receives a new token before it can send any messages or tokens. Because all networks participate in transferring tokens in a round-robin fashion, the system will resynchronize when the token is sent via the slowest network. On average, this happens every Nth hop and, in the worst case, after $N-1$ complete rotations. This mechanism is sufficient to fulfill Requirement P2.

The network health monitor for passive replication consists of $M+1$ monitoring modules (where M is the number of nodes in the system): one module to monitor the message traffic for each of the nodes and one module to monitor the token traffic. Because tokens are unicast, the token monitor module is limited to the sending unit of the token sender, the receiving unit of the token receiver, and all network devices that are in the direct path. Although this might not cover all components of the networks, token monitoring is a useful alternative during periods in which no messages are sent.

Message and token monitoring modules are identical and are shown in Figure 5. Such a module counts the numbers of messages or tokens for each network. It checks whether all networks receive the same number of messages or tokens. If the difference in receptions exceeds a threshold, the network with the smaller number is marked as faulty. This mechanism ensures Requirement P4.

When running for an extensive period of time, sporadic loss events might accumulate and cause the monitor module to declare a healthy network as faulty, which violates Requirement P5. This condition is prevented by slowly increasing $recvCount$ for networks that lag behind. This mechanism, which can be either time or message driven, is not shown in Figure 5.

7. Active-Passive Replication

The active-passive replication style is a combination of active replication and passive replication. It can be used if there are at least three redundant networks available.

Active-passive replication uses a combination of the algorithms described in Sections 5 and 6 for active replication and passive replication. A node broadcasts K copies of each message and each token. Similar to passive replication, it sends messages and tokens to the networks in a round-robin fashion: If a node has sent its last message via network n^m, it sends the next message via networks $n^{(m+1) \bmod N}, \ldots, n^{(m+K) \bmod N}$. A similar scheme

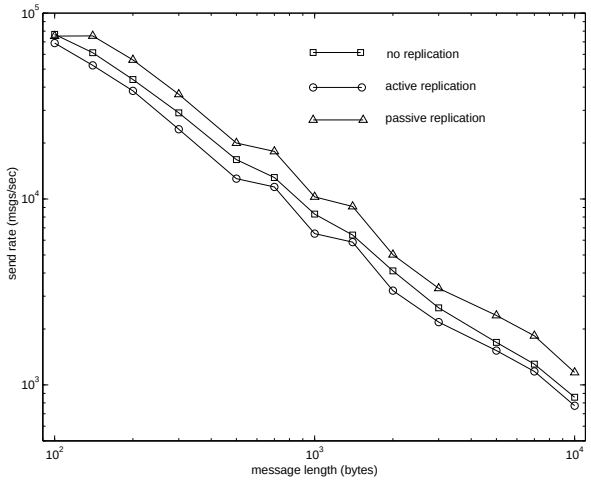

Figure 6. Transmission rate of the Totem RRP in msgs/sec for four nodes.

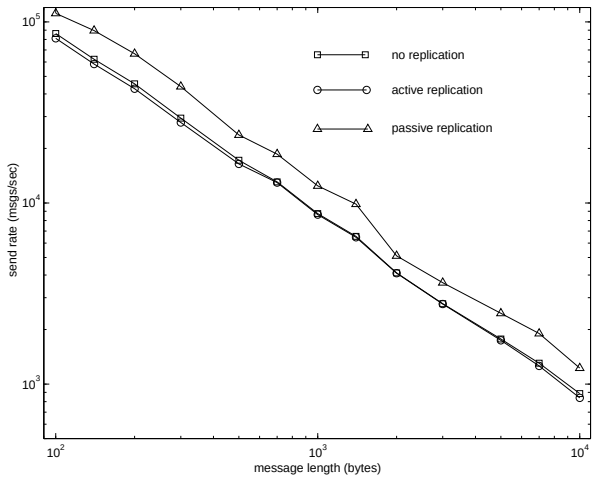

Figure 7. Transmission rate of the Totem RRP in msgs/sec for six nodes.

is used for sending tokens. If a node has sent its last token via network n^t, it sends the next token via networks $n^{(t+1) \bmod N}, \ldots, n^{(t+K) \bmod N}$.

On the receiver side, the active-passive replication algorithm can be regarded as a two-stage pipeline consisting of the algorithm used for passive replication, followed by the algorithm used for active replication. The first stage monitors the network by keeping track of the number of messages sent by each node via a particular network. It forwards all messages and all tokens to the second stage, which passes a token if it has received K copies of the token or when a timeout occurs. Again, duplicate messages are suppressed higher up in the Totem SRP protocol stack.

8. Performance

We have investigated the performance of the Totem Redundant Ring Protocol for two configurations. The first configuration consists of four Pentium II 450MHz workstations, and the second configuration consists of six Pentium III workstations with 900MHz and 1GHz clock frequency. All machines are equipped with two 3Com 3C905C TX-NM 100Mbit/s Ethernet network interface cards. The workstations ran the Linux operating system with kernel versions 2.2.15 and 2.2.17, which set the socket send and receive buffers to 64 Kbytes.

We conducted experiments with different message sizes for active replication and passive replication and for non-replicated networks. We did not conduct any experiments for active-passive replication, because it requires a minimum of three networks and we had only two networks available to us.

During the experiments, every node sent as many messages as the Totem flow control mechanism permitted. Figures 6 and 7 display the total send rate of the system as a function of message size, and Figures 8 and 9 show the utilized bandwidth. The peaks for message lengths of 700 and 1400 bytes are caused by the optimal usage of the Ethernet frame. The maximum frame size is 1518 bytes, of which 94 bytes are used for the Ethernet header and trailer, IPv4 header, UDP header and the Totem header. This results in a maximum payload of 1424 bytes for each Ethernet frame. If several messages can fit into that space, they are placed into a single packet by the message packing algorithm. If a message is longer than 1424 bytes, Totem splits it up into multiple packets.

Figures 6 to 9 show that active replication is the most expensive form of network replication. The overhead introduced by broadcasting all messages reduces the payload bandwidth up to 1000-1500 msgs/sec when compared to the unreplicated case. As for unreplicated networks, the 100 Mbit/sec Ethernet is the bottleneck when using active replication. The reduction in throughput is caused by doubling the number of calls to the network protocol stack.

Passive replication, on the other hand, allows the protocol to handle 2000-4000 Kbytes more payload every second than a system that runs on a single Ethernet. The network utilization of the two-way passively replicated network exceeds the capacity of a single 100 Mbit/sec Ethernet, but does not approach twice the transmission rate of the non-replicated system. This indicates that the available network bandwidth is no longer the limiting factor. Instead, the processing time associated with detecting and retransmitting missing messages, imposing a total order on the mes-

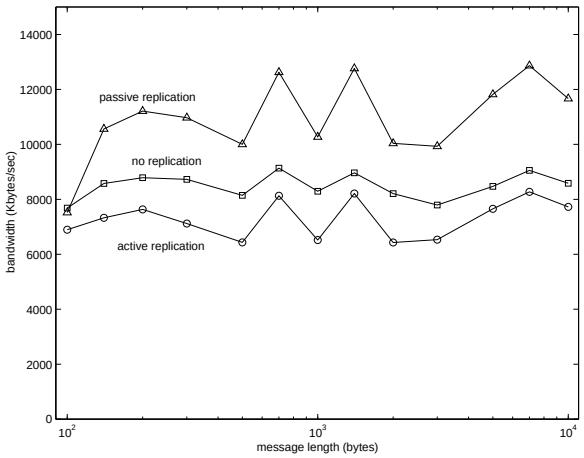

Figure 8. Transmission rate of the Totem RRP in Kbytes/sec for four nodes.

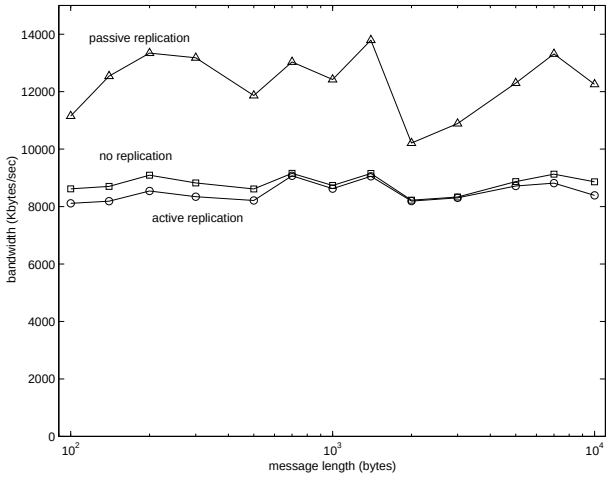

Figure 9. Transmission rate of the Totem RRP in Kbytes/sec for six nodes.

sages, and updating liveness information for all participating nodes determines the maximum throughput of the system. With faster processors, we expect the gap to widen between the throughput of the unreplicated system and the throughput of a system that uses passive replication.

9. Related Work

Over the past 15 years, there has been much work on group communication systems [1, 4, 5, 11, 15, 17]. Those systems are useful both for fault tolerant and highly available applications and for groupware and cooperative work applications.

The Isis, Horus and Ensemble systems [4, 23] provide the services of multicast, causal multicast and atomic (total order) multicast. Those group communication systems provide increasing flexibility in allowing the user to choose the protocol most appropriate for the application. While those systems provide excellent mechanisms for replicated processing to protect against faults in processes and processors, they do not provide mechanisms for replicated communication to protect against faults in the communication network.

The Trans/Total system [15] includes the Trans protocol which provides a causal order on messages, and the Total algorithm which converts this causal order into a total order. The Transis system [1] is based on the Trans protocol and on the Isis application programmer interface. Again, mechanisms are provided for replicated processing but not for replicated communication.

The real-time multicast protocol (RRTM) provides reliable ordered multicast communication for distributed real-time systems [5]. RRTM guarantees real-time message delivery without relying on synchronized clocks. The layering and modularization of RRTM allows an application to select a particular combination of atomicity and degree of ordering. RRTM requires networks that provide multicast features and that manage the group membership, but provides no mechanisms to handle non-transient network faults.

The systems mentioned above focus primarily on overcoming process and processor faults and transient network faults. Some provision is made to handle network partitioning faults, which are regarded as the loss of one or more processors. However, none of those systems is able to provide service in the presence of non-transient network faults.

The Software Implemented Fault Tolerance (SIFT) computer [24] and the Fault Tolerant Multiprocessor (FTMP) [10] are two systems designed for airplanes and spacecraft. Both systems use multiple point-to-point links to interconnect the processors, which ensures that communication faults cannot disable the system, but which prevents the system from scaling to more than a few processors.

The Tandem NonStopI, NonStopII, TXP, and VLX [9] connect up to 16 CPUs via dual 13 MByte/sec buses (Dynabuses). The buses tolerate hot-swapping of CPUs. A four-way redundant fiber optic bus extension allows the connection of up to 14 Dynabuses, which may be several kilometers apart. Tandem's Guardian operating system exploits those interconnects to provide high availability through transaction processing, rather than for fault tolerance using replication.

The Time-Triggered Protocol [14] is a hard real-time group communication protocol based on specialized hardware that accesses a shared bus. All components of the system are two-way redundant, including the communication

buses. The protocol assigns communication time slots to each sender according to a preplanned schedule. The system provides a high quality of fault tolerance, being resilient even to Byzantine faults, and protects the communication media from being over-utilized by any component. However, the design requires each message to be transmitted four times, and requires preplanned scheduling.

The Beowolf project [21] connects standard Linux workstations with multiple Ethernet networks to form a network of workstations (NOW). Multiple networks are to increase the available bandwidth of the cluster, rather than to achieve fault tolerance.

In [6] Christian describes a synchronous atomic broadcast protocol that is based on multiple redundant networks. Similar to our active approach, identical copies of a message are broadcast via all networks. The author assumes upper bounds for message delivery latency of the networks. This protocol is used in the Advanced Automation System (AAS) air traffic control network [7].

Delta-4 [19] is framework for a fault-tolerant real-time computing system. To communicate to groups of replicated objects an atomic multicast protocol, AMp, is used. AMp was implemented for redundant token buses and token rings, but can be adjusted to run on any broadcast channels.

In [3], the authors introduce a a reliable broadcast protocol based on redundant broadcast channels. Unlike the other work discussed here, which is based on a crash fault model, this protocol protects against Byzantine faults.

10. Conclusion

In this paper we have presented the Totem Replicated Ring Protocol (RRP), a group communication system that utilizes multiple networks to achieve resilience against network faults. The Totem RRP supports three styles of replication: active, passive and active-passive replication. We have measured the performance of the Totem RRP for active and passive replication and have compared it with the performance of an unreplicated system. Because of the additional network overhead, the system experiences a decrease in performance when run on an actively replicated network. Moreover, the throughput of the Totem RRP using passive replication exceeds the throughput of the unreplicated system while being more resilient against network faults.

References

[1] Y. Amir, D. Dolev, S. Kramer and D. Malki, "Transis: A communication subsystem for high availability," *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing*, Boston, MA (July 1992), pp. 76–84.

[2] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, "The Totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, no. 4 (November 1995), pp. 311–342.

[3] O. Babaoglu and R. Drummond, "Streets of Byzantium: Network architectures for fast reliable broadcasts," *IEEE Transactions on Software Engineering*, vol. 11, no. 6 (June 1985), pp. 546–554.

[4] K. Birman and R. van Renesse, "Reliable distributed computing with the Isis toolkit," IEEE Computer Society Press, 1994.

[5] W. C. H. Cheng, X. Jia and S. Kutty, "Towards fault-tolerant and synchronous multicast protocols for distributed systems," *Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems*, Dijon, France (September 1996), pp. 418–425.

[6] F. Cristian, "Synchronous atomic broadcast for redundant broadcast channels," *The Journal of Real-Time Systems*, vol. 2, no. 3 (September 1990), pp. 195–212.

[7] F. Cristian, B. Dancey and J. Dehn, "Fault-tolerance in air traffic control systems," *ACM Transactions on Computer Systems*, vol. 14, no. 2 (August 1996), pp. 265–286.

[8] M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr and R. Schantz, "AQuA: An adaptive architecture that provides dependable distributed objects," *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, West Lafayette, IN (October 1998), pp. 245–253.

[9] J. Gray, J. Bartlett and R. W. Horst, "Fault tolerance in Tandem computer systems," *The Evolution of Fault-Tolerant Computing*, eds. A. Avizienis, H. Kopetz and J. C. Laprie. Springer-Verlag, 1987.

[10] A. L. Hopkins, T. B. Smith and J. H. Lala, "FTMP – a highly reliable fault-tolerant multiprocessor for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10 (October 1978), pp. 1221–1240.

[11] W. Jia, J. Kaiser and E. Nett, "RMP: Fault-tolerant group communication," *IEEE Micro*, vol. 16, no. 2 (April 1996), pp. 59–67.

[12] K. P. Kihlstrom, L. E. Moser and P. M. Melliar-Smith, "The SecureRing protocols for securing group communication," *Proceedings of the IEEE Hawaii International Conference on System Sciences*, Kona, HI (January 1998), vol. 3, pp. 317–326.

[13] R. R. Koch, "The Atomic Group protocols: Reliable ordered message delivery for ATM networks," Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara (December 2000).

[14] H. Kopetz and G. Grunsteidl, "TTP – A protocol for fault-tolerant real-time systems," *Computer*, vol. 27, no. 1 (January 1994), pp. 14–23.

[15] P. M. Melliar-Smith, L. E. Moser and V. Agrawala, "Broadcast protocols for distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1 (January 1990), pp. 17–25.

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

[17] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM* vol. 39, no. 4 (April 1996), pp. 54–63.

[18] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, "Consistent object replication in the Eternal system," *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), pp. 81–92.

[19] D. Powell (Ed.), "Delta–4: A Generic Architecture for Dependable Distributed Computing," Springer Verlag, 1991.

[20] L. Rodrigues, H. Fonseca, P. Verissimo, "Totally ordered multicast in large-scale systems," *Proceedings of the IEEE 16th International Conference on on Distributed Computing Systems*, Hong Kong (May 1996), pp. 503–510.

[21] T. Sterling, *et al*, "BEOWULF: A parallel workstation for scientific computation,", *Proceedings of the 24th International Conference on Parallel Processing*, Oconomowoc, WI (1995), pp. 11–14.

[22] T. Tachikawa, H. Higaki, M. Takizawa, M. Gerla, M. T. Liu and M. Deen, "Flexible wide-area group communication protocols–International experiments," *Proceedings of the 1998 Workshop on Architectural and OS Support for Multimedia Applications*, Minneapolis, MN (August 1998), pp. 105–112.

[23] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd and D. Karr, "Building adaptive systems using Ensemble," *Software - Practice and Experience*, vol. 28, no. 9 (July 1998), pp. 963–979.

[24] J. H. Wensley, P. M. Melliar-Smith, *et al*, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proceedings of the IEEE*, vol. 66, no. 10 (October 1978), pp. 1240–1255.