A Reliable Dissemination Protocol for Interactive Collaborative Applications

Rajendra Yavatkar, James Griffioen, and Madhu Sudan
Department of Computer Science
University of Kentucky
Lexington, KY 40506
{raj,griff,madhu}@dcs.uky.edu
(606) 257-3961

ABSTRACT

The widespread availability of networked multimedia workstations and PCs has caused a significant interest in the use of collaborative multimedia applications. Examples of such applications include distributed shared whiteboards, group editors, and distributed games or simulations. Such applications often involve many participants and typically require a specific form of multicast communication called *dissemination* in which a single sender must reliably transmit data to multiple receivers in a timely fashion. This paper describes the design and implementation of a reliable multicast transport protocol called *TMTP* (Tree-based Multicast Transport Protocol). TMTP exploits the efficient best-effort delivery mechanism of IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into a hierarchical control tree. The control tree hierarchy employs *restricted nacks with suppression* and an *expanding ring search* to distribute the functions of state management and error recovery among many members, thereby allowing scalability to large numbers of receivers. An Mbone-based implementation of TMTP spanning the United States and Europe has been tested and experimental results are presented.

KEYWORDS

Reliable Multicast, Transport Protocols, Mbone, Interactive Multipoint Services, Collaboration

INTRODUCTION

Widespread availability of IP multicast [6, 2] has substantially increased the geographic span and portability of collaborative multimedia applications. Example applications include distributed shared whiteboards [15], group editors [7, 14], and distributed games or simulations. Such applications often involve a large number of participants and are interactive in nature with participants dynamically joining and leaving the applications. For example, a large-scale conferencing application (e.g., an IETF presentation) may involve hundreds of people who listen for a short time and then leave the conference. These applications typically require a specific form of multicast delivery called *dissemination*. Dissemination involves 1xN communication in which a single sender must reliably multicast a significant amount of data to multiple receivers. IP multicast provides scalable and efficient routing and delivery of IP packets to multiple receivers. However, it does not provide the reliability needed by these types of collaborative applications.

Our goal is to exploit the highly efficient best-effort delivery mechanisms of IP multicast to construct a scalable and efficient protocol for reliable dissemination. Reliable dissemination on the scale of tens or hundreds of participants scattered across the Internet requires carefully designed flow and error control algorithms that avoid the many potential bottlenecks. Potential bottlenecks include host processing capacity [18] and network resources. Host processing capacity becomes a bottleneck when the sender must maintain state information and process incoming acknowledgements and retransmission requests from a large number of receivers. Network resources become a bottleneck unless the frequency and scope of retransmissions is limited. For instance, loss of packets due to congestion in a small portion of the IP multicast tree should not lead to retransmission of packets to all the receivers. Frequent multicast retransmissions of packets also wastes valuable network bandwidth.

This paper describes the design and implementation of a reliable dissemination protocol called *TMTP* (Tree-based Multicast Transport Protocol) that includes the following features:

1. TMTP takes advantage of IP multicast for efficient

packet routing and delivery.

2. TMTP uses an *expanding ring search* to dynamically organize the dissemination group members into a *hierarchical control tree* as members join and leave a group.

3. TMTP achieves scalable reliable dissemination via the hierarchical control tree used for flow and error control. The control tree takes the flow and error control duties normally placed at the sender and distributes them across several nodes. This distribution of control also allows error recovery to proceed independently and concurrently in different portions of the network.

4. Error recovery is primarily driven by receivers who use a combination of *restricted negative acknowledgements with nack suppression* and periodic positive acknowledgements. In addition, the tree structure is exploited to restrict the scope of retransmissions to the region where packet loss occurs; thereby insulating the rest of the network from additional traffic.

We have completed a user-level implementation of TMTP based on IP/UDP multicast and have used it for a systematic performance evaluation of reliable dissemination across the current Internet Mbone. Our experiments involved as many as thirty group members located at several sites in the US and Europe. The results are impressive; TMTP meets our objective of scalability by significantly reducing the sender's processing load, the total number of retransmissions that occur, and the end-to-end latency as the number of receivers is increased.

Background

A considerable amount of research has been reported in the area of group communication. Several systems such as the ISIS system [1], the V kernel [4], Amoeba, the Psynch protocol [17], and various others have proposed group communication primitives for constructing distributed applications. However, all of these systems support a general group communication model (NxN communication) designed to provide reliable delivery with support for atomicity and/or causality or to simply support an unreliable, unordered multicast delivery. Similarly, transport protocols specifically designed to support group communication have also been designed before [13, 5, 3, 19, 9]. These protocols mainly concentrated on providing reliable broadcast over local area networks or broadcast links. Flow and error control mechanisms employed in networks with physical layer multicast capability are simple and do not necessarily scale well to a wide area network with unreliable packet delivery.

Earlier multicast protocols used conventional flow and error control mechanisms based on a *sender-initiated* approach in which the sender disseminates packets and uses either a *Go-Back-N* or a *selective repeat* mechanism for error recovery. If used for reliable dissemination of information to a large number of receivers, this approach has several limitations. First, the sender must maintain and process a large amount of state information associated with each receiver. Second, the approach can lead to a *packet implosion* problem where a large number of ACKs or NACKs must be received and processed by the sender over a short interval. Overall, this can lead to severe bottlenecks at a sender resulting in an overall decrease in throughput [18].

An alternate approach based on *receiver-initiated* methods [19, 15] shifts the burden of reliable delivery to the receivers. Each receiver maintains state information and explicitly requests retransmission of lost packets by sending negative acknowledgements (NACKs). Under this approach, the receiver uses two kinds of timers. The first timer is used to detect lost packets when no new data is received for some time. The second timer is used to delay transmission of NACKs in the hope that some other receiver might generate a NACK (called *nack suppression*).

It has been shown that the receiver-initiated approach reduces the bottleneck at the sender and provides substantially better performance [18]. However, the receiver-initiated approach has some major drawbacks. First, the sender does not receive positive confirmation of reception of data from all the receivers and, therefore, must continue to buffer data for long periods of time. The second and most important drawback is that the end-to-end delay in delivery can be arbitrarily large as error recovery solely depends on the timeouts at the receiver unless the sender periodically polls the receivers to detect errors [19]. If the sender sends a train of packets and if the last few packets in the train are lost, receivers take a long time to recover causing unnecessary increases in end-to-end delay. Periodic polling of all receivers is not an efficient and practical solution in a wide area network. Third, the approach requires that a NACK must be multicast to all the receivers to allow suppression of NACKs at other receivers and, similarly, all the retransmissions must be multicast to all the receivers. However, this can result in unnecessary propagation of multicast traffic over a large geographic area even if the packet losses and recovery problems are restricted to a distant but small geographic area[1]. Thus, the approach may unnecessarily waste valuable bandwidth.

In this paper we present an alternative approach that achieves scalable reliable dissemination by reducing the processing bottlenecks of sender-initiated approaches

[1]Assume that only a distant portion of the Internet is congested resulting in packet loss in the area. One or more receivers in this region may multicast repeated NACKS that must be processed by all the receivers and the resulting retransmissions must also be forwarded to and processed by all the receivers.

and avoiding the long recovery times of receiver-initiated approaches.

OVERVIEW OF OUR APPROACH

Under the TMTP dissemination model, a single sender multicasts a stream of information to a *dissemination group*. A *dissemination group* consists of processes scattered throughout the Internet, all interested in receiving the same data feed. A session directory service (similar to the session directory *sd* from LBL [12]) advertizes all active dissemination groups.

Before a transmitting process can begin to send its stream of information, the process must create a dissemination group. Once the dissemination group has been formed, interested processes can dynamically join the group to receive the data feed. The dissemination protocol does not provide any mechanism to insure that all receivers are present and listening before transmission begins. Although such a mechanism may be applicable in certain situations, we envision a highly dynamic dissemination system in which receiver processes usually join a data feed already in progress and/or leave a data feed prior to its termination. Consequently, the protocol makes no effort to coordinate the sender and receivers, and an application must rely on an external synchronization method when such coordination is necessary.

For the purposes of flow and error control, TMTP organizes the group participants into a hierarchy of subnets or *domains*. Typically, all the group members in the same subnet belong to a domain and a single *domain manger* acts as a representative on behalf of the domain for that particular group. The domain manager is responsible for recovering from errors and handling local retransmissions if one or more of the group members within its domain do not receive some packets.

In addition to handling error recovery for the local domain, each domain manager may also provide error recovery for other domain managers in its vicinity. For this purpose, the domain managers are organized into a *control tree* as shown in Figure 1. The sender in a dissemination group serves as the root of the tree and has at most K domain managers as children. Similarly, each domain manager will accept at most K other domain managers as children, resulting in a tree with maximum degree K. The value of K is chosen at the time of group creation and registration and does *not* include local group members in a domain (or subnet). The degree of the tree (K) limits the processing load on the sender and the internal nodes of the control tree. Consequently, the protocol overhead grows slowly, proportional to the Log_K(Number_Of_Receivers).

Packet transmission in TMTP proceeds as follows. When a sender wishes to send data, TMTP uses IP multicast to transmit packets to the entire group. The transmission rate is controlled using a sliding window based protocol described later. The control tree ensures reliable delivery to each member. Each node of

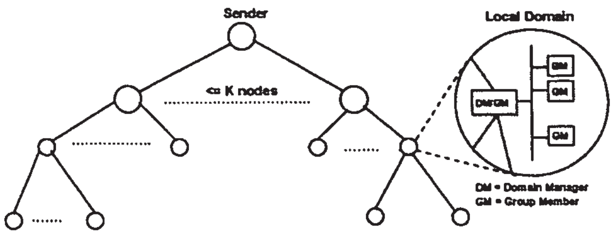


Figure 1: An example control tree with the maximum degree of each node restricted to K. Local group members within a domain are indicated by GM. There is no restriction on the number of local group members within a domain.

the control tree (including the root) is only responsible for handling the errors that arise in its immediate K children. Likewise, children only send periodic, positive acknowledgments to their immediate parent. When a child detects a missing packet, the child multicasts a NACK in combination with nack suppression. On the receipt of the NACK, its parent in the control tree multicasts the missing packet. To limit the scope of the multicast NACK and the ensuing multicast retransmission, TMTP uses the *Time-To-Live* (TTL) field to restrict the transmission radius of the message. As a result, error recovery is completely localized. Thus, a dissemination application such as a world-wide IETF conference would organize each geographic domain (e.g., the receivers in California vs. all the receivers in Australia) into separate subtrees so that error recovery in a region can proceed independently without causing additional traffic in other regions. TMTP's hierarchical structure also reduces the end-to-end delay because the retransmission requests need not propagate all the way back to the original sender. In addition, locally retransmitted packets will be received quickly by the affected receivers.

The control tree is self-organizing and does not rely on any centralized coordinator, being built dynamically as members join and leave the group. A new domain manager attaches to the control tree after discovering the closest node in the tree using an *expanded ring* search. Note that *the control tree is built solely at the transport layer and thus does not require any explicit support from, or modification to, the IP multicast infrastructure inside the routers.*

The following sections describe the details of the TMTP protocol.

GROUP MANAGEMENT

The session directory provides the following group management primitives:

CreateGroup(GName,CommType): A sender creates a new group (with identifier *GName*) using the CreateGroup routine. *CommType* specifies the type of communication pattern desired and may be either *dissemination* or *concast*[2]. If successful, CreateGroup returns an IP multicast address and a port number to use when transmitting the data.

JoinGroup(Gname): Processes that want to receive the data feed represented by *GName* call JoinGroup to become a member of the group. Join returns the transport level address (IP multicast address and port number) for the group which the new process uses to listen to the data feed.

LeaveGroup(Gname): Removes the caller from the dissemination group *GName*.

DeleteGroup(GName): When the transmission is complete, the sending process issues a DeleteGroup request to remove the group *GName* from the system. DeleteGroup also informs all participants, and domain managers that the group is no longer active.

CONTROL TREE MANAGEMENT

Each dissemination group has an associated control tree consisting of domain managers. Over the lifetime of the dissemination group, the control tree grows and shrinks dynamically in response to additions and deletions to and from the dissemination group membership. Specifically, the tree grows whenever the first process in a domain joins the group (i.e., a domain manager is created) and shrinks whenever the last process left in a domain leaves the group (i.e., a domain manager terminates).

There are only two operations associated with control tree management: *JoinTree* and *LeaveTree*. When a new domain manager is created, it executes the JoinTree protocol to become a member of the control tree. Likewise, domain managers that no longer have any local processes to support may choose to execute the LeaveTree protocol.

Figures 2 and 3 outline the protocols for joining and leaving the control tree. The join algorithm employs an *expanding ring search* to locate potential connection points into the control tree. A new domain manager begins an expanding ring search by multicasting a SEARCH_FOR_PARENT request message with a small time-to-live value (TTL). The small TTL value restricts the scope of the search to nearby control nodes by limiting the propagation of the multicast message. If the manager does not receive a response within some fixed timeout period, the manager resends the SEARCH_FOR_PARENT message using a larger TTL value. This process repeats until the manager receives a WILLING_TO_BE_PARENT message from one or more domain managers in the control tree. All existing domain managers that receive the SEARCH_FOR_PARENT message will respond with a

[2]Although this paper focuses on dissemination, TMTP also supports efficient concast style communication[10].

```
While (NotDone) {
   Multicast a SEARCH_FOR_PARENT msg
   Collect responses
   If (no responses)
      Increment TTL /* try again */
   Else
      Select closest respondent as parent
      Send JOIN_REQUEST to parent
      Wait for JOIN_CONFIRM reply
      If (JOIN_CONFIRM received)
          NotDone = False
      Else      /* try again */
}
```

(A) New Domain Manger Algorithm

```
Receive request message
If (request is SEARCH_FOR_PARENT)
    If (MAX_CHILDREN not exceeded)
        Send WILLING_TO_BE_PARENT msg
    Else
        /* Do not respond */
Else If (request is JOIN_REQUEST)
    Add child to the tree
    Send JOIN_CONFIRM msg
```

(B) Existing Domain Manger Algorithm

Figure 2: The protocol used by domain managers to join the control tree. A new domain manager performs algorithm (A) while all other existing managers execute algorithm (B).

```
If (I_am_a_leaf_manager)
   Send LEAVE_TREE request
    to parent
   Receive LEAVE_CONFIRM
   Terminate
Else /* I am an internal manager */
   Fulfill all pending obligations
   Send FIND_NEW_PARENT message to children
   Receive FIND_NEW_PARENT reply from all children
   Send LEAVE_TREE request to parent
   Receive LEAVE_CONFIRM
   Terminate
```

Figure 3: The algorithm used to leave the control tree after the last local group member terminates.

WILLING_TO_BE_PARENT message unless they already support the maximum number of children. The new domain manager then selects the closest domain manager (based on the TTL values) and directly contacts the selected manager to become its child. For each domain, its manager maintains a *multicast radius* for the domain, which is the TTL distance to the farthest child within the domain. The domain manager keeps the children informed of the current multicast radius. As described later in the description of the error control part of TMTP, both parent and its children in a domain use the current multicast radius to restrict the scope of their multicast transmissions.

Before describing the LeaveTree protocol, note that a domain manager typically has two types of children. First, a domain manager supports the group members that reside within its local domain. Second, a domain manager may also act as a parent to one or more children domain managers. We say a manager is an *internal manager* of the tree if it has other domain managers as children. We say a manager is a *leaf manager* if it only supports group members from its local domain.

A domain manager may only leave the tree after its last local member leaves the group. At this point, the domain manager begins executing the LeaveTree protocol shown in Figure 3. The algorithm for leaf managers is straightforward. However, the algorithm for internal managers is complicated by the fact that internal managers are a crucial link in the control tree, continuously servicing flow and error control messages from other managers, even when there are no local domain members left. In short, a departing internal node must discontinue service at some point and possibly coordinate children with the rest of the tree to allow seamless reintegration of children into the tree. Several alternative algorithms can be devised to determine when and how service will be cutoff and children reintegrated. The level of service provided by these algorithms could range from "unrecoverable interrupted service" to "temporarily interrupted service" to "uninterrupted service". Our current implementation provides "probably uninterrupted service" which means children of the departing manager continue to receive the feed while they reintegrate themselves into the tree. However, errors that arise during the brief reintegration time might not be correctable. We are still investigating alternatives to this approach.

After a departing manager has fulfilled all obligations to its children and parent, the departing manager instructs its children to find a new parent. The children then begin the process of joining the tree all over again. Although we investigated several other possible algorithms, we chose the above algorithm for its simplicity. Other, more static algorithms, such as requiring orphaned children to attach themselves to their grandparents, often result in poorly constructed control trees. Forcing the children to restart the join procedure ensures that children will select the closest possible connection point. Other more complex dynamic methods can be used to speed up the selection of the closest connection point but, in our experience, the performance of our simple algorithm has been acceptable.

DELIVERY MANAGEMENT

TMTP couples its packet transmission strategy with a unique tree-based error and flow control protocol to provide efficient and reliable dissemination. Conventional flow and error control algorithms employ a sender- or receiver-initiated approach. However, using the control tree, TMTP is able to combine the advantages of each approach while avoiding their disadvantages. Logically, TMTP's delivery management protocol can be partitioned into three components: data transmission, error handling, and flow control. The following sections address each of these aspects.

The Transmission Protocol

The basic transmission protocol is quite simple and is best described via a simple example. Assume a sender process S has established a dissemination group X and wants to multicast data to group X. S begins by multicasting data to the $\langle IP\_multicast\_addr, port\_no \rangle$ representing group X. The multicast packets travel directly to all group members via standard IP multicast. In addition, all the domain managers in the control tree listen and receive the packets directly.

As in the sender-initiated approach, the root S expects to receive positive acknowledgments in order to reclaim buffer space and implement flow control. However, to avoid the *ack implosion* problem of the sender initiated approach, the sender does not receive acknowledgments directly from all the group members and, instead, receives ACKs only from its K immediate children. Once a domain manager receives a multicast packet from the sender, it can send an acknowledgment for the packet to its parent because the branch of the tree the manager represents has successfully received the packet (even though the individual members may not have received the packet). That is, a domain manager does not need to wait for ACKs from its children in order to send an ACK to the parent. In addition, each domain manager only periodically sends such ACKs to its parent. This feature substantially reduces ACK processing at the sender (and each domain manager).

Error Control

Before describing the details of TMTP's error control mechanism we must define an important concept called *limited scope multicast* messages. A limited scope multicast restricts the scope of a multicast message by setting the TTL value in the IP header to some small value which we call the multicast radius. The appropriate multicast radius to use is obtained from the expanding ring search that domain managers use to join the tree. Limited scope multicast messages prevent messages targeted to a particular region of the tree from propagating throughout the entire Internet.

TMTP employs error control techniques from both sender and receiver initiated approaches. Like the sender initiated approach, a TMTP traffic source (sender) requires periodic (unicast) positive acknowledgements and uses timeouts and (limited scope multicast) retransmissions to ensure reliable delivery to all its immediate children (domain managers). However, in addition to the sender, the domain managers in the control tree are also responsible for error control after they receive packets from the sender. Although the sender initially multicasts packets to the entire group, it is the domain manager's responsibility to ensure reliable delivery. Each domain manager also relies on periodic positive ACKs (from its immediate children), timeouts, and retransmissions to ensure reliable delivery to its children. When a retransmission timeout occurs, the sender (or domain manager) assumes the packet was lost and retransmits it using IP multicast (with a small TTL equal to the multicast radius for the local domain so that it only goes to its children).

In addition to the sender initiated approach, TMTP uses *restricted NACKs with NACK suppression* to respond quickly to packet losses. When a receiver notices a missing packet, the receiver generates a negative acknowledgment that is multicast to the parent and siblings using a restricted (small) TTL value. To avoid multiple receivers generating a NACK for the same packet, each receiver delays a random amount of time before transmitting its NACK. If the receiver hears a NACK from another sibling during the delay period, it suppresses its own NACK. This technique substantially reduces the load imposed by NACKs. When a domain manager receives a NACK, it immediately responds by multicasting the missing packet to the local domain using a limited scope multicast message.

Flow Control

TMTP achieves flow control by using a combination of rate-based and window-based techniques. The rate-based component of the protocol prohibits senders from transmitting data faster than some predefined maximum transmission rate. The maximum rate is set when the group is created and never changes. Despite its static nature, a fixed rate helps avoid congestion arising from bursty traffic and packet loss at rate-dependent receivers while still providing the necessary quality-of-service without excessive overhead.

TMTP's primary means of flow control consists of a window-based approach used for both dissemination from the sender and retransmission from domain managers. Within a window, senders transmit at a fixed rate.

TMTP's window-based flow control differs slightly from conventional point-to-point window-based flow control. Note that retransmissions are very expensive because they are multicast. In addition, transient traffic conditions or congestion in one part of the network can put backpressure on the sender causing it to slow the data flow. To oversimplify, TMTP avoids both of these problems by partitioning the window and delaying retransmissions as long as possible. This increases the chance of a positive acknowledgement being received and it also allows domain managers to rectify transient behavior before it begins to cause backpressure.

TMTP uses two different timers to control the window size and the rate at which the window advances. $T_{retrans}$ defines a timeout period that begins when the first packet in a window is sent. Since the transfer rate is fixed, $T_{retrans}$ also defines the window size. A second timer, T_{ack}, defines the periodic interval at which each receiver is expected to unicast a positive ACK to its parent.

The sender specifies the value of T_{ack} based on the RTT to its farthest child. $T_{retrans}$ is chosen such that $T_{retrans} = n \times T_{ack}$, where n is an integer, $n \geq 2$. Both $T_{retrans}$ and T_{ack} are fixed at the beginning of transmission and do not change. A sender must allocate enough buffer space to hold packets that are transmitted over the $T_{retrans}$ period.

Figure 4 illustrates the windowing algorithm graphically. The sender starts a timer and begins transmitting data (at a fixed rate). Consider the packets transmitted during the first T_{ack} interval. Although the sender should see a positive ACK at time T_{ack}, the sender does not require one until time $T_{retrans}$. Instead, the sender continues to send packets during the second and third interval. After $T_{retrans}$ amount of time, the timer expires. At this point, the sender retransmits all unACK'd packets that were sent during the first T_{ack} interval. Retransmissions continue until all packets in the T_{ack} interval are acknowledged at which point the window is advanced by T_{ack}. On the receiving end, packets continue to arrive without being acknowledged until T_{ack} amount of time has expired[3].

[3]However, a receiver may generate a *restricted NACK* as soon as it detects a missing packet.

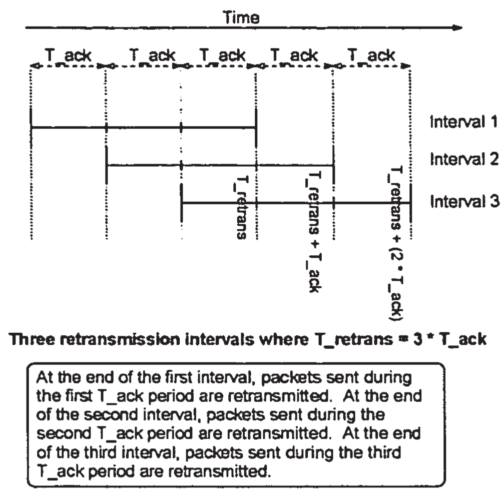
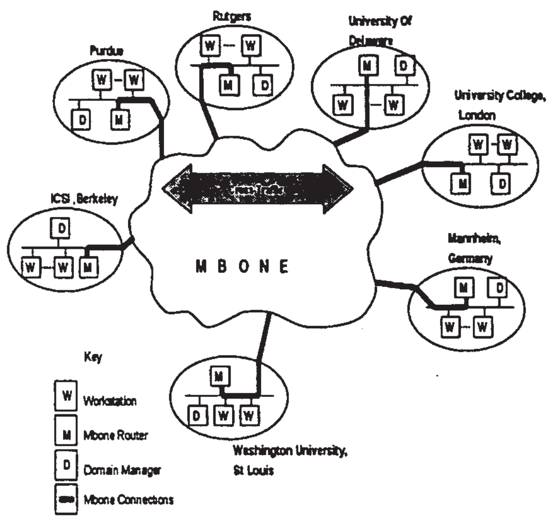
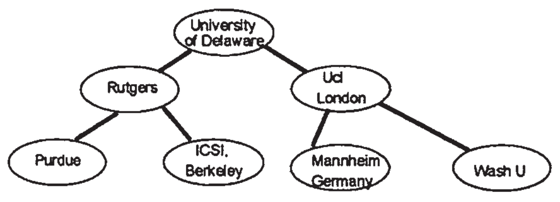

Time
T_ack
T_ack
T_ack
T_ack
T_ack
Interval 1
Interval 2
Interval 3
T_retrans
T_retrans + T_ack
T_retrans + (2 * T_ack)
Three retransmission intervals where T_retrans = 3 * T_ack
At the end of the first interval, packets sent during the first T_ack period are retransmitted. At the end of the second interval, packets sent during the second T_ack period are retransmitted. At the end of the third interval, packets sent during the third T_ack period are retransmitted.

Figure 4: Different Stages in Sending Data



(5a) The Internet Mbone Used



(5b) The Control Tree

Figure 5: Figure 5a shows the test environment consisting of seven geographically distant sites connected by the Mbone. Figure 5b shows the corresponding control tree configuration used in the experiments.

A domain manager must continue to hold packets in its buffer until all of its children have acknowledged them. If the children fail to acknowledge packets, the domain manager's window will not advance and its buffers will eventually fill up. As a result, the domain manager will drop and not acknowledge any new data from the sender, thereby causing backpressure to propagate up the tree which ultimately slows the flow of data.

There are three reasons for using multiple T_{ack} intervals during a retransmission timeout interval ($T_{retrans}$). First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. Second, a larger retransmission interval gives receivers sufficient time to recover missing packets using receiver-initiated recovery when only one (or a few) packets in a window are lost. This avoids unnecessary multicast retransmissions of a window full of data. Third, multiple T_{ack} intervals during the retransmission interval provide sufficient opportunity for a domain manager to recover from transient network load in its part of the subtree without unnecessarily applying backpressure to the sender.

We have chosen the value of the multiplying factor n to be 3 based on empirical evidence; the appropriate value depends on several factors including expected error rates, variance in RTT, and expected length of the intervals with transient, localized congestion. Further study is necessary to determine whether value of n should be chosen dynamically using an adaptive algorithm.

RESULTS

The Test Environment

Figure 5a illustrates the environment in which the experiments were run. Our tests involved seven geographically distinct Internet Mbone sites across the United States and Europe: Washington University in St. Louis, Purdue University, the International Computer Science Institute at Berkeley, Rutgers University, the University of Delaware, University College at London, and the University of Mannheim in Germany. All of our experiments were conducted using standard IP multicast across the Internet Mbone and thus experienced real Internet delays, congestion, and packet loss.

As a point of comparison, we implemented a standard sender-initiated reliable multicast transport protocol both with and without window-base flow control (called *WIN_BASEP* and *BURST_BASEP* respectively). Under both protocols, the sender maintains state information for all receivers, expects positive ACKs from each receiver, and uses timeouts and global multicast retransmissions to recover from missing acknowledgments. The two BASEP protocols illustrate the performance bottlenecks related to processor load and end-to-end latency. All three protocols used the same packet size (1 Kbytes). TMTP and WIN_BASEP used a window size of 5. TMTP uses a transmission rate of 10 packets per second, while both BASEP protocols transmit packets as fast as possible (up to the window size in the case of WIN_BASEP). Both BASEP protocols set the retransmission timeout period to be twice the RTT to the farthest site (approx. 2 seconds in our tests). TMTP uses a retransmission period of $T_{retrans} = N \times T_{ack}$. T_{ack} is dynamically set based on the RTT to the farthest group member (approximately 1.1 seconds for our tests). After some preliminary evaluation of different setting for N, our empirical results indicated that $N = 3$ provides sufficient time for local domains to recover without delaying acks unnecessarily or consuming too much buffer space. Consequently, $T_{retrans}$ was approximately 3.3 seconds in our tests. The following sections describe the performance measures used and detail the actual experiments performed.

Performance Measures

To evaluate the performance of our protocol, we identified two important measures of performance: *end-to-end delay* and *processing load*. In addition, we monitored the total number of retransmissions to estimate the amount of network traffic generated by TMTP.

From the application's perspective, the primary concern is the delay in reliably delivering the entire data feed (e.g., video, audio, or file data) to the multiple recipients of the group. To measure the end-to-end delay, we required that each receiving application send back a single positive acknowledgment (a GOT_IT message) to the sending application when the entire data transmission was complete. The sending application then calculated the end-to-end delay as the time between the beginning of the transmission and the time at which the last group member's final GOT_IT message is received.

From the network's perspective, the primary concern is network load and scalability of the algorithm. If the protocol provides low end-to-end delay but consumes large amounts of network resources, the protocol will not scale well, congesting the Internet by consuming shared resources required by other Internet users. There are two aspects to network load: processing load and bandwidth consumption. To measure the processing load at the sender, receivers, and domain managers, we monitored the following processing activities:

- receiving and processing a selective positive acknowledgment
- receiving and processing a negative acknowledgment
- handling a timer event (such as a retransmission timeout)
- performing a retransmission

Because it is hard to measure the amount of processing time needed for each of the events listed above (and highly dependent on the operating system and architecture), we have chosen to simply count the total number of such events at the sender to estimate the processing load generated by a protocol.

The second important measure of network load is bandwidth consumption. The precise amount of bandwidth consumed by each protocol is much harder to quantify since we were unable to collect traces of traffic across the Mbone to determine the number of links traversed and the amount of bandwidth consumed over each link. However, our results indicate that TMTP generated far fewer retransmissions than the BASEP protocols, and most TMTP retransmissions are local to a particular domain. For example, under the BASEP protocols most timeouts/retransmissions occurred as a result of dropped ACKs. TMTP's hierarchy substantially reduced the number of lost ACKs, experiencing only 6 local retransmissions totaled across all domain managers (four occurring concurrently) as opposed to 9 global retransmission for BURST_BASEP (out of thirty 1K messages).

Experiments Performed

Each of our experiments measured the performance of a single dissemination group consisting of many processes evenly distributed across the seven sites pictured in Figure 5a. The total number of processes acting as receivers was varied between five and thirty processes. The five process case used only five domains while all other cases used seven domains. In each experiment, a sending process created a dissemination group, waited for the receiving processes to join the group and organize their domains into a control tree. Multiple tree configurations are possible depending on when, and in what order, domain mangers join the tree. However to ensure consistency across tests, we held the tree configuration constant across all tests (see Figure 5b). After all receivers joined the group, the sender disseminated a data file to the group, and then waited for the final GOT_IT message from all receivers. The values reported for each test are averaged over at least five runs taken during weekdays at roughly the same time so that the observed Internet traffic conditions remain similar across tests.

To gauge the scalability of the protocol, we monitored the changes in processing load at the senders, receivers, and managers. To measure the effective throughput, we measured the changes in end-to-end delay as perceived by the sender. Both processing load and end-to-end delay were recorded under a variety of workloads. In the first set of tests, the sender transmitted a 30 Kbyte file to a varying number of receivers. The dissemination was considered complete when all the receivers correctly receive the entire file. In the second set of tests, the number of processes was fixed at 30 and we incrementally increased the file size from 3K to 30 Kbytes. The end-to-end delay is measured as the time between the beginning of the file transfer and the time at which the last group member's final GOT_IT message is received.

To measure the processing load, we counted the total

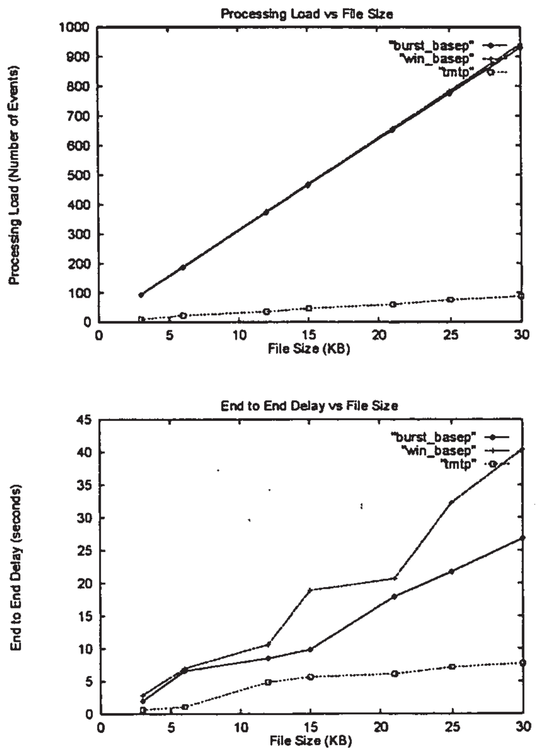


Figure 6: (a) Effect of the amount of data transmitted on the processing load. (b) Effect of the amount of data transmitted on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken with a dissemination group of size 30.

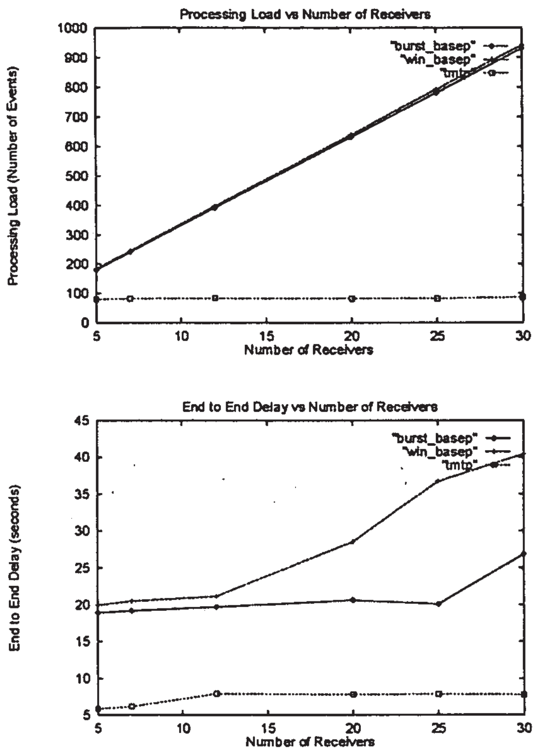


Figure 7: (a) Impact of group size (no. of receivers) on the processing load. (b) Impact of group size (no. of receivers) on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken for a dissemination of a 30 KB file.

number of events at the sender that contribute to the processing load. Similarly, we recorded the number of events at each domain manager. Figure 6 only shows the number of events processed at the sender. However, the balanced nature of our control tree meant the event processing load was spread equally among the sender and all domain managers. Consequently, the number of events processed at each domain manager is approximately the same as the number of events processed at the sender. Variations occurred based on the number of NACKs received.

Figures 6 and 7 show the results for each of the experiments performed. From these results we draw the following observations:

Impact of the Data Size

Figures 6a and 6b show how the file size affects the processing load and end-to-end delay. As the file size increases, the number of packets transmitted increases, thereby increasing the number of events (such as ACK/NACK processing or timer events) that affect the processing load at the sender (or a domain manager). Similarly, end-to-end delay is likely to increase due to time needed to deliver all the packets and due to increased probability of packet loss.

As the plots show, both the versions of the BASEP benchmark protocol show a significant increase in the processing load at the sender and the end-to-end delay. Note that the delay for WIN_BASEP (with flow control) is actually higher than BURST_BASEP (no flow control). This occurs because the WIN_BASEP sender expects acknowledgments from all its receivers before advancing the flow control window.

In the case of TMTP, the processing load shows only a small increase because the work is distributed among many nodes in the control tree. Consequently, the sender does not have to process acknowledgments or retransmission requests from all the receivers. TMTP's end-to-end delay is substantially lower than that of the BASEP protocols for all file sizes. Although all three protocols experience an increase in end-to-end delay resulting from larger data transmissions, packet losses, and retransmissions, TMTP's end-to-end delay rises at a significantly lower rate than that of the BASEP protocols. This occurs because error recovery in TMTP proceeds concurrently in different parts of the control tree rather than sequentially as in the BASEP cases.

Impact of the Group Size

Figures 7a and 7b show how the number of receivers (group size) affects the processing load and end-to-end delay.

Again, as the plots show, two versions of BASEP protocol show sharp increases in processing load with increase in number of receivers because the sender solely shoulders the responsibility for processing acknowledgments and retransmission requests (or timeouts) from each receiver. In the case of TMTP, the processing load at the sender (and each domain manger) is limited by the maximum number of immediate children in the control tree and, therefore, shows almost no increase as the number of receivers is increased. This results from the fact that the number of domains remains at seven for more than seven receivers. An increase in the number of domains participating in the dissemination group would cause a slight load increase on domain managers who adopt the new children.

Figure 7a shows that the end-to-end delay of both BASEP protocols is significantly higher than that of TMTP. The primary reason for this difference stems from TMTP's receiver-initiated capabilities that respond to and correct errors quickly. In contrast, the BASEP protocols will not correct an error until a retransmission timeout occurs.

In the case of TMTP end-to-end delays increases gradually because error recovery proceeds concurrently and independently in different parts of the control tree as explained earlier. Figure 7b shows that the end-to-end delay stabilizes to almost a constant value beyond a point. That is, to a small extent, an artifact of our tests in which we did not add any new domains to the control tree, but rather only added new processes to the existing tree. However, in other experiments involving varying number of domains, we have observed a similar trend of gradual increase in end-to-end delays with increasing number of receivers at additional domains.

RELATED WORK

A considerable amount of work has been reported in the literature regarding reliable multicast [13, 5, 3, 18, 12, 1, 4, 19, 15, 8, 11, 16]. Most of the earlier approaches achieve reliable delivery using a *sender-initiated* approach which is not suitable for large-scale, delay-sensitive, reliable dissemination.

Pingali and others[18] recently analyzed and compared both sender- and receiver-initiated approaches to demonstrate the limitations of the sender-initiated approach for large-scale dissemination. Our work is also motivated by similar observations, but combines the elements of both the approaches to achieve fast, local error recovery.

The reliable multicast protocol used in LBL's whiteboard tool (*wb*) [15, 8] and the log-based reliable multicast protocol [11] are two recent examples of the receiver-initiated approach for reliable delivery. Unlike TMTP, these protocols do not combine sender-initiated with receiver-initiated approaches and differ significantly in flow control mechanisms and buffering mechanisms. Our work is related to the *wb* work in that the *wb* protocol also uses a *NACKs with NACK suppression* mechanism. The *wb* protocol reduces state management overhead and achieves high degree of fault tolerance by relying solely on the receiver to recover from a packet loss. However, the protocol incurs the overhead of global (sometimes redundant) multicasts; a receiver multicasts a *repair request* to the entire group and one or more receivers in the group who have missing data (irrespective of their proximity to the complaining receiver) will multicast the missing packet(s) to the entire group even though the loss (or congestion) is restricted to a small region of the group topology. TMTP restricts the scope of multicast NACKs and retransmissions to the local domain to avoid generating redundant multicast transmissions over a wider region. Similar to TMTP, receivers using the wb protocol delay their NACKs to suppress duplicate NACKs in case another receiver multicasts a NACK. However, in the wb protocol, each receiver delays its NACK (and the response) by a random amount that depends on the RTT to the original sender. This can result in higher latency in recovering from packet losses. TMTP, on the other hand, uses localized recovery and, thus, the amount of random delay is bounded by the largest RTT between the local domain manager and one of the receivers in the domain. In addition, TMTP allows recovery from different errors to proceed concurrently in different domains to allow faster and efficient recovery.

Cheriton et. al.[8] have recently proposed a collection of strategies (called log-based receiver-reliable multicast or LRBM) for achieving large-scale, reliable multicast delivery. Some elements of LRBM are similar to TMTP's mechanisms to some extent. LRBM uses a hierarchy of logging servers with a primary log server responsible for sending positive acknowledgments to the multicast source. The primary log server stores the packets as long as an application desires and the receivers must recover from errors by contacting a logging server. A secondary server at each site may log received packets and satisfy local retransmission requests to reduce load on the primary server. Deployment of LRBM in the Internet is necessary to evaluate its performance in achieving reliable delivery in a wide area network environment.

Recently Paul et. al. [16] have proposed and are examining three multicast alternatives with features similar to those of TMTP. In contrast to these protocols, TMTP uses a multi-level hierarchical control tree and a dynamic group management protocol, as opposed to a static two-level hierarchy, to evenly distribute the protocol processing load and allow finer grained independent and concurrent error recovery. TMTP targets a best-effort multicast system such as IP multicast rather than an ATM-like network with allocated resources. TMTP imposes no additional load on network-level routers and requires no modification to the network-level routers, but yet incorporates both local retransmissions and combined acknowledgments. Furthermore, TMTP employs receiver-initiated recovery techniques (*restricted negative acknowledgments with nack suppression* combined with periodic positive acknowledgments) and a unique flow control mechanism that can provide quick recovery from transient congestion and lost acknowledgments.

CONCLUSION

Based on our experimental results, we believe that TMTP can scale well to provide reliable delivery on a large scale without sacrificing end-to-end latency. Under TMTP, the network processing load increases very gradually, indicating that the protocol will scale well as the number of receivers increases. Moreover, TMTP provides significantly better application-level throughput because of the concurrency resulting from local retransmissions as shown by the end-to-end measurements.

References

[1] Ken Birman and Thomas Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb 1987.

[2] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM Computer Communication Review*, 22(3):92–97, July 1992.

[3] J. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[4] David. R. Cheriton and W. Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[5] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proccedings of ACM SIGCOMM '88*, pages 247–256, August 1988.

[6] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[7] Prasun Dewan. A Guide to Suite: Version 1.0. Technical Report SERC-TR-60-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, February 1990.

[8] S. Floyd, V. Jacobsen, S. McCanne, C-G Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *sigcomm95*, 1995. to appear.

[9] I. Gopal and J. Jaffe. Point-to-multipoint Communication over Broadcast Links. *IEEE Transactions on Communications*, 32, September 1984.

[10] James Griffioen and Rajendra Yavatkar. Clique: A Toolkit for Group Communication using IP Multicast. In *Proceedings of the Workshop on Services in Distributed and Networked Environments*, June 1994.

[11] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *sigcomm95*, 1995. to appear.

[12] Van Jacobson. *SD: Session Directory*. Lawrence Berkeley Laboratory, March 1993.

[13] M Frans Kaashoek, A.S. Tanenbaum, S.F. Hummel, and H.E. Bal. An Efficient Reliable Broadcast Protocol. *ACM Operating Systems Review*, 23(4), October 1989.

[14] Amit Mathur and Atul Prakash. Protocols for integrated audio and shared windows in collaborative systems. In *Proceedings of ACM Multimedia '94*, October 1994.

[15] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Technical report, Real Time Systems Group, Lawrence Berkeley Laboratory, Berkeley, CA, September 1992. unpublished report.

[16] S. Paul, K. Sabnani, and D. Kristol. Multicast Transport Protocols for High Speed Networks. In *IEEE Int. Conf. on Network Protocols*, 1994 Oct.

[17] L. Peterson, N. Buchholz, and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[18] Sridhar Pingali, Don Towsley, and James F. Kurose. A comparision of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceddings of ACM SIGMETRICS '94*, volume 14, pages 221–230, 1994.

[19] S. Ramakrishnan and B.N. Jain. A Negative Acknowledgement Protocol with Periodic Polling Protocol for Multicast over Lans. In *Proccedings of IEEE INFOCOMM '87*, pages 502–511, March-April 1987.

Routing Strategies for Fast Networks

Yossi Azar
DEC - Systems Research Center
130 Lytton Ave.
Palo-Alto, CA 94301

Joseph Naor
Department of Computer Science
Technion
Haifa 32000, Israel

Raphael Rom
Sun Microsystems
Mountain View, CA
and
Technion, Haifa Israel

Abstract

Modern fast packet switching networks forced to rethink the routing schemes that are used in more traditional networks. The reexamination is necessitated because in these fast networks switches on the message's route can afford to make only minimal and simple operation. For example, examining a table of a size proportional to the network size is out of the question.

In this paper we examine routing strategies for such networks based on flooding and predefined routes. Our concern is to get both efficient routing and an even (balanced) use of network resources. We present efficient algorithms for assigning weights to edges in a controlled flooding scheme but show that the flooding scheme is not likely to yield a balanced use of the resources. We then present efficient algorithms for choosing routes along: (i) breadth-first search trees; and (ii) shortest paths. We show that in both cases a balanced use of network resources can be guaranteed.

1 Introduction

Traditional computer networks were designed on the premise of fast processing capability and relatively slow communications channels. This manifested itself by burdening network nodes with frequent network management decisions such as flow control and routing [1, 2, 3]. In a typical packet-switching network the routing decision at every node is based on the packet's destination and on routing information stored locally. This routing information may become quite voluminous, increasing the per-packet processing time.

Changes in technology, applications, and network sizes have forced to rethink these strategies. Modern fast packet switching networks [4, 5] relegate most of the routing computation to the end-nodes leaving all but the minimal computation to the intermediate nodes once the packet is on its way. This paper considers and compares several routing strategies for such fast networks. We assume that links are of high capacity so that message length is of no great concern. Computation capability in intermediate nodes is assumed limited so that all decisions made enroute should be simple and could not rely, for example, on generating random numbers or on tables that grow with the size of the network.

The first to encounter similar problems were the designers of parallel computers. Their solution, in the form of an interconnection network, typically derives the route directly from the destination address [6]. This approach, however, is limited to specific types of network topology and a structured layout which cannot be assumed for a general network. Furthermore, deriving the route from the address in general conflicts with alternate routing approach.

Flow-based techniques, used in many existing networks [7, 8], are also inadequate for our environment. These routing strategies are destination based (typically require a table entry per destination) but more importantly, result in bifurcated routing necessitating intermediate nodes to generate random numbers.

Two strategies are considered in this paper – controlled flooding and fixed routing. Flooding is a routing strategy that guarantees fast arrivals with minimal enroute computation at the expense of excessive bandwidth use. The scheme we use here, first proposed in [9], limits the extent to which a message is flooded through the network. Essentially, each link is assigned a cost for traversing it, thereby limiting the extent of the flood. The problem is to assign the link costs so as to achieve best performance. We show two methods of computing optimal weights that are drawn from a polynomial range (as opposed to the exponential range proposed in [9]). However, we do show that the assignment does not result in a routing scheme that uses network resources in a balanced way.

In the fixed routing scheme the route of the message is determined at the source node and is included in the message. No further routing decision are done enroute. The problem is therefore to find a set of routes, one for each pair of nodes, such that all the network's links will be used in a

This work was done while the author was in the department of Computer Science, Stanford University, CA 94305-2140, and was supported by a Weizmann Fellowship and contract ONR N00014-88-K-0166

Most of this work was done while the author was a postdoctoral fellow at the Computer Science Deptartment, Stanford University and supported by contract ONR N00014-88-K-0166.

balanced manner. We propose two methods to achieve this. In the first one, we force the messages to be routed along a (topological) breadth first search tree. The problem can be formulated as finding a set of rooted BFS trees such that the maximum load on a link is minimized. Notice that no link in the network remains unused. We provide polynomial algorithms to generate such a set of balanced routes.

In the second method, routing is done along paths that do not necessarily form trees. One of the shortest paths between every pair of nodes is designated as the path along which these two nodes exchange messages. We prove that a set of paths can be chosen that yields a balanced load. We define the notion of a balanced load with respect to randomized choices of paths, i.e., every pair chooses uniformly in random one of the shortest paths connecting them. We first show that with high probability the load on every edge will be close to its expected value. We then show how to construct deterministically in polynomial time such a set of balanced paths via the method of conditional probabilities.

2 Routing Along Trees

In this section we consider the option of routing along fixed BFS trees. Routing along trees can be viewed in two ways: (1) the tree rooted at a node specifies the routes used by the root when acting as a source of messages, or (2) the tree rooted at the node specifies the routes used by the other nodes with the root serving as the destination. From a design standpoint these are identical and in both we strive to balance the load on the links as much as possible.

As before we consider the network as a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. In addition we single out a vertex r called the root. The graph is divided into layers relative to root r by conducting a breadth-first search on G from r (i.e., we construct a tree of the shortest paths from r to all the other nodes in the graph). In this division, layer i, $0 \leq i \leq n-1$, contains all the vertices whose distance from r is i. The corresponding resultant tree is denoted T_r. Note that for a given G and r, the layers are defined uniquely but the BFS tree is not. Also note that given a BFS tree, the edges of the original graph connect vertices only from adjacent layers or in the same layer.

Let $v \in V$ be some vertex in layer i (for some $1 \leq i \leq n-1$). Define d_v^r as the number of neighbors of v at layer $i-1$ in graph G rooted at r; by convention $d_r^r = 0$. The following proposition establishes relations which we shall use later on.

Proposition 2.1 *For any graph G*

1. *The number of different BFS trees from root r is $\prod_{v \in G-r} d_v^r$*
2. *For any r, $\sum_{v \in V} d_v^r \leq m$*

Proof:

1. All the BFS trees can be constructed by having each vertex $v \in G - r$ choose independently a parent out of its neighbors in the previous layer, and each such construction corresponds to a legal and different BFS tree rooted at r. Hence the claim follows.
2. Each edge contributes unity to the sum if its two endpoint vertices are not in the same layer, and zero otherwise. Thus, this sum is exactly equal to the number of edges connecting vertices of different (and therefore adjacent) layers.

□

2.1 Homogeneous Sources

In this section we assume that each node sends (or receives) the same amount of data to every other node, and our aim, as we indicated, is to use the resources evenly. To that end we define the load on an edge as follows. Assume that for every vertex r in the graph we are given a single BFS tree rooted at that vertex (thus determining node's r routing). The load on an edge is defined (relative to this set of trees) as the number of trees which contain this edge. Formally, we are given a set $\{T_r\}_{r \in V}$ containing a single T_r for every $r \in V$ and we define the *load* of an edge as

$$l(e) = |\{r \in V | e \in T_r\}|.$$

Note that $l(e) \leq n$ and $\sum_{e \in E} l(e) = n(n-1)$, since there are n BFS trees with $n-1$ edges in each and each edge in a BFS tree contributes a unity to the sum. The capacity of an edge e, denoted $c(e)$, is defined as the maximum number of BFS trees that may contain it.

Our goal is to choose a set $\{T_r\}_{r \in V}$ such that the maximum load of the edges is minimized. We do this by solving a more general problem in which edges have limited capacities that are not necessarily equal. Assume that we are given the edge capacity $c(e)$ for each edge $e \in E$. We are seeking a feasible solution that is, a set $\{T_r\}_{r \in V}$ such that $l(e) \leq c(e)$ for all e. A solution for the capacitated problem can be easily used to solve the problem of minimizing the maximum load (in the uncapacitated problem). We just let $c(e) = c$ for all e and perform a binary search on $1 \leq c \leq n$, thereby increasing the complexity by a factor of $\log n$.

In order to solve the capacitated problem we define the following bipartite graph $H = (A \cup B, F)$. Side A consists of $n(n-1)$ vertices denoted by pairs (r, v) for all $v, r \in V$, $v \neq r$ (this pair will subsequently be interpreted as a root r and some vertex v in G). Side B consists of m vertices, each corresponding to (and denoted by) an edge e for all $e \in E$. Each vertex $(r, v) \in A$ is connected to a vertex $e \in B$ iff $\exists T_r$ (i.e., a tree rooted at r) in which $e \in E$ connects v to a vertex from the previous level.

Note that the degree of vertex (r, v) is d_v^r as per the definition of d_v^r. Also, from proposition 2.1 $|F| = \sum_{v,r} d_v^r \leq \sum_r m = nm$.

The key observation is that in order to solve our problem we need to find $n(n-1)$ edges in the graph H such that the degree of each vertex in A is exactly 1 (matching), and the degree of vertex $e \in B$ is at most $c(e)$. These edges define the n BFS trees in G. Specifically, the edges of T_r are the vertices in B which are adjacent to the vertices (r, v) for all $v \in G - r$. We present two algorithms for finding these trees.

Algorithm 1. Each vertex $e \in B$ with all its incident edges is duplicated $c(e)$ times, generating an "exploded" graph. Now, it is clear that solving the problem is equivalent to finding a perfect matching for side A into side B. The number of vertices in the exploded graph is $n(n-1) + \sum_e c(e) < n^2 + mn$ and the number of edges is at most $n|F| \leq n^2 m$. The complexity of computing a maximum matching in a bipartite graph is $O(|E|\sqrt{|V|}) = O(m^{3/2} n^{5/2})$ [11].

The latter complexity can be improved by the next algorithm.

Algorithm 2. Add to the graph $H = (A \cup B, F)$ a source node s and sink t. Add directed edges from s to all the vertices in A, each with capacity 1, and directed edges from each vertex $e \in B$ to t, each with capacity $c(e)$. Finally, direct all the edges from A to B and assign each the capacity 1 (any capacity greater than 1 will also do).

Consider an integer flow problem with source s and destination t obeying the specified capacities. It is clear that any such legal flow starts with some edges from s to A with flow 1. Then, each vertex in A that has an incoming edge with one unit of flow also has one outgoing edge with one unit flow to a vertex in B. Finally, all the flow reaching B continues to t. Thus we conclude that there is a feasible solution to our problem iff the maximum flow between s and t is exactly $n(n-1)$.

We will use Dinic's algorithm for finding the max-flow [12]. A careful analysis of the algorithm for our case yields a better complexity than more recent max-flow algorithms that perform better on general graphs. We first give a short review of Dinic's algorithm. The algorithm has $O(|V|)$ phases; at each phase only augmenting paths of length i, $1 \leq i \leq |V|$, are considered. The invariant maintained at phase i is that there are no augmenting paths of length less than i. The complexity of each phase is $O(|E||V|)$ in general graphs and $O(|E|)$ in 0-1 networks.

We first convert our graph into a 0-1 network. Each edge of capacity $c(e)$ is duplicated into $c(e)$ unity capacity edges which yields a 0-1 network. Since $c(e) \leq n$ for every edge e, the total number of new edges is at most nm and thus the number of edges remains $O(nm)$. As mentioned before, the complexity of Dinic's algorithm for 0-1 network is $O(|E||V|)$ which in our case becomes

$$O([n^2 + m][n^2 + mn + mn]) = O(n^2 \cdot mn) = O(mn^3)$$

In fact, the running time can be reduced to $O(mn^2)$. In our graph, there are no edges between vertices in A and also none between vertices in B, and there will not be such in any of the residual graphs. In fact, the residual graph will always start with s, end with t, have only vertices of A in the other even numbered layers and only vertices of B in the other odd-numbered layers. Moreover, the vertices of A will always have, in any residual graph, at most one incoming edge. Let us run the first $n-1$ phases of Dinic's algorithm (where each phase takes time $O(|F|) = O(nm)$). In phase n there will be at least n layers of A (unless we have already finished), one of them having at most $n(n-1)/n = n-1$ vertices. The incoming edges into this layer of A define a cut separating s from t whose capacity is at most $n-1$. Thus, Dinic's algorithm will terminate after at most additional $n-1$ phases, which gives the desired time bound.

2.2 Heterogeneous Sources

The situation at hand in this section is similar to that of the previous subsection except that we no longer assume homogeneous traffic but rather that each node generates a different amount of traffic. Translated into our model, this results in a problem with weighted trees. Formally, let the relative traffic intensity associated with node r be $w(r)$ (assumed to be an integer). This means that the tree associated with r (where r is the root) has a weight of $w(r)$ and we seek a set of BFS trees $\{T_r\}_{r \in V}$ with load $l(e) \leq c(e)$ for all e, where the load $l(e)$ is defined in the natural way, i.e.,

$$l(e) = \left\{ \sum_r w(r) | e \in T_r \right\}$$

The Capacitated Problem of the previous subsection is the special case of our problem with $w(r) = 1$ for all $r \in V$. While the Capacitated Problem in the homogeneous case has an efficient solution, we prove that in the heterogeneous case this problem is NP-complete (it is clear that the problem belongs to class NP). We base our proof on a reduction from the "knapsack" problem which is known to be NP-complete [13], defined as follows.

The Knapsack Problem: Given are integers $x_1 \ldots x_n$ and s. Are there $a_i \in \{0,1\}$, $1 \leq i \leq n$, such that $\sum a_i x_i = s$?

The Reduction: Consider a graph whose vertices are $v_1, \ldots v_n, u_1, u_2, t$. Connect v_i to u_j for $1 \leq i \leq n$, $j = 1, 2$ and connect u_1 and u_2 to t. Let the weight of the sources be $w(v_i) = x_i$ for all i, $w(u_1) = w(u_2) = w(t) = 0$. Finally, let the capacities of the edges be $c(u_1 t) = s$, $c(u_2 t) = \sum_i x_i - s$, and infinite (or big enough) for all the rest. It is clear that each BFS tree from v_i, $1 \leq i \leq n$, contains exactly one of the edges $u_1 t$ or $u_2 t$. Since $c(u_1 t) + c(u_2 t) = \sum_i x_i$, there is a solution iff there is a subset of the integers x_i that sums up to s.

Note that it is possible to eliminate the zero weights (and have the proof still hold) by assigning $w(u_1) = w(u_2) = w(t) = 1$ and also adding 2 to the capacities of the edges $u_1 t$ and $u_2 t$.

2.3 Randomized Capacity Bounds

In this section we develop upper bounds on the capacities that are needed for the edges in the Capacitated Problem of the homogeneous case (section 2.1) in order to achieve "good" load balancing. Our reference is a *random tree* routing scheme in which every node, whenever it needs to send a message, randomly and uniformly chooses a BFS tree in which it is a root, and routes according to this tree. Intuitively, such a routing scheme is likely to achieve a good balancing.

We start by calculating P_e^r - the probability that an edge e participates in a randomly and uniformly chosen BFS tree rooted at r. Let x_e^r be an indicator random variable indicating whether edge e belongs to the BFS tree rooted at r. By our definition

$$l(e) = \sum_{r \in V} x_e^r.$$

Consider an edge $e = (x, y)$. If both x and y are in the same layer (i.e., equidistant from r), then $P_e^r = 0$. Otherwise, they belong to adjacent layers (without loss of generality let x be the vertex that is further away from r), and $P_e^r = \frac{1}{d_x^r}$.

Let $\bar{l}(e)$ be the expected load of e. Clearly $E[x_e^r] = P_e^r$ and also

$$\bar{l}(e) = E\left[\sum_{r \in V} x_e^r\right] = \sum_{r \in V} E[x_e^r] = \sum_{r \in V} P_e^r$$

$$\begin{aligned} \sum_{e \in E} \bar{l}(e) &= \sum_{r \in V} \sum_{e \in E} P_e^r \\ &= \sum_{r} \sum_{x \neq r} \frac{1}{d_x^r} \cdot d_x^r = \sum_{r} n - 1 = n(n-1). \end{aligned}$$

Since $\sum_{e \in E} \bar{l}(e) = n(n-1)$ and also $\sum_{e \in E} l(e) = n(n-1)$, we cannot expect to find a set of BFS trees in which $l(e) \leq \bar{l}(e)$ for every edge e ($\bar{l}(e)$ is not necessarily an integer for instance). However, we can find a set which is almost as good. We show that there always exists a set of BFS trees $\{T_r\}_{r \in V}$ such that the load on any edge satisfies the following:

$$l(e) \leq \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}.$$

We will prove the claim via the probabilistic method; one can easily find such a set by applying the algorithm from section 2.1 as we are guaranteed that a solution exists.

To prove the bound on the load, we show that for each edge e, the probability that $l(e)$ exceeds the claimed bound is less than $\frac{1}{2m}$. Hence, there is a positive probability that the claim holds for all edges in the network. From Chernoff's bounds it can be shown that for all $\lambda \geq 0$,

$$\text{Prob}[l(e) > (1+\gamma)\bar{l}(e)] \leq \frac{E[e^{\lambda l(e)}]}{e^{(1+\gamma)\lambda \bar{l}(e)}}$$

and it can be shown [14] that there exists a choice of λ such that

$$\frac{E[e^{\lambda l(e)}]}{e^{(1+\gamma)\lambda \bar{l}(e)}} \leq e^{-\gamma^2 \bar{l}(e)/2}.$$

Assigning $\gamma = 2\sqrt{\frac{\log n}{\bar{l}(e)}}$, results in

$$\text{Prob}[l(e) > \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}] \leq \frac{1}{n^2} < \frac{1}{2m}$$

which finally yields

$$\text{Prob}[\forall e, l(e) \leq \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}] > \frac{1}{2}$$

meaning that a solution exists with a high probability.

3 Routing Along Shortest Paths

In this section we consider a different option of routing namely, routing along paths that do not necessarily form trees. One of the shortest paths between every pair of nodes is designated as the path along which these two nodes exchange messages. We prove that a set of paths can be chosen that yields a balanced load.

The proof we present follows the exact same lines of the proof in section 2.3 and we adopt the same notation. Again, our reference for a good load balancing is the *random path* routing scheme

We first evaluate P_e^{uv}–the probability that an edge e participates in a randomly and uniformly chosen shortest path connecting vertices u and v. (We will denote this event by the indicator variable x_e^{uv}). To compute this probability, we must count the shortest paths connecting u and v that contain edge e. Let $M_p(u, v)$ denote the number of paths of length p between the vertices u and v. The number of shortest paths between u and v can be computed in polynomial time by the following recursive formula. Let the vertices adjacent to u be $a_1, \ldots, a_d$ and let p be the length of the shortest path from u to v, then

$$M_p(u,v) = \sum_{i=1}^{d} M_{p-1}(a_i, v).$$

We consider a pair of nodes u and v and an edge $e = (x, y)$ (assume without loss of generality that vertex x is closer to u than vertex y). Denote by p_{uv} the distance between the vertices u and v, by p_{ux} the distance between u to x, and by p_{yv} the distance between v and y. Define $p' = p_{uv} - p_{ux} - 1$. If $p_{yv} > p'$, then $P_e^{uv} = 0$; otherwise,

$$P_e^{uv} = \frac{M_{p_{ux}}(u,x) \cdot M_{p_{yv}}(y,v)}{M_{p_{uv}}(u,v)}$$

Similar to the derivation in section 2.3 the expected load on an edge e is $\bar{l}(e) = \sum_{u,v \in V} P_e^{uv}$ and thus we cannot expect to find a set of shortest paths in which $l(e) \leq \bar{l}(e)$ for every edge e. However, again, we can find a set which is almost as good, namely, a set of shortest paths such that the load on any edge satisfies

$$l(e) \leq \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}.$$

An edge whose load does not satisfy the above condition is called an *overloaded* edge. If there are no overloaded edges, then the set of paths is called a *good set*. We will prove that a good set of paths exists via the probabilistic method and then show how to find such a set of paths deterministically.

Let every pair of vertices choose its path uniformly in random (among the shortest paths between them). We show that with high probability, the set of paths chosen is good. The random variable $l(e)$ is a sum of $\binom{n}{2}$ indicator variables x_e^{uv}. These variables are independent because each pair of vertices chooses its path independently of the other pairs. If we show that the probability that edge e is overloaded is less than $\frac{1}{2m}$, then with high probability the claim holds for all edges in the network. As stated in Section 2.3, it can be shown that for all $\lambda \geq 0$,

$$\text{Prob}[l(e) > (1+\gamma)\bar{l}(e)] \leq \frac{E[e^{\lambda l(e)}]}{e^{(1+\gamma)\lambda \bar{l}(e)}}$$

furthermore, there exists a choice of λ [14] such that

$$\frac{E[e^{\lambda l(e)}]}{e^{(1+\gamma)\lambda \bar{l}(e)}} \leq e^{-\gamma^2 \bar{l}(e)/2}$$

Similar to Section 2.3, assigning $\gamma = 2\sqrt{\frac{\log n}{\bar{l}(e)}}$, results in

$$\text{Prob}[l(e) > \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}] \leq \frac{1}{n^2} < \frac{1}{2m}$$

which finally yields

$$\text{Prob}[\forall e, l(e) \leq \bar{l}(e) + 2\sqrt{\bar{l}(e) \log n}] > \frac{1}{2}$$

as was claimed.

Having established that there exists a good set of paths we now show how to find this good set deterministically in polynomial time by the *method of conditional probabilities* [15],[16]. This method was introduced by Spencer [15] with the intention of converting probabilistic proofs of existence of combinatorial structures into efficient deterministic algorithms for actually constructing these structures. The idea is to perform a binary search of the sample space associated with the random variables so as to find a good set. At each step of the binary search, the current sample space is split into two halves and the conditional probability of obtaining a good set is computed for each half. The search is then restricted to the half having a higher conditional probability. The search terminates when only one sample point remains in the subspace, which must belong to a good set.

To apply this method to our case for finding a good set of paths, we will consider the indicator variables one-by-one. In a typical step of the algorithm, the value of some of the indicator variables has already been set, one variable is currently being considered, and the rest are chosen in random. (By choosing in random we mean that for the pair of vertices which is now being considered, the remainder of the path is chosen uniformly in random.) At each step we will compute the (conditional) probability of finding a good set if the variable considered is set to 0 and if it is set to 1.

We denote by P_j the probability of finding a bad set of paths after the variable considered at step j has already been assigned a value and by P_j^i the probability of obtaining a bad set of paths by assigning the value i, for $i = 0, 1$, to the variable considered at step j. Initially, it follows from the existence proof that the probability of choosing a good set of paths is positive; we inductively maintain that $P_j < 1$ for $j \geq 1$, and hence, either $P_j^0 < 1$ or $P_j^1 < 1$.

For the sake of simplicity, assume the following on the order in which the variables are considered:

- For a pair of vertices u and v, for all edges e, the variables x_e^{uv} are considered consecutively.
- For a pair of vertices u and v, the edges are considered according to their distance from u. (Ties are broken arbitrarily).

For example, suppose that we are considering the variable x_e^{uv} where $e = (a, b)$ and assume that vertex a is closer to u than b. Notice that by assigning a value to x_e^{uv},

- The probability P_f^{uv} may change for edges f for which x_f^{uv} has not been determined yet. (These changes in the probabilities can be computed in polynomial time.)

- The value of x_f^{uv} for other edges f may also be determined, e.g., if $x_e^{uv} = 1$, then for all edges f adjacent to a, $x_f^{uv} = 0$.

A major stumbling block in applying the method of conditional probabilities is always the computation of the conditional probabilities. In our case, we do not compute the exact probability that there exists an overloaded edge (even initially), but rather only *estimate* it. Consequently, if the estimator is not chosen judiciously, it may happen that when a variable is considered, according to the estimator, no value assigned to it can lead to a good solution. To overcome this difficulty, following Raghavan [16], the notion of a pessimistic estimator is introduced. We call $\hat{P}_j$ a *pessimistic estimator* of the conditional probability P_j if it satisfies the following conditions:

1. $\hat{P}_0 < 1$.
2. For any partial assignment of the first j variables, $P_j \le \hat{P}_j$.
3. $\min\{\hat{P}_j^0, \hat{P}_j^1\} \le \hat{P}_{j-1}$ where $\hat{P}_j^i$ is the estimator of P_j^i for $i = 0, 1$.
4. The pessimistic estimators can be computed in polynomial time.

It is not very hard to see that such a pessimistic estimator can equally well be used in the method of conditional probabilities instead of the exact conditional probabilities which are hard to compute in general. We now show that the pessimistic estimator that we will choose indeed satisfies the above conditions. We have earlier proved that initially,

$$\begin{aligned}\text{Prob [set is bad]} &\le \sum_{f\in E} \text{Prob}[l(f) > (1+\gamma_f)\bar{l}(f)] \\ &\le \sum_{f\in E} \frac{E[e^{\lambda_f l(f)}]}{e^{(1+\gamma_f)\lambda_f \bar{l}(f)}} < 1\end{aligned}$$

Notice that λ_f and γ_f depend on the edge f. We define

$$\hat{P}_0 = \sum_{f\in E} \frac{E[e^{\lambda_f l(f)}]}{e^{(1+\gamma_f)\lambda_f \bar{l}(f)}}$$

The estimator at Step j is defined to be

$$\hat{P}_j = \sum_{f\in E} \frac{E[e^{\lambda_f l_j(f)}]}{e^{(1+\gamma_f)\bar{l}(f)\lambda_f}}$$

where $l_j(f)$ is a random variable denoting the load on edge f at the end of Step j. For example, suppose that $l(f) = x_1 + x_2 + x_3 + x_4$ and at the end of Step j, $x_2 = 0$ and $x_4 = 1$. Then, $l_j(f) = 1 + x_1 + x_3$. ($\bar{l}(f)$, γ_f and λ_f retain their original values).

Condition (4) holds since the changes in the probabilities at each step can be computed in polynomial time as mentioned earlier. (Notice that the random variable $l_j(f)$ is the sum of independent random variables). Condition (2) holds since

$$\begin{aligned}P_j &\le \sum_{f\in E} \text{Prob}[l_j(f) > (1+\gamma_f)\bar{l}(f)] \\ &\le \sum_{f\in E} \frac{E[e^{\lambda_f l_j(f)}]}{e^{(1+\gamma_f)\lambda_f \bar{l}(f)}} = \hat{P}_j.\end{aligned}$$

Let us show that condition (3) holds as well. Suppose that at Step $j+1$ variable x_e^{uv} is being considered. By definition,

$$\begin{aligned}\sum_{f\in E} E[e^{\lambda_f l_j(f)}] &= P_e^{uv} \cdot \sum_{f\in E} E[e^{\lambda_f l_j(f)} | x_e^{uv} = 1] \\ &+ (1 - P_e^{uv}) \cdot \sum_{f\in E} E[e^{\lambda_f l_j(f)} | x_e^{uv} = 0]\end{aligned}$$

where the probability of choosing edge e as part of the path from u to v is P_e^{uv} (given the assignments of the previous j steps). Now,

$$\begin{aligned}\hat{P}_{j+1}^1 &= \sum_{f\in E} \frac{E[e^{\lambda_f l_j(f)} | x_e^{uv} = 1]}{e^{(1+\gamma_f)\bar{l}(f)\lambda_f}} \\ \hat{P}_{j+1}^0 &= \sum_{f\in E} \frac{E[e^{\lambda_f l_j(f)} | x_e^{uv} = 0]}{e^{(1+\gamma_f)\bar{l}(f)\lambda_f}}\end{aligned}$$

Hence,

$$\hat{P}_j = P_e^{uv} \cdot \hat{P}_{j+1}^1 + (1 - P_e^{uv}) \cdot \hat{P}_{j+1}^0$$

and clearly, $\min\{\hat{P}_{j+1}^0, \hat{P}_{j+1}^1\} \le \hat{P}_j$. The value of x_e^{uv} is set to the value for which $\hat{P}_{j+1}^i$ is minimized, for $i = 0, 1$.

4 Assigning Weights for Controlled Flooding

In this section we consider a more dynamic approach of routing—that of controlled flooding. Flooding is a routing strategy that guarantees fast arrivals with minimal enroute computation at the expense of excessive bandwidth use. To limit the extent of flooding we adopt the controlled flooding scheme first proposed in [9]. Consider a network in which each link is assigned a *weight* (sometimes referred to as *cost*) for traversing it and every message carries with it a *wealth*. A message arriving at an intermediate node will be duplicated and forwarded along all outgoing links (except the one it came from) whose cost is lower than the message wealth. The cost of the link is then deducted from the duplicated-message wealth. Consider for example the network in figure 1 depicting a message with a wealth of 10 arriving at node 2.

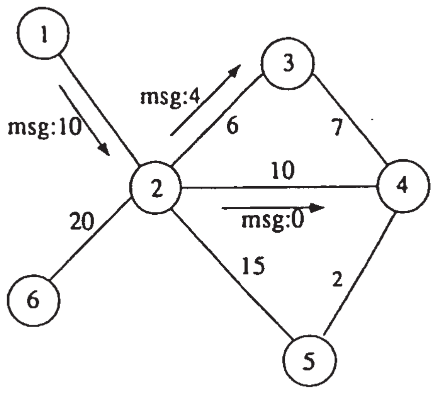


Figure 1: Example of controlled flooding

The link to node 3 has a cost of 6 associated with it resulting in a copy of the message with wealth 4 to be transmitted along that link. Similarly, a copy of the message with a wealth of 0 will arrive at node 4. Nodes 5 and 6 will not receive a copy of the message.

Since the controlled flooding scheme is a derivative of a flooding algorithm, it is impossible to assure that a message always arrives only at the nodes it is intended to. In particular, when used for point-to-point routing it is evident that more nodes than necessary might receive a message. In the above example, if the original message had arrived at node 2 with a wealth of 13 node 4 would have received two copies. Note also that there is no way for node 1 to send a message to node 4 without node 3 also receiving it. Clearly, different weight assignments may change the pattern of flooding.

The problem is to assign the link costs so as to achieve best performance. To that end a figure of merit is defined which is proportional to the (average) number of nodes that will receive every message. An optimal weight assignment is one that minimizes the figure of merit. To formalize our discussion let the network be represented by the graph $G(V,E)$ with $|V| = n$ and $|E| = m$, let the length of a path in the network be defined as the sum of the weights of the edges of the path, and let the shortest path between two nodes be the path with minimal length. Then, it is shown in [9] that for an assignment to be optimal, the following requirements (referred to as *optimality* requirements) must hold for every vertex (node) r:

- For every vertex $v \in V$, the shortest path from r to v is unique.
- For any two vertices $u, v \in V$, the length of the shortest path from r to u is different from the length of the shortest path from r to v.

Assignments that satisfy the above requirements are called *good*. An assignment is good with respect to r if all shortest paths from r satisfy the above requirements. Let us assume without loss of generality that the weights assigned are all positive integers.

Let $[1 \ldots R]$ denote the range of numbers from which weights are drawn and let n denote the number of nodes in the network. If $R = 2^{|E|}$, it is easy to find a good assignment [9]. For example, assigning 2^i as the weight of edge e_i assures that any two different paths will have different lengths. However, because the length of the path is carried by every message it is desirable to reduce R as much as possible.

We present two methods for constructing good assignments such that R is polynomial in n. In the first method the communication is restricted to a spanning tree T of the graph. This is done by assigning infinite weight to edges that are not in the tree. Denoting the tree edges by $e_1, \ldots e_i \ldots$, the algorithm is recursively defined as follows. Let v_l be a leaf of T, let u_l be its neighbor in the tree, and let e_l be the edge connecting u_l and v_l.

1. Compute (recursively) a good assignment for the tree $T - v_l$.
2. Extend the good assignment from $T - v_l$ to T.

We assume inductively that a good assignment was computed in Step 1. Step 2 can be implemented by checking all the values in the range $1 \ldots R$ and finding one that satisfies the requirements for a good assignment. Obviously, a good value for e_l exists if R is large enough. The next lemma bounds the value of R.

Lemma 4.1 *If $R \geq n^2$, then there exists a good assignment.*

Proof: Since a good assignment was computed for $T - v_l$ at Step 1, any value assigned to e_l will complete a good assignment with respect to v_l. The number of distinct values that e_l cannot assume is at most $(n-1)(n-2)$: for each vertex $r \in T - v_l$, the distance from r to v_l should be different from the distance from r to any other vertex, and thus, there can be at most $n - 2$ forbidden values (with respect to r), and the claim follows. □

The complexity of the weight assignment algorithm is $O(n^3)$ since each step can be implemented in $O(n^2)$ time. For each vertex $v_i \in V$, a table of all its distances to the other vertices is maintained and for each node all the forbidden values in the range $[1 \ldots n^2]$ are marked. One of the unmarked numbers is chosen arbitrarily for e_l. Then, the tables of all other nodes are updated.

The above assignment, being tree based, makes no use of many of the network links. The second assignment, which we present next, has the property that the whole network participates in the communication. We present two algorithms; the first is a randomized one that lends itself to distributed computation because the weight for each edge is chosen independently of the other edges. This algorithm generates a good assignment with high probability. The second algorithm is deterministic, and the weights are chosen from a smaller range than in the randomized algorithm.

Our main tool in the randomized case is the *Isolating Lemma* of Mulmuley, Vazirani and Vazirani [10]. A set system (S, F) consists of a finite set S of elements, $S = \{x_1, \dots, x_n\}$, and a family F of subsets of S, $F = \{S_1, \dots, S_k\}$. Let a weight w_i be assigned to each element of S. The weight of a subset is defined to be the sum of the weights of its elements.

Lemma 4.2 (Isolating Lemma) *Let $R \geq n$ and let (S, F) be a set system whose elements are assigned integer weights chosen uniformly and independently from the range $[1 \dots R]$. Then, Prob*[There is a unique minimum (maximum) weight set in F] $\geq 1 - \frac{n}{R}$.

(Note: the lemma in its original form in [10] was proven for $R = 2n$ but actually holds for all $R \geq n$). □

We start by proving that the following randomized process will generate a good assignment with high probability. Let a weight for each edge be chosen randomly and uniformly from the range $[1 \dots R]$.

Lemma 4.3 *For $R \geq n^4$ the probability that an assignment is good is at least $\frac{1}{2}$.*

Proof: Let A_{ij} be the event *the shortest path between nodes v_i and v_j is not unique.* Then $A = \cup_{i,j} A_{ij}$ is the event indicating the existence of at least one pair of nodes with non-unique shortest path between them. For each pair of nodes v_i and v_j let the set system F be the set of all paths connecting them. From the isolating lemma we have that the shortest path between them will be unique with probability at least $1 - \frac{n}{R}$, or, Prob$[A_{ij}] \leq \frac{n}{R}$. Hence, Prob$[A] \leq \sum_{i,j}$ Prob$[A_{ij}] \leq \binom{n}{2} \cdot \frac{n}{R}$.

Let B_{ijk} represent the event that nodes v_i, v_j, and v_k form a bad triplet, namely that the length of the shortest path between v_i and v_k equals that between v_j and v_k. $B = \cup_{ijk} B_{ijk}$ then represents the existence of at least one bad triplet in the network. In a way similar to the above we get Prob$[B] \leq \binom{n}{3} \cdot \frac{n}{R}$.

Finally, $A \cup B$ is the event indicating that the requirements are not met, and thus

$$\begin{aligned} \text{Prob}[\textit{good assignment}] &\geq 1 - Prob[A] - Prob[B] \\ &\geq 1 - \frac{n^2(n-1)}{2R} \\ &\quad - \frac{n^2(n-1)(n-2)}{6R}. \end{aligned}$$

For $R \geq n^4$, the right handside exceeds $\frac{1}{2}$. □

The last lemma provides us with a randomized distributed algorithm for constructing a good assignment. The probability of failure can be made arbitrarily small by increasing the value of R.

Notice that this method does not ensure that every edge participates in at least one shortest path. This can be fixed by forcing the weight assignment so that the BFS tree resulting from the weight assignment is also a BFS tree in the underlying graph without weights. To that end assign weights to the edges according to any of the above described algorithms and then add the value $n \cdot R$ to each weight. Now every edge takes part in at least one shortest path.

Next we show how a good assignment can be constructed deterministically. One way would be to derandomize the above randomized process. Notice that the proof of Lemma 4.1 actually implies that every partial assignment that does not violate the optimality requirements can be completed to a good assignment. We can thus assign weights to the edges one-by-one ensuring at every step that none of the requirements is violated.

A better way of doing this is by the following algorithm that constructs a good assignment with $R = n^3$ (compared with n^4). Initially, every edge e_i is assigned weight $n^4 \cdot 2^i$. The weights of the edges are then changed one-by-one to fit into the range $[1 \dots R]$ while maintaining the goodness of the assignment. At each step, the weight of the heaviest edge is changed.

Lemma 4.4 *If $R \geq n^3$, a good assignment can be constructed.*

Proof: The invariant which is maintained at the end of each step is that the assignment remains good. This is true initially. Let w_i be the new weight assigned to edge e_i at step i, where e_i connects vertices x and y. We prove that w_i can be fitted into the range $[1 \dots R]$ by bounding the number of forbidden values for w_i and showing that at least one permitted number exists. Let l_{uv} denote the value of the shortest distance between vertex u and vertex v when edge e_i is removed from the graph (l_{uv} might be infinite).

To maintain goodness we must accommodate both optimality requirement. We first show how to maintain the uniqueness of the shortest path between every pair of vertices. Let r and v be a pair of vertices, and assume without loss of generality that $l_{rx} < l_{ry}$. (They cannot be equal by the invariant). If the removal of edge e_i from the graph leaves vertices r and v in different connected components, then any value can be chosen for w_i with respect to r and v. Assume this is not the case. Since edge e_i had the largest weight in the graph (i.e., $n^4 \cdot 2^i$), the shortest path from r to v cannot contain edge e_i and l_{rv} is the value of the shortest distance from r to v. Hence, to maintain the uniqueness of the shortest path requirement, it is enough that

$$l_{rv} \neq l_{rx} + w_i + l_{yv}.$$

(Notice that the shortest path will remain unique even if it contains edge e_i, because of the uniqueness of the shortest paths from r to x and from y to v). This condition generates at most $n-1$ forbidden values for w_i with respect to every vertex r in the graph, or $n(n-1)$ forbidden values altogether.

Let us now show how the second requirement of optimality is maintained. Let r, u and v be a triplet of vertices. Again, notice that if the removal of edge e_i from the graph leaves vertex r in one connected component, and vertices u and v in a different connected component, then any value can be chosen for w_i with respect to r, u and v. The same holds if the removal of e_i leaves y separated from r, u, and v. Assume this is not the case. It follows from the above discussion that the shortest distance from r to u is either l_{ru}, or $l_{rx} + w_i + l_{yu}$. Similarly, the shortest distance from r to v is either l_{rv}, or $l_{rx} + w_i + l_{yv}$.

By the invariant,

$$l_{rv} \neq l_{ru} \quad \text{and} \quad l_{rx} + w_i + l_{yu} \neq l_{rx} + w_i + l_{yv}.$$

Hence, to maintain the second requirement of optimality, it is enough that

$$l_{rv} \neq l_{rx} + w_i + l_{yu}$$

and

$$l_{ru} \neq l_{rx} + w_i + l_{yv}.$$

These two conditions add at most $2 \cdot \binom{n-1}{2}$ forbidden values for w_i with respect to every vertex r in the graph, for a total of $2n \cdot \binom{n-1}{2}$.

Altogether, the number of forbidden values for w_i is $n(n-1)(n+1) < n^3$, and the lemma follows. □

Note that the initial assignment ($e_i = n^4 \cdot 2^i$) is chosen to ensure that every edge is treated exactly once, and when it is treated it does not participate in any shortest path unless it is a bridge.

The complexity of the algorithm is $O(n^3 m)$ since each step can be implemented in $O(n^3)$ time. Every vertex $v_i \in V$ maintains a table with all its shortest distances to the other vertices; it then marks all the forbidden values in the range $[1 \dots n^3]$. One of the unmarked numbers is chosen arbitrarily for e_i. Then, the tables of all other vertices are updated.

The reason why the range can be made smaller in the deterministic case is that it is enough to ensure at each step that there is *one* good value, whereas in the randomized case, one has to ensure success with high probability.

A desirable property of a routing scheme is having the traffic be evenly distributed among the edges. Unfortunately, this is the drawback of routing with random weights. The following example shows that with high probability this scheme does not yield a balanced load.

Let the load on an edge be defined as the number of shortest paths that contain it, and consider a graph made of two cliques of size k that are interconnected by two edges, e_1 and e_2. The weight for each edge is chosen uniformly and independently from the range $[1 \dots R]$. In each clique, the distribution of the weights is uniform and thus, if the weights of e_1 and e_2 are not close to one another, most of the traffic between the two cliques would go through the edge with smaller weight. Since this event will happen with high probability, the communication would not be balanced with high probability.

5 Conclusion

In this paper we examined several routing strategies for fast modern packet switching networks. The relevant characteristic of these networks is the inability to make elaborate routing decisions while packets are being switched. At the switching speeds being considered, looking up a table whose size is proportional to the number of network nodes is considered too costly.

These requirements limit the number of applicable routing strategies. The simplest and most natural strategy is to use fixed routing schemes in which the route between every pair of source-destination nodes is fixed in advance. The problem would then be to find a set of routes so that network resources are utilized as evenly as possible. Two such strategies are analyzed in this paper: routing along trees and routing along paths. For both cases polynomial algorithms are devised. we show that in both cases no network link remains unused but that routing along paths is likely to be a better strategy from load balancing standpoint.

Deviating from the fixed routing scheme we analyze a controlled flooding scheme in which every message essentially floods the networks but the extent of its flooding can be controlled by link weights. We provide a polynomial algorithm to compute these weights but show that the scheme cannot guarantee a good balance of load.

Acknowledgement

We would like to thank Noga Alon for many helpful discussions on this paper and in particular for his help in analyzing the algorithm of Section 2.1.

References

[1] A. Ephremides, "The routing problem in computer networks," in *Communications and Networks* (I. Blake and H. Poor, eds.), pp. 299–324, New York: Springer Verlag, 1986.

[2] M. Schwartz and T. Stern, "Routing techniques used in computer communication networks," *IEEE Trans. on Communications*, vol. COM-28, pp. 539–555, April 1980.

[3] P. Green, "Computer communications: Milestones and prophecies," *IEEE Communications*, pp. 49–63, 1984.

[4] I. Cidon and I. Gopal, "Paris: An approach to integrated high-speed private networks," *International Journal of Digital and Analog Cabled Systems*, vol. 1, pp. 77–86, April-June 1988.

[5] J. Turner, "Design of a broadcast packet switching network," *IEEE Trans. on Communications*, vol. COM-36, pp. 734–743, June 1988.

[6] H. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies.* Lexington, MA: Lexington Books, 1984.

[7] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store and forward communication network design," *Networks*, vol. 3, no. 2, pp. 97–133, 1973.

[8] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. on Communications*, vol. COM-25, pp. 73–85, January 1977.

[9] O. Lesser and R. Rom, "Routing by controlled flooding in communication networks," in *Proceedings of IEEE Infocom'90*, (San Francisco, California), pp. 910–917, IEEE, June 1990.

[10] K. Mulmuley, U. Vazirani, and V. Vazirani, "Matching is as easy as matrix inversion," *Combinatorica*, vol. 7, no. 1, pp. 105–113, 1987.

[11] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," *Siam J. Computing*, vol. 2, pp. 225–231, 1973.

[12] S. Even, *Graph Algorithms.* New York: Computer Science Press, 1979.

[13] M. Garey and D. Johnson, *Computers and Intractability.* San Francisco: W.H. Freeman and Company, 1979.

[14] D. Angluin and L. G. Valiant, "Fast probabilistic algorithms for hamiltonian circuits and matchings," *Journal of Computer and System Sciences*, vol. 18, pp. 155–193, 1979.

[15] J. Spencer, *Ten Lectures on the Probabilistic Method.* Philadelphia, Pennsylvania: SIAM, 1987.

[16] P. Raghavan, "Probabilistic construction of deterministic algorithms: Approximating packing integer programs," *Journal of Computer and System Sciences*, vol. 37, pp. 130–143, October 1988.

Business Wire
The Global Leader in News Distribution

Boeing and Panthesis Complete SWAN Transaction

Business Wire; New York; Jul 22, 2002; Business Editors & Aerospace Writers;

*NAICS:*336411 *NAICS:*336413 *NAICS:*336414 *Duns:*00-925-6819
Start Page: 1
Companies: Boeing Co *Ticker:*BA *Duns:*00-925-6819 *NAICS:*336411 *NAICS:*336413 *NAICS:*336414

Abstract:
IRVINE, Calif.--(BUSINESS WIRE)--July 22, 2002--The Boeing Co. and Panthesis Inc., today announced that they have completed a transaction that gives Boeing an equity stake in Panthesis and provides Panthesis with an exclusive right to commercialize Boeing's Small-world Wide Area Networking (SWAN) technology.

Based in Bellevue, Wash., Panthesis, was established in 2001 to develop and commercialize innovative software technology. Its co- founders, current Chief Development Officer Dr. Fred Holt and Chief Technology Officer Virgil Bourassa, are both former employees of The Boeing Co., where they co-invented SWAN technology while working in the Mathematics and Computing Technology unit of the Boeing Phantom Works R&D division.

Full Text:

IRVINE, Calif.--(BUSINESS WIRE)--July 22, 2002--The Boeing Co. and Panthesis Inc., today announced that they have completed a transaction that gives Boeing an equity stake in Panthesis and provides Panthesis with an exclusive right to commercialize Boeing's Small-world Wide Area Networking (SWAN) technology.

SWAN technology was originally developed by Boeing to allow multiple geographically dispersed people to conduct collaborative meetings and engineering design reviews in real time.

"SWAN is a revolutionary technology that can be used to enhance numerous computing, networking and communications functions," said Linda Magnotti, CEO of Panthesis. "The sophisticated mathematics and software architecture underlying SWAN technology can provide reliable server-less communication for communities anywhere in the world."

Magnotti added that Panthesis is currently focusing its development efforts on providing the bandwidth multiplication needed for use in massive multi-player online games, real-time online auctions, content distribution and other large-scale, unlimited online collaborations.

Based in Bellevue, Wash., Panthesis, was established in 2001 to develop and commercialize innovative software technology. Its co- founders, current Chief Development Officer Dr. Fred Holt and Chief Technology Officer Virgil Bourassa, are both former employees of The Boeing Co., where they co-invented SWAN technology while working in the Mathematics and Computing Technology unit of the Boeing Phantom Works R&D division.

"Because Panthesis clearly has the expertise for adapting SWAN technology to a broad range of potential applications, we were confident in giving them the exclusive right to commercialize this technology in the global marketplace," explained Gene Partlow, vice president of Boeing's Intellectual Property Business.

The potential for this agreement was created through Boeing's Chairman's Innovation Initiative, which promotes the development of new business ventures based on entrepreneurial ideas from employees. While some ideas are developed into spin-off companies, others are spun into Boeing business units for further development or, like SWAN, into the Intellectual Property Business for other types of business transactions.

Panthesis is currently seeking investment capital to support company expansion and market penetration, and is engaged in developing relationships with key customers in the online auction and gaming markets.

The Boeing Co., with headquarters in Chicago, is the world's leading aerospace company and the No. 1 U.S. exporter. It is the largest manufacturer of satellites, commercial jetliners and military aircraft, and it provides a full range of lifecycle support for these and other products. The company is also a global market leader in missile defense, human space flight and launch services. Boeing capabilities also include financial services and advanced information and communications systems.

Microsoft Boosts Accessibility to Internet Gaming Zone With Latest Release

PR Newswire; New York; Apr 27, 1998;

Start Page: 1
Dateline: Washington
Companies: Microsoft Corp

Abstract:

REDMOND, Wash., April 27 /PRNewswire/ -- Microsoft Corp. (Nasdaq: MSFT) today released its latest update for the Microsoft(R) Internet Gaming Zone (http://www.zone.com/), featuring support for Netscape 4.0 and the latest versions of Microsoft Internet Explorer. The new version makes the Zone accessible to the majority of Internet users. With this new version, the Zone also introduced the new Zone Rating System, which allows game players to determine how they fare against other players. Chess and Age of Empires(R) will be the first games with the Zone Rating System, and new games are scheduled to be added to the system in the coming weeks.

The Zone is a collective place for gamers to play today's best games against others for free. Players have a wide variety of games to choose from -- including parlor games like Hearts and Chess, and action and strategy games like Jedi Knight: Dark Forces II, Age of Empires and the Fighter Ace(TM) online multiplayer game, the site's first premium game designed specifically for massive multiplayer gaming via the Internet. Furthermore, visitors can navigate through the site before downloading the Zone software required for game play.

Full Text:

Industry: COMPUTER/ELECTRONICS; INTERNET MULTIMEDIA ONLINE

Netscape Support and Player Rating System Featured in Newest Version

Of the Leading Internet Gaming Site

REDMOND, Wash., April 27 /PRNewswire/ -- Microsoft Corp. (Nasdaq: MSFT) today released its latest update for the Microsoft(R) Internet Gaming Zone (http://www.zone.com/), featuring support for Netscape 4.0 and the latest versions of Microsoft Internet Explorer. The new version makes the Zone accessible to the majority of Internet users. With this new version, the Zone also introduced the new Zone Rating System, which allows game players to determine how they fare against other players. Chess and Age of Empires(R) will be the first games with the Zone Rating System, and new games are scheduled to be added to the system in the coming weeks.

"We believe online gaming is all about social interaction with a large and active community," said Ed Fries, general manager of the games group at Microsoft. "So we're very pleased that this new version of the Zone provides access for virtually everyone online."

Already home to nearly 1.5 million online gamers, the Zone has more than 7,500 simultaneous users at peak times -- and is gaining new registered members at the rate of one every 20 seconds.

The Zone is a collective place for gamers to play today's best games against others for free. Players have a wide variety of games to choose from -- including parlor games like Hearts and Chess, and action and strategy games like Jedi Knight: Dark Forces II, Age of Empires and the Fighter Ace(TM) online multiplayer game, the site's first premium game designed specifically for massive multiplayer gaming via the Internet. Furthermore, visitors can navigate through the site before downloading the Zone software required for game play.

In addition to Netscape 4.0 support and the Zone Rating System, the newest version of the Zone also features a new, streamlined interface, which reduces download times and makes getting into a game even easier. The Zone further assists its members with improved help and chat features.

Variety and Popularity of Games Drive Growth

The Zone offers a popular variety of classic card and board games such as Spades, Bridge and Backgammon. In fact, Spades has grown to become the most popular game on the Zone with peak usage of more than 2,000 players. In the past year, the Zone's lineup of CD-ROM games with free matchmaking has expanded rapidly with the addition of such popular Microsoft games as Age of Empires and Flight Simulator 98, and other top titles such as Jedi Knight: Dark Forces II from LucasArts Entertainment Co., Quake II from id Software and Scrabble from Hasbro Interactive, a unit of Hasbro Inc. These additions have brought the total number of games available for play on the Zone to 32. The Zone also recently announced support for upcoming Tom Clancy titles Rainbow Six and Dominant Species from Red Storm Entertainment.

The Internet Gaming Zone has served Internet gamers since October 1995. In May 1996, Microsoft acquired Electric Gravity Inc., the original designer of the Internet Gaming Zone. The Internet Gaming Zone offers free membership with three components: free classic card and board games, free matchmaking for retail games, and access to premium games designed exclusively for the Zone (connect-time charges may apply). Most recently, Microsoft launched Fighter Ace, a World War II aerial combat premium game designed specifically for the Internet in which more than 100 players can dogfight in a single flight arena.

Founded in 1975, Microsoft is the worldwide leader in software for personal computers. The company offers a wide range of products and services for business and personal use, each designed with the mission of making it easier and more enjoyable for people to take advantage of the full power of personal computing every day.

For online product information:

Microsoft Web site: http://www.microsoft.com/

Microsoft Internet Gaming Zone Web site: http://www.zone.com/

NOTE: Microsoft, Age of Empires and Fighter Ace are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. Other product and company names herein may be trademarks of their respective owners. SOURCE Microsoft Corp.

Microsoft Announces Launch Date for UltraCorps, Its Second Premium Title For The Internet Gaming Zone

PR Newswire; New York; May 27, 1998;

Start Page: 1
Dateline: Washington
Companies: Microsoft Corp

Abstract:

REDMOND, Wash., May 27 /PRNewswire/ -- Microsoft Corp. (Nasdaq: MSFT) today announced plans to launch UltraCorps, its second premium online-only game for the Microsoft(R) Internet Gaming Zone (http://www.zone.com/), on June 25. The game is currently in open beta testing. Players can join the free beta by going to the Zone and proceeding to the UltraCorps link in the Strategy Games section. More than 3,500 players have participated in the beta so far. Microsoft also plans to spotlight two additional premium online-only titles for the Zone, plus the latest Fighter Ace(TM) online multiplayer game upgrade, at the Electronics Entertainment Expo (E3) trade show, May 28-30 in Atlanta (Booth 4420 in West Hall, Georgia Congress Center).

UltraCorps, developed by VR-1 Inc., is a turn-based strategy game that pits thousands of players against each other for domination of the universe. Players command one of 14 alien races, develop new technologies and weapons, dispatch fleets to colonize other planets, and manage resources to maintain their growing empires. Social interaction is a key component of the game as players form alliances, draw up treaties or taunt their enemies. As a turn-based game, it is well-suited to Internet play because it can challenge thousands of players without latency issues.

"UltraCorps is a galactic game of chess that forces gamers to outthink their opponents each day when they go online," said Adam Waalkes, product unit manager for the Zone team at Microsoft. "The Zone is the perfect platform to deliver UltraCorps to gamers because the size and scope of the game is a great match for our large community of players."

Full Text:

Industry: COMPUTER/ELECTRONICS; INTERNET MULTIMEDIA ONLINE

'Oblivion,' Asheron's Call and Fighter Ace Upgrade Among Other Premium Titles

To Be Showcased at 1998 Electronics Entertainment Expo

REDMOND, Wash., May 27 /PRNewswire/ -- Microsoft Corp. (Nasdaq: MSFT) today announced plans to launch UltraCorps, its second premium online-only game for the Microsoft(R) Internet Gaming Zone (http://www.zone.com/), on June 25. The game is currently in open beta testing. Players can join the free beta by going to the Zone and proceeding to the UltraCorps link in the Strategy Games section. More than 3,500 players have participated in the beta so far. Microsoft also plans to spotlight two additional premium online-only titles for the Zone, plus the latest Fighter Ace(TM) online multiplayer game upgrade, at the Electronics Entertainment Expo (E3) trade show, May 28-30 in Atlanta (Booth 4420 in West Hall, Georgia Congress Center).

UltraCorps, developed by VR-1 Inc., is a turn-based strategy game that pits thousands of players against each other for domination of the universe. Players command one of 14 alien races, develop new technologies and weapons, dispatch fleets to colonize other planets, and manage resources to maintain their growing empires. Social interaction is a key component of the game as players form alliances, draw up treaties or taunt their enemies. As a turn-based game, it is well-suited to Internet play because it can challenge thousands of players without latency issues.

"UltraCorps is a galactic game of chess that forces gamers to outthink their opponents each day when they go online," said Adam Waalkes, product unit manager for the Zone team at Microsoft. "The Zone is the perfect platform to deliver UltraCorps to gamers because the size and scope of the game is a great match for our large community of players."

The arrival of Microsoft's second premium game on the Zone will cap its latest string of 1998 milestones, including the recent addition of support for Netscape Communicator 4.0, surpassing 1.5 million registered members, and its recent mark of more than 8,600 simultaneous users.

"Oblivion" Will Let Gamers Blow Opponents to Smithereens on the Zone

"Oblivion," current code name for a space-action premium game that is scheduled to arrive on the Zone late in 1998, combines detailed 3-D accelerated graphics, fluid motion and rich sound with the intellectual challenge of a strategy game. Players can engage hundreds of others online in territorial team wars, amid endless permutations of roles, missions and challenges. "Oblivion" is being developed by Microsoft Research.

More than 30 unique user-controlled spacecraft and space stations are modeled with lifelike textured exteriors and articulated parts. A panorama of cosmic phenomena includes planets, stars, black holes and wormholes rendered in graphic detail, accompanied by unearthly stereo sounds ranging from the din of asteroid impacts to the scream of failing force fields.

Asheron's Call: An Epic Online Adventure

Asheron's Call(TM) online multiplayer game, which is scheduled to arrive on the Zone in early 1999, draws together thousands of players within a dynamic, 3-D online world. Players can create truly unique characters, with varied combinations of visual appearance, attributes and skill sets. The setting for the game is a 24-by-24-mile island with all types of terrain, including mountain glaciers, desert wastelands, swamps and subterranean dungeons. The game immerses players in an intense fantasy role-playing environment where they must choose to compete against or cooperate with thousands of other real players. An extensive system of allegiance and influence greatly enhances social interaction. The story line in Asheron's Call evolves dynamically over time based on the decisions and actions of the Asheron's Call community. The game is being developed by Turbine Entertainment Software.

Fighter Ace Upgrade Set to Take Flight

Fighter Ace, a premium World War II aerial combat game that allows hundreds of players to dogfight simultaneously in a single arena, is scheduled to get new features later this summer. These include new terrain with greater geographic diversity; a new layout featuring airfields grouped farther apart so gamers can group and coordinate attacks; heavy bombers for flying missions against enemy installations; military, industrial and civilian ground targets; support for force-feedback joysticks; and improved anti-aircraft weapons.

Free Classic Games, Retail Matchmaking Continue

The Zone also offers free software and matchmaking for a variety of popular classic card and board games such as Spades, Bridge and Backgammon. In fact, Spades has grown to become the most popular game on the Zone, with concurrent usage at peak times of more than 2,100 players. In the past year, the Zone's lineup of CD-ROM games with free matchmaking has expanded rapidly with the addition of new

Microsoft games such as Outwars(TM) and Monster Truck Madness(R) 2 racing simulation, and other new titles such as Star Wars(R) Rebellion from LucasArts Entertainment Co., Quake II from id Software and SORRY! from Hasbro Interactive, a unit of Hasbro Inc. The lineup will continue to expand as Microsoft has recently announced relationships with Red Storm Entertainment and MicroProse Inc. to bring some of their new titles to the Zone.

Evolution of the Zone Continues

The Internet Gaming Zone has served Internet gamers since October 1995. In May 1996, Microsoft acquired Electric Gravity Inc., the original designer of the Internet Gaming Zone. The Internet Gaming Zone offers free membership with three components: free classic card and board games, free matchmaking for retail games, and access to premium games designed exclusively for the Zone (connect-time charges may apply).

Founded in 1975, Microsoft is the worldwide leader in software for personal computers. The company offers a wide range of products and services for business and personal use, each designed with the mission of making it easier and more enjoyable for people to take advantage of the full power of personal computing every day.

NOTE: Microsoft, Fighter Ace, Asheron's Call and Monster Truck Madness are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. Star Wars is a registered trademark of Lucasfilm Ltd. Outwars is a trademark of Singletrac Studio, a GT Interactive Company. Other product and company names herein may be trademarks of their respective owners.

For online product information:

Microsoft Games Web site: http://www.microsoft.com/games/ SOURCE Microsoft Corp.

DISTRIBUTED ALGORITHMS FOR SHORTEST-PATH, DEADLOCK-FREE ROUTING AND BROADCASTING IN ARBITRARILY FAULTY HYPERCUBES

Michael Peercy *Prithviraj Banerjee*

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

ABSTRACT

We present a distributed table-filling algorithm for point to point routing in a degraded hypercube system. This algorithm finds the shortest length existing path from each source to each destination in the faulty hypercube and fills the routing tables so that messages are routed along these paths. We continue with a distributed algorithm to fill tables used for broadcasting in a faulty hypercube. A novel scheme for broadcast routing with tables is proposed, and the algorithm required to fill the broadcast tables given the point to point routing tables is presented. In addition, we give the modifications necessary to make these algorithms ensure deadlock-free routing. We conclude with a quantitative and qualitative comparison of previously proposed reroute strategies with table routing, where the tables are filled with our algorithms.

1. INTRODUCTION

Message-passing multiprocessors such as hypercubes [1] consist of many processing nodes that interact by sending messages over communication channels between the nodes. However, the existence of a large number of components in such systems makes them vulnerable to failures. It is therefore extremely important to have schemes for message passing in such systems that can route messages efficiently in the presence of failures in nodes and links. This paper deals with message routing in *hypercube* networks.

Hypercubes today generally route messages using the *e-cube* routing algorithm [1]. This algorithm resolves the bit differences between the source s and the destination d from the lowest dimension to the highest and ensures the minimum length path. Numerous proposals and investigations have been made regarding routing and broadcasting in faulty hypercubes [2, 3, 4, 5, 6, 7]. Also, routing schemes which are designed to avoid network congestion can provide fault tolerant rerouting [8].

Previous schemes for routing in hypercubes have the following drawbacks. First, many of them are nonoptimal algorithms, i.e., they route massages through nonshortest paths, or fail to route messages even when paths exist. Also, algorithms that are close to optimal require very complicated algorithms whose hardware requirements are much greater than the *e-cube* routing hardware; really complicated algorithms might require microprogrammed control. Besides, the cost of the routing algorithm appears every time a message is routed.

In this paper we investigate reroute strategies based on *routing tables* [9, 10]. It should be noted that while routing tables have been proposed for loosely coupled distributed systems, they have not conventionally been used for hypercubes. The primary reason is that for fault-free hypercubes, the routing algorithms are so simple that messages can be routed optimally using minimal hardware. However, in the presence of faults, the routing algorithms become complex, and thus it is appropriate to reconsider table routing.

In distributed table routing, each node's communication coprocessor contains its own routing table. Let T_p be the routing table located at node p. T_p consists of N locations, where N is the number of processors ($N = 2^n$ in an n-dimensional hypercube). Location d of T_p, represented as $T_p[d]$, contains the dimension l for a message being routed to d to take from p. In this way a message moves from its source s to its destination d along a path (s d) derived from routing tables in each intermediate node. Ideally the path (s d) a message takes should succeed if at all possible and should be of minimum feasible length.

Note that n is the dimension of the hypercube of size $N=2^n$. We are not suggesting table routing for massively parallel programming, so the N by log$N$ size of the table should cause no concern. For instance, in a thousand processor hypercube, the required RAM is 1K by 12 (using one bit to indicate an unreachable or faulty node). Fast RAMs of this size are very inexpensive relative to the other hardware or microcode options provided by alternative fault-tolerant routing schemes. Also note that the time required for routing with tables is small and constant: the time to compute the outgoing link is the time of one memory read. Some serialization is possible among the input ports as they try to access the RAM, but, again, the RAM is fast compared to other transmission delay components. If this sequential access to the RAM is of concern, multiple copies of the routing table, or interleaving a single copy, are possible modifications.

The routing tables must be filled by some algorithm. Ideally, this algorithm would be designed to find the optimal possible paths in creating the routing tables. This algorithm needs to be run only when the configuration F of the system has changed. Researchers [11, 12, 13, 14] have presented algorithms which incrementally modify the routing tables in general networks when a change in the topology is recognized by nodes neighboring the change. Recently, Kim and Reed [15] investigated routing with tables produced by a central node using information delivered from local nodes. In this paper, we concentrate on globally designed distributed algorithms, specifically taking advantage of the hypercube topology, which entirely refill the tables after a fault or repair. Subsequently, the routing tables work

This research was supported in part by the SDIO Innovative Science and Technology Office and managed by the Office of Naval Research under Contract N00014-88-K-0624 and in part by the Joint Services Electronics Program under Contract N00014-90-J-1270.

independently, routing messages along the shortest paths until the configuration changes again and the system needs to run the table-filling algorithm once more.

In this paper we propose a distributed table-filling algorithm (TFA) which determines the routing table for each node s at node s itself. This was developed from a centralized TFA in which the system host finds shortest paths using Dijkstra's algorithm [16]. In our distributed algorithm each node gathers information about the hypercube configuration F exclusively through communication with its nearest neighbors. After presenting the distributed table-filling algorithm, we propose a broadcasting technique which utilizes tables. In this scheme, a broadcast message would carry in its header the fact that it is a broadcast and the original source of the broadcast. Each node s along the broadcast paths would then lookup in a broadcast routing table on which links the broadcast should be routed from s. We give another distributed algorithm which fills the broadcast routing tables from the original routing tables. Next we provide a method to ensure that the paths found by our table-filling algorithms are deadlock-free. By splitting links in dependency cycles into two virtual links, we can route upon the links so that no cycles exist in the new configuration, and thus avoid deadlock. We present an algorithm to modify the tables produced by the distributed table-filling algorithm so that the routes are free of the possibility of deadlock.

2. DISTRIBUTED TABLE-FILLING ALGORITHM

2.1. Distributed Algorithm

The key to a distributed table-filling algorithm (TFA) is that the shortest path from a node s to a node d is the extension of the shortest path found by one of the neighbors of s. In our TFA, each node cycles through its n neighbors, exchanging tentative routing tables, until these tables cease to change. The distributed TFA D is given below.

ALGORITHM $D(s)$ {in parallel on all nodes s}

```
Let the current dimension $l$ be $n-1$
Repeat until table unmodified in $n$ consecutive dimensions
  Exchange routing tables with neighbor along dimension $l$
  For each destination in own table
    If path through neighbor shorter than presently recorded path
    Or dimension $l$ lower than initial dimension of presently
          recorded path
      Place new path, identified as dimension $l$ and length, in table
    Endif
  Endfor
  Decrement (mod $n$) dimension $l$
Enduntil
For next $n$ dimensions
  Inform neighbor along dimension $l$ that own table is done
  Decrement (mod $n$) dimension $l$
Endfor
```

To facilitate the proof of the operation of this algorithm, we define a *sweep* as one set of consecutive iterations from dimension $n-1$ through dimension 0. That is, a sweep consists of one iteration in each dimension.

THEOREM 2.1: Algorithm D terminates with the shortest paths.

PROOF of shortest paths: By induction.

BASE CASE: All paths of length 1 (all paths to nearest neighbors) are shortest paths and are discovered in the first sweep.

ASSUME: After k sweeps every node s has all shortest paths of length k that it sources.

THEN: Because every subpath of a shortest path is itself a shortest path, a shortest path of length $k+1$ from a node s to some destination d includes a shortest path of length k from a neighbor of s to destination d. In sweep $k+1$ every node s receives the length k path information from each of its neighbors. Therefore, every shortest path of length $k+1$ sourced by each node s is determined by appending the appropriate dimension onto the shortest path of length k sourced by a neighboring node. After $k+1$ sweeps every node s has all shortest paths of length $k+1$ that it sources.

PROOF of termination: To see that algorithm D does not terminate until all shortest paths are found, consider the path sourced by node s that would be the last discovered by algorithm D; say that this path P is of length L. P necessarily contains L different paths, to different destinations, all sourcing at s. In fact, there is exactly one path of each length from 1 to L which is a (shortest) subpath of the (shortest) path P. By the induction step above, it is clear that each sweep advances the maximum length of the discovered shortest paths by 1. Therefore, in each sweep a new shortest path, which happens to be a subpath of P, is discovered by s, and algorithm D does not terminate until the last path P is found. The termination condition given in algorithm D follows when we recognize that the phase of the sweep does not matter.

Algorithm D can find more than one link of a path in each sweep. Thus the number of sweeps actually required to find all the shortest paths is a configuration-dependent value between 1 and $N-1$. The former value is for a fault-free cube and the latter value is an upper bound for a worst-case completely connected cube where the maximum length shortest path is $N-1$ links long. Thus the algorithm has a time complexity of $O(N^2 \log N)$. However, the possibility of so poor a performance is minimal, and most faulty configurations will give a time complexity much closer to that in a perfect cube: $O(N \log N)$.

D is constructed to use local information only, and builds its paths on the near end. By adding links of lowest possible dimension to the source end of its current paths, D ensures *e-cube*-like routing in a fault-free hypercube. In algorithm D we start with the highest dimension ($n-1$) and move down through the dimensions; after dimension 0 we move to dimension $n-1$ again. The reason we decrement through dimensions in D as opposed to any other order of taking dimensions is that in n iterations, that is, n exchanges of information, all nodes in a fault-free hypercube fill their routing tables with the *e-cube* paths. It is an interesting result that with only one table exchange in each dimension, every node in a fault-free cube fills its routing table perfectly. This makes the cost of implementing table-routing very small as far as filling tables in perfect cubes. However, in general faulty hypercube configurations, the tables are filled in few more iterations.

Figure 1 shows a cube with a failure in node 5, and Figure 2 shows the last 4 of the 6 steps required to fill the routing tables with the optimum paths. After the first three steps, every path which is an *e-cube* path has been identified; this set of *e-cube* paths is shown in Figure 2(a). We now note two points in Figure 2: how the shortest path is selected and how the increasing-in-

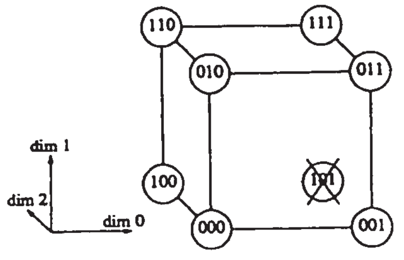


Figure 1. 3-cube with Fault in Node 5

dimension path is selected. By the third iteration, we have not yet filled $T_4[7]$ (row 4, column 7 in the routing matrix). In iteration four, TFA D places a 2 in that location. The path from node 4 to node 7 which the matrix after iteration four dictates is [4 0 1 3 7], a length 4 path. But we can see in Figure 1 that a length 2 path exists: namely [4 6 7]. This is corrected in iteration five, as we swap along dimension 1 and node 4 learns from node 6 of a shorter path to node 7. The other point to note is exemplified by $T_6[1]$, the first step on a distance 3 path. After three iterations, this location is unfilled. In each subsequent iteration, a lower first dimension of the path is found. Thus algorithm D finds the path with the lowest first dimension out of the set of shortest paths.

2.2. Extension to Partial Failures

The above description of Algorithm D handles link failures and total node failures. However, in the case of partial node failures, i.e., loss of the main processor but continuing operation of the *widowed* communication coprocessor, the TFA D so far does not operate ideally. In fact, as given above, D would be unable to use the routing table in a functioning, but widowed, coprocessor, and would view such a node as totally faulty when routing paths. Minor modifications to algorithm D correct this deficiency.

If a coprocessor's table is independently receivable from neighboring nodes, then another node can run Algorithm D for a widowed coprocessor. We modify Algorithm D to allow the *claiming* of a widowed coprocessor w by an active processor v. If there is a viable path from v to w, and w is as yet unclaimed, then v claims w. Since D forces synchronization by its very nature, at any one time w is approached only on one dimension, so the uniqueness of v is assured. After v claims w, v then executes D as though it were being executed on w (call this $D(w)$), starting with the next dimension in the iterations of the algorithm. Of course, since v must also execute its own version of D, the time it takes for v to perform each step is doubled. The claiming of widowed coprocessors can be recursive, i.e., v may need to claim w' which lies on the other side of w from v. Then, in executing $D(w')$, v must communicate through w.

The reasons the routing table must be writable from other nodes are twofold. First, messages which are sent to a claimed w must be fooled into going to v. That is, to claim w, v must write a path to itself into the location w in $T_w[w]$. Second, when the algorithm is complete and all routing tables are finalized, v must write into T_w the routing table it determined running $D(w)$.

s	d							
	0	1	2	3	4	5	6	7
0	+	0	1	0	2	.	1	0
1	0	+	0	1	0	.	0	1
2	1	0	+	0	1	.	2	0
3	0	1	0	+	0	.	0	2
4	2	-	1	-	+	.	1	-
5	.	.	.	.	.	+	.	.
6	1	-	2	0	1	.	+	0
7	0	-	0	2	0	.	0	+

2(a): After third iteration (after dims. 2, 1, 0)

s	d							
	0	1	2	3	4	5	6	7
0	+	0	1	0	2	.	1	0
1	0	+	0	1	-0	.	0	1
2	1	0	+	0	1	.	2	0
3	0	1	0	+	0	.	0	2
4	2	2	1	2	+	.	1	2
5	.	.	.	.	.	+	.	.
6	1	2	2	0	1	.	+	0
7	0	2	0	2	0	.	0	+

2(b): After fourth iteration (dim. 2)

s	d							
	0	1	2	3	4	5	6	7
0	+	0	1	0	2	.	1	0
1	0	+	0	1	0	.	0	1
2	1	0	+	0	1	.	2	0
3	0	1	0	+	0	.	0	2
4	2	2	1	1	+	.	1	1
5	.	.	.	.	.	+	.	.
6	1	1	2	0	1	.	+	0
7	0	2	0	2	0	.	0	+

2(c): After fifth iteration (dim. 1)

s	d							
	0	1	2	3	4	5	6	7
0	+	0	1	0	2	.	1	0
1	0	+	0	1	0	.	0	1
2	1	0	+	0	1	.	2	0
3	0	1	0	+	0	.	0	2
4	2	2	1	1	+	.	1	1
5	.	.	.	.	.	+	.	.
6	1	0	2	0	1	.	+	0
7	0	2	0	2	0	.	0	+

2(d): Complete routing table (after dim. 0)

Figure 2. Algorithm D on 3-cube with Fault in Node 5

3. BROADCASTING WITH TABLES

We now propose table routing methods for one-to-all broadcast. To implement broadcast, our routing table requires an additional $n+1$ bits of information per word. Let us describe these additional bits per word as a separate table U_s, with location b represented by $U_s[b]$. The broadcast algorithm to use this table is as follows: a 1 in bit l of $U_s[b]$ means that, if s receives a broadcast message which originates at b, it should copy that message and send it along dimension l. Therefore, an adequate header for a broadcast message would be an indicator that it is a broadcast and the address of the original source of the broadcast. An algorithm is required to fill the table U in each node. This algorithm executes after D and determines broadcast paths from the optimal length paths found by D. We call this broadcast table-filling algorithm (BTFA) D^B. Before giving the algorithm, we introduce the concept of the link partition.

For a node s, given an original broadcast source b and a list of destination nodes $M_s[b]$, we define the *link partition* of the set of destinations $M_s[b]$. $M_s[b]$ is the union of the disjoint sets $M_s[b]_0$, $M_s[b]_1$, $\cdots$, $M_s[b]_{n-1}$, $M_s[b]_n$, where $M_s[b]_l = \{ d : d \in M_s[b] \text{ AND } T_s[d]=l \}$. $M_s[b]_l$ is that set of destinations in $M_s[b]$ for which the first dimension of the paths from s to those destinations is l. $M_s[b]_n$ contains the single element s, the current node. The partition $M_s[b]_l$ is determined from the routing table T_s. In fact, $M_s[s]_l$ is the inverse of $T_s[d]$; the former maps l to d, and the latter maps d to l.

For example, given as row 3 in Figure 2(d), we have the routing table for node 3 in a 3-cube with a faulty node 5: T_3=[0 1 0 3 0 . 0 2], where 3=n signifies the current node and the period signifies an unreachable node. Using this routing table, $M_3[3]_3=\{3\}$, $M_3[3]_2=\{7\}$, $M_3[3]_1=\{1\}$, and $M_3[3]_0=\{0,2,4,6\}$.

$M_s[b]$ in each node s is the set of destinations to which s is expected to forward a broadcast message from b, and the link partition gives the set of destinations each neighbor of s is expected to forward. The table $U_s[b]$ will have a 1 in every bit l for which $M_s[b]_l$ is nonempty. Node s would then forward a broadcast by (1) recognizing the original source of the broadcast b, and (2) forwarding it along each and every link l for which $U_s[b]_l = 1$.

ALGORITHM $D^B(s)$ {for every node s}

```
$M_s[s] \leftarrow$ all viable destinations
$I \leftarrow \{s\}$
Determine link partition of $M_s[s]$ =
        $M_s[s]_0, M_s[s]_1, \cdots, M_s[s]_{n-1}, M_s[s]_n$
$l \leftarrow 0$
While $I \neq \emptyset$ or there are viable sources $s$ has not heard from
  Send $(i, M_s[i]_l)$, for all $i \in I$ and $M_s[i]_l \neq \emptyset$, along link $l$
  For all $i \in I$ and $M_s[i]_l \neq \emptyset$
    $U_s[i]_l \leftarrow 1$
    $M_s[i] \leftarrow M_s[i] - M_s[i]_l$
    if $M_s[i] == M_s[i]_n$ then $U_s[i]_n \leftarrow 1$; $I \leftarrow I - \{i\}$
  Endfor
  Receive $(j, M_s[j])$, for all $j \in J$ (for some set $J$), along link $l$
  For all $j \in J$
    Determine link partition of $M_s[j]$ =
        $M_s[j]_0, M_s[j]_1, \cdots, M_s[j]_{n-1}, M_s[j]_n$
    If $M_s[j] == M_s[j]_n$ then $U_s[j]_n \leftarrow 1$; $J \leftarrow J - \{j\}$
  Endfor
  $I \leftarrow I \cup J$
  $l \leftarrow l +_{\text{mod } n} 1$
Endwhile
```

THEOREM 3.1: The tables filled by algorithm D^B will broadcast using shortest paths.

PROOF: Broadcast paths are exactly those shortest paths found by algorithm D.

For simplicity of presentation, our BTFA D^B shows the broadcast table bits $U_s[b]_l$ modified with each iteration l. We could write all of $U_s[b]$ once the partition of $M_s[b]$ is done. $U_s[b]_l$=1 if and only if $M_s[b]_l \neq \emptyset$. We use the n^{th} partition set and the n^{th} bit of the broadcast table as a convenience to imply that any broadcast message forwarded by node s should be received and absorbed by node s as well. Note also that, in subsequent steps of the algorithm, the determination of the link partition of newly received sets can be computed for each l from the initial link partition as $M_s[j]_l = M_s[j] \cap M_s[s]_l$.

BTFA $D^B(s)$ is very efficient. The amount of work involved in the send, receive, and partition steps is proportional to the length of the lists. The algorithm's complexity is $O(N^2 \log N)$, but, as with algorithm D, this order is reached only at degenerate worst cases. On a perfect cube the algorithm runs in time $O(N \log N)$, only executing one iteration for each dimension.

Figure 3 shows an example of the operation of this algorithm on node 0 of a fault-free 3-cube. Each horizontal block in Figure 3(a) is one iteration of BTFA D^B. The first block is the initial state, with all destinations reachable from node 0 partitioned by the first links in their respective paths. In each iteration k, node 0 sends along link $l=k$ mod n the lists of all destinations the paths to which node 0 routes on link l. Then the current partitions are modified to show the removal of the just-sent lists, and newly received lists are partitioned and included as current. When the list of nodes in a current partition b includes only node 0, signifying that all other nodes have been taken care of, then the partition is removed from further consideration.

We also give an example of D^B executing on a faulty cube. Recall the single-fault hypercube of Figure 1; in this 3-cube, node 5 is faulty. Row 3 of Figure 2(d) is T_3, the table used in computing the initial link partition. Figure 4 shows the operation on D^B from the viewpoint of node 3 and the broadcasting table at node 3 which results. To illustrate the basic rule behind the operation of D^B, we describe what happens when node 3 receives (6, {1,3}). This information tells D^B that node 6 expects its broadcasts to reach nodes 1 and 3 through node 3. Node 3 then determines how *it* reaches nodes 1 and 3. It sends to node 1, according to the routing table, using dimension 1. Thus the algorithm waits until dimension 1 is dictated by execution, and sends (6, {1}), telling the neighbor along dimension 1 that node 6 expects to communicate with node 1 through that neighbor. Node 6 expects to communicate also with node 3 through node 3, but that path is trivial and no further computation is necessary.

Executing an all-to-all broadcast, in which each node sends the same message to every other node, could be accomplished in one of two ways. The messages could be broadcast independently and asynchronously, mutually contending for limited link resources, or the messages could be broadcast synchronously. Specifically for the synchronous case, when an all-to-all broadcast is required, the nodes could execute a variant of algorithm D^B. Every node would thus communicate along the same

	send		receive		current partitions $M_0[i]_l$				
k	i	$M_0[i]_k$	j	$M_0[j]$	i	$l=3$	$l=2$	$l=1$	$l=0$
-					0	0	4	2,6	1,3,5,7
0	0	{1,3,5,7}			0	0	4	2,6	.
			1	{0,2,4,6}	1	0	4	2,6	.
1	0	{2,6}			0	0	4	.	.
	1	{2,6}			1	0	4	.	.
			2	{0,4}	2	0	4	.	.
			3	{0,4}	3	0	4	.	.
2	0	{4}			0	0	.	.	.
	1	{4}			1	0	.	.	.
	2	{4}			2	0	.	.	.
	3	{4}			3	0	.	.	.
			4	{0}	4	0	.	.	.
			5	{0}	5	0	.	.	.
			6	{0}	6	0	.	.	.
			7	{0}	7	0	.	.	.

3(a): Operation of D^B at node 0 in 3-cube

broadcast routing table $U_0[b]_l$				
b	$l=3$	$l=2$	$l=1$	$l=0$
0	1	1	1	1
1	1	1	1	0
2	1	1	0	0
3	1	1	0	0
4	1	0	0	0
5	1	0	0	0
6	1	0	0	0
7	1	0	0	0

3(b): Broadcast routing table for node 0

Figure 3. Example of Algorithm D^B on Fault-Free 3-cube

k	send		receive		current partitions $M_3[i]_l$				
mod n	i	$M_0[i]_k$	j	$M_3[j]$	i	$l=3$	$l=2$	$l=1$	$l=0$
-					3	3	7	1	0,2,4,6
0			2	{1,3,7}	2	3	7	1	.
	3	{0,2,4,6}			3	3	7	1	.
1			0	{3,7}	0	3	7	.	.
			1	{3,7}	1	3	7	.	.
	2	{1}			2	3	7	.	.
	3	{1}			3	3	7	.	.
2	0	{7}			0	3	.	.	.
	1	{7}			1	3	.	.	.
	2	{7}			2	3	.	.	.
	3	{7}			3	3	.	.	.
			6	{1,3}	6	3	.	1	.
			7	{1,3}	7	3	.	1	.
0					6	3	.	1	.
					7	3	.	1	.
1	6	{1}			6	3	.	.	.
	7	{1}			7	3	.	.	.
2			4	{3}	4	3	.	.	.

4(a): Operation of D^B at node 3 in 3-cube with faulty node 5

broadcast routing table $U_3[b]_l$				
b	$l=3$	$l=2$	$l=1$	$l=0$
0	1	1	0	0
1	1	1	0	0
2	1	1	1	0
3	1	1	1	1
4	1	0	0	0
5	0	0	0	0
6	1	0	1	0
7	1	0	1	0

4(b): Broadcast routing table for node 3

Figure 4. Example of Algorithm D^B on Single-Fault 3-cube

dimension at the same time a composite message of individual broadcasts. Since in fact algorithm D^B is an all-to-all broadcast of dynamic messages (the destination lists), a synchronous all-to-all broadcast would take exactly as many steps as D^B. The broadcast routing tables filled by D^B would serve either the synchronous or asynchronous all-to-all broadcast.

4. DEADLOCK AVOIDANCE

The standard routing algorithm *e-cube* is the primary algorithm for routing messages in hypercubes today. Three principal reasons explain this preference for *e-cube*: (1) it is easy to implement, (2) it spreads messages evenly throughout the network, and (3) it prevents deadlock. The prevention of deadlock can be assured if and only if there are no cycles in the channel dependency graph [17]. The reason that no cycles exist in the *e-cube* algorithm is that every channel is dependent only on channels of higher dimension. No dependency can go backwards in dimension. Thus deadlock is impossible in *e-cube*.

However, in a hypercube containing faulty links (channels), extra precautions must be taken to ensure deadlock-free routing. We adapt the method given in [17] to avoid deadlock. Essentially this method consists of defining virtual channels along the physical links. Each virtual channel is distinguished from the others on one link by a unique address and its own queue. The virtual channels can be time multiplexed on the physical links with the use of these queues. By maintaining a strict ordering of these virtual channels, we can show that the new channel dependency graph is free of cycles, and thus the network is free of deadlock.

As an example of the configuring of virtual channels to avoid deadlock, we show Figures 5 and 6. Figure 5 gives a configuration of a hypercube with directed links (one each way between processors) which has a possible deadlock configuration. The links which may cause deadlock are extracted from Figure 4 and given with explicit unidirectionality in Figure 6(a). We call such a set of links in a hypercube a *loop*. A loop contains two cycles, one in each direction on the loop.

We represent the possibility of deadlock with the channel dependency graph of Figure 6(b). The vertices of this graph are the links from Figure 6(a); the edges represent the (nontransitive) dependencies. The vertices are labeled with a unique link label. Each link is identified by an ordered pair (s,l), where s is the

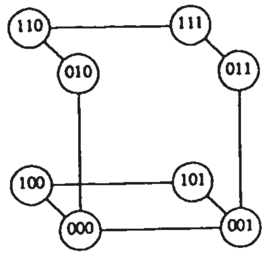


Figure 5. Faulty Configuration of 3-cube Inducing Cycles

node sourcing the link and l is the dimension of the link. For example, the link from node 0 to node 2 is represented in the right cycle of Figure 6(b) by the vertex (0,1).

To prevent deadlock, we split each of the links in a cycle into two virtual links which share the same physical communication line but have different queues (Figure 6(c)). Then we address the virtual links in the cycle with labels of the form xyz, where $x \in \{0,1\}$, y increases around the cycle, and z is a unique cycle identifier. Thus we can break the cycle by permitting dependencies only in increasing order of the virtual link addresses (Figure 6(d)). To force the dependencies to be acyclic, we give each processor node in each cycle the label yz, where the node sources virtual links $0yz$ and $1yz$, and enforce the following message routing rule at each source or intermediate node: if the current node label is less than the destination node label, route along the higher addressed link; if the current node label is greater than the destination node label, route along the lower addressed link. Note that, in our example, links $15a$ and $15b$ are not used.

In general, after running a TFA we have routing tables for which the dependency graph contains cycles. The problem facing us is that these routing tables are distributed among the nodes, and it would be very inefficient to detect cycles from the local tables. However, we can globally broadcast all the routing tables so that each processor has the complete routing matrix. This could be a large amount of communication, but the following theorem and corollary allows us to reduce it.

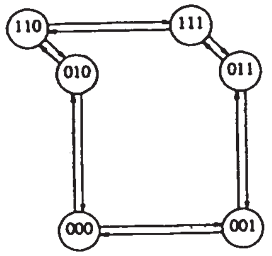
THEOREM 4.1: For any path found by TFA D for configuration F, all subpaths of that path are themselves paths found by TFA D.

PROOF: Follows from the way paths are determined during routing.

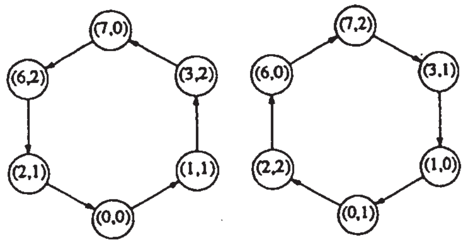
THEOREM 4.2: The dependence graph for a routing matrix found by TFA D can be constructed with information on paths of length 2 only.

PROOF: Since every path (i.e., every string of channel dependencies) is composed of paths of length 2, the paths of length 2 capture all the consecutive dependencies. The transitive dependencies can be ignored in finding cycles.

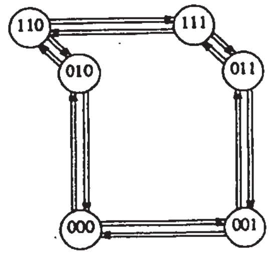
We only need to communicate the paths of length 2 throughout the network to provide full channel dependency information. Paths of length 2 can be derived from an abridged routing matrix which contains source-destination pairs no more than distance 2 from each other. Thus each node need only communicate its table for distance 1 and distance 2 destinations. There are



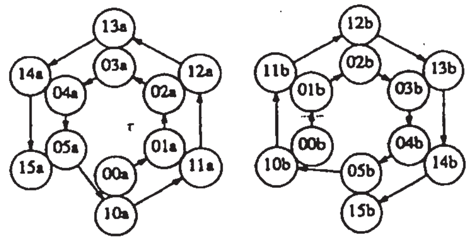
6(a): Loop extracted from faulty cube



6(b): Channel dependency graph of cycles



6(c): Virtual links to break cycles



6(d): Dependency graph of virtual links

Figure 6. Example of Breaking Cycles in Faulty 3-cube

$C_1^n + C_2^n$ of these in each node, where C_k^n denotes the number of ways to choose k items from n items.

Once each node has complete information of the total routing matrix, it can construct the channel dependency graph and find all cycles. An algorithm such as that given in [16] is used to find the cycles. Every node has the same information and, if each runs the same algorithm, each finds the same cycles. Then, by splitting each cycle into a spiral of virtual channels, the nodes remove dependencies from the graph. The nodes then modify their routing tables so that, given a destination, they indicate the correct virtual link to reach that destination without the possibility of deadlock.

We have not yet considered the impact of our cycle removal schemes on paths which deviate from the cycles. For example, in Figure 5 node 2 routes to node 5 along two links of the counterclockwise cycle included in the path [2 0 1 5]. We need to correctly identify which link (higher or lower) we should take to reach each destination. The correct link will be dependent on the *last* intermediate node in the cycle that the path routes through to reach its destination. From information of paths of length 2, we cannot construct each longer path, we cannot determine the last node for each path in the intersection of path and cycle, and we therefore can not tell from paths of length 2 whether to route each path along the higher or the lower virtual link in a cycle. We can correct this problem by passing complete routing tables along cycles, so that every source knows exactly how far along the cycle every path goes. The cycle address of the last node in the path-cycle intersection determines whether the higher or lower link is taken along the cycle.

Below is Algorithm DEADLOCK_FREE, which modifies the routing tables found by D to ensure the avoidance of deadlock in path selection. The hardware and encoding in the routing architecture at each node s must be altered to permit the addressing of multiple virtual links per physical link. In the presentation of DEADLOCK_FREE below, we simply show the routing table getting the virtual link address, i.e., $T_s[d] \leftarrow xz$. (We suppress the y from our notation xyz because the y implicitly refers to the current node s.)

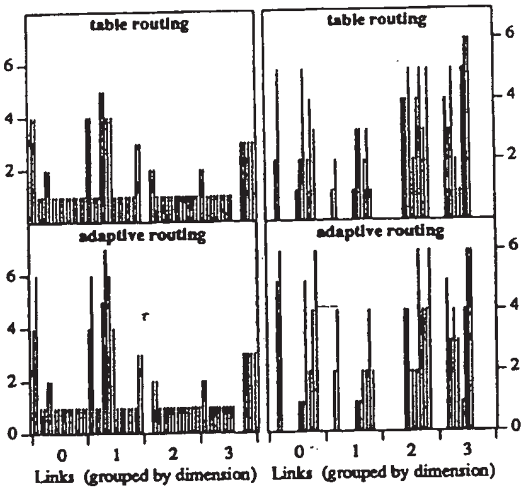
```
ALGORITHM DEADLOCK_FREE (in each node $s$)

Run algorithm $D$
Run algorithm $D^B$
Do all-to-all broadcast of routing table contents for distance
        1 and 2
Construct channel dependency graph
Find all cycles using cycle detection algorithm
For each cycle $z$ found which includes an outgoing link of $s$
  Create two virtual channels $0z$ and $1z$ to replace instance of
        link in $z$
Allocate and address one queue for each outgoing virtual channel
Exchange complete routing tables around each cycle to determine
        complete paths along each cycle
For each destination $d$ with path $(s\ d)$
  If $T_s[d]$ is in some cycles
    Choose a cycle $z$ which intersects $(s\ d)$ along the greatest
        length
    Let $p$ be the last node in the intersection of the path and $z$
    If the label of $p$ in cycle $z$ is less than that of $s$
        (denoted $y$ in text)
      $T_s[d] \leftarrow 0z$
    Else
      $T_s[d] \leftarrow 1z$
    Endif
  Endif
Endfor
```

Two questions which arise are the following. Do we need to alter the broadcast routing tables? Will any part of algorithm DEADLOCK_FREE induce deadlock before the avoidance techniques are in place? The answer to both these questions is found in the single statement: deadlock cannot involve a single-link path. Deadlock involves a path acquiring one link and holding it while it awaits another. In single-link paths, once the first link is acquired, no waiting need be done: the path is complete, the message is sent, and the link is freed. Both algorithms D and D^B operate synchronously on single-link paths. Broadcast, also, generally occurs in single-link paths. If we wished to allow broadcasts to operate on multiple-length paths, we could simply rerun D^B after DEADLOCK_FREE, this time partitioning the destinations, and the broadcast table, among the virtual links; the rest of the algorithm D^B is unchanged.

5. PERFORMANCE OF TABLE ROUTING

We now compare the performance of table-routing under TFA D with another proposed reroute scheme, the adaptive scheme of Chen and Shin [2]. Their reroute method, which we will here refer to as *adaptive*, finds a path from source to destination by starting an *e-cube* path, then altering it as necessary when it is blocked by a fault. A tag is used to mark blocked and extra dimensions to prevent oscillation.

We compare these with two measures of performance applicable to reconfigured networks [18]. The reconfiguration strategy we use is that of process adoption [19]: an adjacent processor adopts the task running on a processor after it fails. The



7(a): Dilation 7(b): Congestion

Figure 7. Comparison of Dilation and Congestion

first measure, called *dilation*, gives length in links of the logical replacement of a previously physical link. That is, a path between two adjacent processes in a fault-free cube may be mapped to a multiple-link path due to fault and subsequent reconfiguration. The second measure is called *congestion*. This measures the number of logical links which use each fault-free physical link in the faulty configuration.

Our example configuration F is one with four faulty nodes: node 0, node 5, node 6, and node 15. The process from node 0 is mapped to node 4, process 5 is mapped to node 13, process 6 to node 3, and process 15 to node 10. The results are shown in Figure 7.

Both the dilation and congestion measurements are a constant 1 across all links in a fault-free hypercube; every logical link is on exactly one physical link, and every physical link carries exactly one logical link. However, in our faulty hypercube example, with node 4 very far from other nodes in the network, the dilation and congestion measurements are quite high. The areas of the dilation and congestion histograms are equal; this area is essentially the number of physical links all the logical links use. The area for this four-fault hypercube is 104 with the routes determined by algorithm *D* and 112 with the *adaptive* routing scheme, demonstrating the reduced system communication load due to the shorter paths of table routing.

6. CONCLUSIONS

We have introduced table routing in faulty hypercubes, demonstrating the power and ease of such a routing method. Our distributed algorithms have shown table routing to be not only possible, but preferable in faulty hypercubes. Our distributed table-filling algorithm *D* executes in $O(N^3 \log N)$ time in the very rare worst case. Generally, performance of *D* is of the order of $N^2 \log N$. We have also proposed the use of tables for broadcast in faulty hypercubes. The broadcast table-filling algorithm runs in $O(N^2 \log N)$ worst case time, with a general performance around $N \log N$.

We have shown the superior dilation and congestion measures of the shortest paths generated by *D* in faulty hypercubes and the minimal extra hardware and communication delay of table routing. Also we have presented a deadlock prevention scheme applied to distributed routing tables. To our knowledge, this is the first routing scheme that has been proposed for faulty hypercubes that is shortest-path and deadlock-free.

REFERENCES

[1] H. Sullivan and T. R. Bashkow, "A large scale homogeneous fully distributed parallel machine," *Proc. 4th Symp. Computer Architecture*, pp. 105-117, Mar. 1977.

[2] M. S. Chen and K. G. Shin, "Message routing in an injured hypercube," *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, pp. 312-317, Jan. 1988.

[3] T. C. Lee and J. P. Hayes, "Routing and broadcasting in faulty hypercube computers," *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, pp. 346-354, Jan. 1988.

[4] J. M. Gordon and Q. F. Stout, "Hypercube message routing in the presence of faults," *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, pp. 318-327, Jan. 1988.

[5] M. S. Chen and K. G. Shin, "Routing in the presence of an arbitrary number of faults in hypercube multicomputers," *4th Conf. Hypercube Concurrent Computers and Applications*, Mar. 1989.

[6] Kasho, et. al., "Distributed fault tolerant routing in hypercubes," *4th Conf. Hypercube Concurrent Computers and Applications*, Mar. 1989.

[7] Al-Dhelaan and Bose, "Efficient fault-tolerant broadcasting algorithm for the hypercube," *4th Conf. Hypercube Concurrent Computers and Applications*, Mar. 1989.

[8] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, "Hyperswitch network for the hypercube computer," *Proc. 15th Int. Symp. Computer Architecture*, pp. 90-99, May 1988.

[9] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Passing Parallel Processing*. Cambridge, MA: MIT Press, 1987.

[10] A. S. Tanenbaum, *Computer Networks*. Englewoods Cliffs, NJ: Prentice-Hall, Inc., 1981.

[11] W. D. Tajibnapis, "A correctness proof of a topology information maintenance protocol for a distributed computer network," *Commun. of the ACM*, vol. 20, pp. 477-485, July 1977.

[12] Gallager R. G., "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, vol. COM-25, pp. 73-85, Jan. 1977.

[13] Segall A., "Advances in verifiable fail-safe routing procedures," *IEEE Transactions on Communications*, vol. COM-29, pp. 491-497, Apr. 1981.

[14] E. M. Gafni and D. P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on Communications*, vol. COM-29, pp. 11-18, Jan. 1981.

[15] C. Kim and D. A. Reed, "Adaptive packet routing in a hypercube," *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, pp. 625-629, Jan. 1988.

[16] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Englewoods Cliffs, NJ: Prentice-Hall, Inc., 1974.

[17] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547-553, May 1987.

[18] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a hypercube in the presence of faults," *Proc. 19th ACM Symp. Theory of Computing*, pp. 274-284, 1987.

[19] P. Banerjee, "Reconfiguring a hypercube in the presence of faults," *Proc. 4th Conf. Hypercube Concurrent Computers and Applications*, Mar. 1989.

Your search matched **46** of **972916** documents.

A maximum of **46** results are displayed, **25** to a page, sorted by **Relevance** in **descending** order.
You may refine your search by editing the current search expression or entering a new one the text box.
Then click **Search Again**.

((flood* or broadcast*) <near/3> routing and ((hyper Search Again

Results:
Journal or Magazine = **JNL** Conference = **CNF** Standard = **STD**

1 **Distributed algorithms for shortest-path, deadlock-free routing and broadcasting in arbitrarily faulty hypercubes**
Peercy, M.; Banerjee, P.;
Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium , 26-28 June 1990
Page(s): 218 -225

[Abstract] [PDF Full-Text (652 KB)] **IEEE CNF**

2 **Multi-level hypercube network**
Aboelaze, M.A.;
Parallel Processing Symposium, 1991. Proceedings., Fifth International , 30 April-2 May 1991
Page(s): 475 -480

[Abstract] [PDF Full-Text (428 KB)] **IEEE CNF**

3 **Cross-cube: a new fault tolerant hypercube-based network**
Haq, E.;
Parallel Processing Symposium, 1991. Proceedings., Fifth International , 30 April-2 May 1991
Page(s): 471 -474

[Abstract] [PDF Full-Text (280 KB)] **IEEE CNF**

4 **Distributed algorithms for shortest-path, deadlock-free routing and broadcasting in Fibonacci cubes**

A DISTRIBUTED RESTORATION ALGORITHM FOR MULTIPLE-LINK AND NODE FAILURES OF TRANSPORT NETWORKS

Hiroaki Komine, Takafumi Chujo, Takao Ogura, Keiji Miyazaki, and Tetsuo Soejima

Fujitsu Laboratories, Ltd.
1015 Kamikodanaka, Nakahara-ku, Kawasaki, 211, Japan

Abstract

Broadband optical fiber networks will require fast restoration from multiple-link and node failures as well as single-link failures. This paper describes a new distributed restoration algorithm based on message flooding. The algorithm is an extension of our previously proposed algorithm for single-link failure. It restores the network from multiple-link and node failures, using multi-destination flooding and path route monitoring. We evaluated the algorithm by computer simulation, and verified that it can find alternate paths within 0.5s whenever the message processing delay at a node is 5ms.

1. Introduction

There is an increasing dependency on today's communication networks to implement strategic corporate functions. User demands for high-speed and economical communications services lead to the rapid deployment of high-capacity optical fibers in the transport networks. At the same time, the demands for high-reliability services raise a network survivability problem. For example, if the network is disabled for one hour, up to $6,000,000 loss of revenue can occur in the trading and investment banking industries [1]. As the capacity of the transmission link grows, a link cut results in more loss of services. Therefore, rapid restoration from failures is becoming more critical for network operations and management.

There have been many algorithms developed to restore networks, including centralized control [1] and distributed algorithms [2-4]. In centralized control, the network is controlled and managed from a central office. In distributed control, the processing load is distributed among the nodes and restoration is thus faster. However, more computation capability and high speed control data channels are required. Recently it has been possible to provide high performance microprocessors for digital cross-connect system (DCS). High capacity optical fibers enable high speed data transmission for OAM through overhead bytes, which is under study by CCITT.

The distributed algorithms proposed so far [2-4] are based on simple flooding [5]. When a node detects failure, it broadcasts a restoration message to adjacent nodes to find an alternate route. In the algorithm [2], a restoration message requests a spare DS-3 or STS-1 path and is sent through the path overhead of each spare path. To avoid congestion of the messages in this algorithm, a message in both the algorithms [3,4] requests a bundle of spare paths and is sent through the section overhead of each link. Algorithm [3] finds the maximum capacity along an alternate route, and our algorithm [4] finds the shortest alternate route. As described in [4], our algorithm was faster. However these algorithms are designed to handle single-link failures, they cannot handle multiple-link or node failures.

In this paper, we first discuss the major issues that must be addressed in order to handle multiple-link and node failures in Section 2. Based on these consideration, we propose a new restoration algorithm using multi-destination flooding and path route monitoring. These are described in Section 3. For a node failure, the node which detected the failure sends a restoration message to the last N-consecutive nodes each logical path passed through. An alternate path is made between the message sender node and one of the multiple nodes specified in the message. Each node collects the identifier of these nodes, using a path route monitoring technique. The algorithm was evaluated by computer simulation for multiple-link failure as well as for node failure. The results will be described in Section 4.

2. Limitations of simple flooding

In this section, we review simple flooding and discuss its limitations to handle multiple-link and node failures. In principle, the distributed algorithms [2-4] based on simple flooding work as follows. When a link fails, the two nodes connected to the link detect the failure and try to restore the path. One node becomes the sender and the other becomes the chooser (Fig. 1). The sender broadcasts restoration messages to all links with spare capacity. Every node except the sender and the chooser respond by re-broadcasting the message. When the restoration message reaches the chooser, the chooser returns an acknowledgement to the sender. In this way, alternate paths are found. Message congestion caused by routing messages far away is avoided by limiting the number of hops.

These algorithms based on simple flooding [2-4] usually assume a single-link failure, but in reality, some links which go different nodes may be in the same conduit. Therefore, if the conduit is cut, many links fail at the same time [3]. This is the case of multiple-link failure. Fire or earthquakes can also damage a large number of nodes, so the restoration algorithm must be able to handle these situations.

Simple flooding can not handle multiple-link or node failures because of following problems.

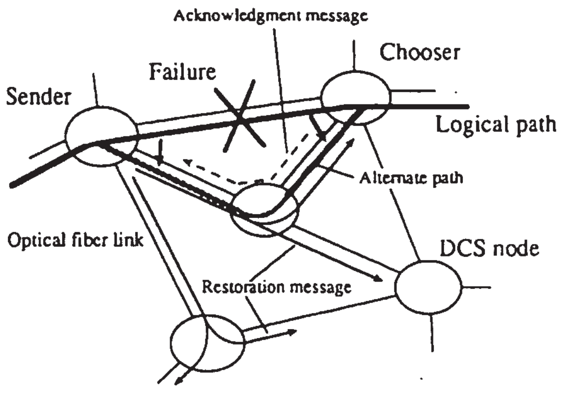


Fig. 1 Distributed restoration based on simple flooding

- Contention of spare capacity

In case of multiple-link failure, restoration messages coming from different nodes might contend for spare capacity on the same link. For example, if capacity is assigned to arriving messages in turn, the first message reserves the capacity. Whether or not the reserved capacity is later used for an alternate path, the reserved capacity is not released and therefore can not be assigned to another restoration message. Thus, the restoration ratio decreases.

- Fault location

Because the algorithms assume link failure, one of the two nodes connected to the failed link becomes the sender and the other becomes the chooser. However, for a node failure, there is a chooser and sender for each affected path. They are neighbors of the failed node and depend on the route of the paths. Each node detects failure by the loss of the signal on the link, and cannot distinguish between link or node failure.

The first problem could be alleviated by simple message cancelling. Spare capacity is assigned to restoration messages on a first-come, first-served basis. Assignment is cancelled when the message can not go forward due to hop limits or lack of capacity. During message flooding, cancel messages are sent to inform a node that a restoration message, which reserves spare capacity on a specific link, did not reach its destination and the served capacity of this link can be released for other restoration messages. Restoration messages are canceled immediately after reception if they are identical to messages already received, if the hop limit is reached, or if there is no more capacity at the node. In these cases, the unused capacity can be assigned to another restoration message.

Solving the second problem requires more sophisticated techniques and we propose a new distributed restoration algorithm in the following section.

3. Multi-destination flooding

To solve the fault location problem described above, we propose a new multi-destination flooding technique. We also propose path route monitoring which is essential to achieve multi-destination flooding.

3.1 Principle of multi-destination flooding

Simple flooding methods assume just one chooser. We extended this to allow multiple choosers as message destinations. When a node detects the loss of a signal from a link, the node can not tell whether the link or the node at the other end has failed. It sends a restoration message directed to the node which is the chooser in a link failure as well those that are choosers in a node failure. In Fig.2, for example, the link between nodes B and C fails, node B is the chooser for all affected paths, and nodes A and D are possible choosers for paths P1 and P2. If node B fails, nodes A and D become choosers for paths P1 and P2. The restoration message contains all choosers and the required capacity for each sender-chooser pair. The node which received the restoration message checks the destination field of the message, and if it is a chooser candidate, it returns an acknowledgment to the sender.

Thus, by extending simple flooding into multi-destination flooding, link or node failures do not have to be distinguished because there is always at least one chooser. Different messages are sent to the chooser candidates, but the same restoration message listing all candidates is sent towards all candidates. The number of restoration messages decreases and congestion is reduced.

Restoration processing consists of a broadcast phase, an acknowledgment phase, and a confirmation phase. To handle multiple failures, cancel processing is performed during the broadcast and acknowledgment phases.

The node states are sender, chooser, reserved tandem, and fixed tandem. The sender is the node which detected the failure. The chooser is the destination node of a restoration message. Chooser candidates set by the sender become choosers when they receive

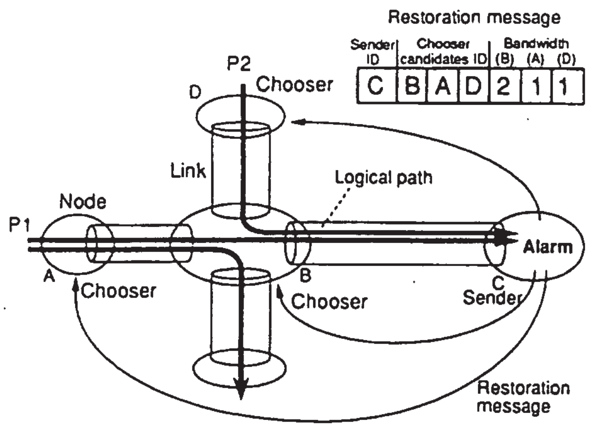


Fig. 2 Multi-destination flooding

a restoration message. The reserved tandem is a candidate node for alternate paths reserved by the restoration message. A received confirmation message of the sender turns a reserved tandem node into a fixed tandem node.

a) Broadcast phase

In the broadcast phase, the sender broadcasts restoration messages which reserve spare capacity in the network toward chooser candidates. A failure occurring on a link or node is detected by the next node on the path below the failure. This node becomes the sender. The sender looks up the chooser candidates and their capacities for the failed paths which were determined before by the path route monitoring described in the following section. The restoration message is then broadcast.

The restoration message contains the following information.

1) Message type : restoration, acknowledgment, confirmation, cancel
2) Message index
3) Sender ID
4) Chooser IDs (Multiple destination)
5) Required capacity of each sender-chooser pair
6) Reserved capacity
7) Hop count

The message index is set by the sender. It represents the number of flooding waves broadcast. The combination of the message index, the sender ID and chooser IDs is the Message ID. The required capacity is the capacity required between the sender and the various choosers. The reserved capacity is the capacity of the route taken by the restoration message.

The sender broadcasts the restoration message to all connected links except failed links and then waits for an acknowledgment from one of the choosers. Each node in the network except the sender and chooser receives a restoration message, and examines the hop count and the Message ID. If the hop count reaches the limit set by the sender, or a message with the same ID has arrived before, the node returns a cancel message to the link originating

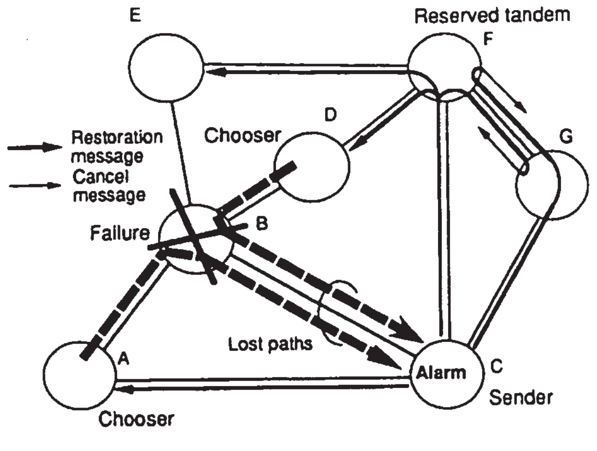


Fig. 3 Broadcast phase

the restoration message. Otherwise, the state of the node is set to reserved tandem. If spare capacity is available, a restoration message is broadcast. If the spare capacity of a link is insufficient, the reserved capacity is set to the spare capacity of the link. A node that finds its own node ID among the chooser IDs in the restoration message becomes the chooser. Figure 3 shows the broadcast phase when a failure has occurred at node B.

b) Acknowledgment phase

In the acknowledgment phase, the chooser sends an acknowledgment message to the sender. By the entries in the acknowledgment message, the sender is informed which chooser the acknowledgement message is from. If another restoration message with the same message ID arrives at the chooser, it is canceled.

A reserved tandem node which receives an acknowledgment message passes it back to the source of the corresponding restoration message. All other reserved spare capacity of this restoration message is canceled. Message flow during an acknowledgment phase is shown in Fig. 4.

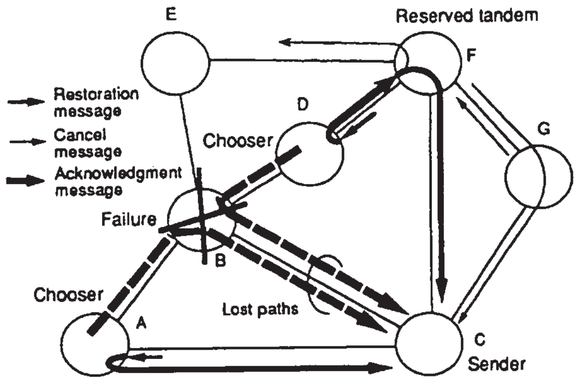


Fig. 4 Acknowledgment phase

c) Confirmation phase

When the acknowledgment message reaches the sender, a confirmation message is sent to the chooser. The reserved spares are switched over to alternate paths. If the sender received acknowledgment or canceled messages from all links it sent restoration messages to, and if the restoration of the failure is not completed, the sender increments the message index and attempts restoration from the broadcast phase again.

The reserved tandem node which received a confirmation message changes its status to fixed tandem and connects the reserved spares. In Fig. 5, node F has become fixed tandem, and the failed path between node D and node C is rerouted through the nodes D, F, and C. The other path which failed between node A and node C are also rerouted.

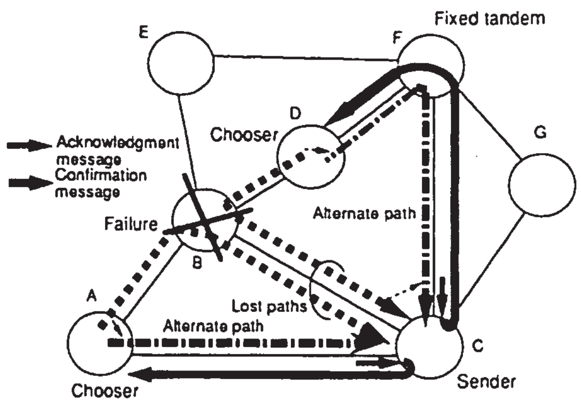


Fig. 5 Confirmation phase

3.2 Path route monitoring

For multi-destination flooding, each node must have route information on the paths passing through the node. One approach is to have the central office distribute such route information to all nodes. However, the routes are changing dynamically under customer control and nodes might receive inconsistent route information because updating route data takes time. We propose a path route monitoring method in which each node collects route information in real time.

The route information required at every node are the ID's of the last two consecutive nodes in every path before the node. This information is collected as follows. Node ID's are sent through assigned space in the path overhead. For every path going through a node, the data in the ID area is shifted and the ID of the node it is going through is written in. In this way, every node receives continuous and real-time route information.

4. Simulation

4.1 Simulation tool and conditions

We evaluated the ability of the algorithm to restore multiple-link and node failures using an event-driven network simulator [4,6] which works on the SUN3 workstation. We used the mesh network model shown in Fig. 6. This network consists of 25 nodes and 40 links. Each link length was generated at random, and the average link length is 184 km. Every link has 35 working paths.

We assumed a transmission speed of 64 kb/s. Messages were 16 bytes long, and the hop limit was 9. In a SONET frame structure, 64 kb/s for transmission speed means that one byte of overhead is used for message communications between nodes. The processing delay time from the arrival of a message to the end of the processing depends on the architecture of the DCS hardware. We assumed a 5 ms delay. This simulation does not include failure detection or crossconnection times.

4.2 Simulation results

Figure 7 shows a cumulative restoration ratio of node failure. The restoration ratio of the network is the ratio of restored to lost paths. For node failure, paths terminating at the failed node are not counted as lost paths because it is impossible to restore them. We also simulated the algorithm for single-link failure. The result is shown in Fig. 7.

Figure 8 shows the cumulative restoration ratio in a multiple-link failure. There are many link combinations, but only one is shown. Failures between node N8 and N13, and one of the other links, occured simultaneously on two links. The results indicate

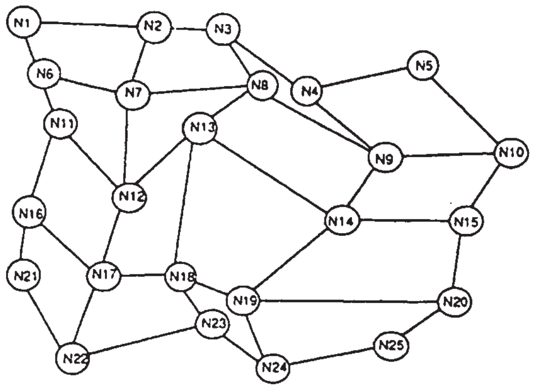


Fig. 6 Network model

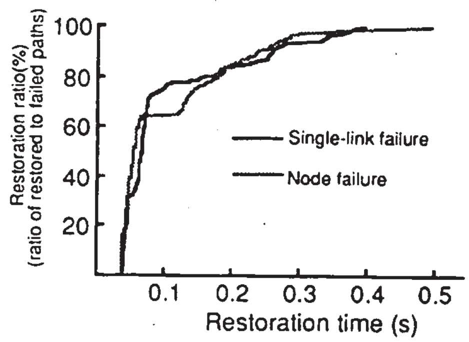


Fig. 7 Simulation results on single-link and node failure

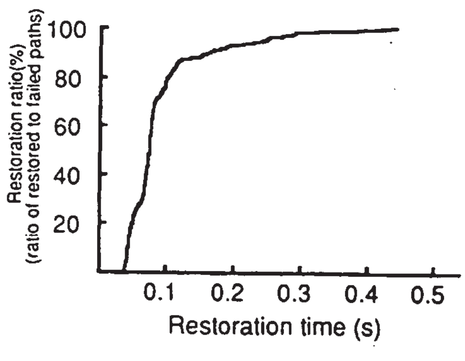


Fig. 8 Simulation result on multiple-link failure

that the proposed algorithm can handle multiple-link and node failure as well as single-link failure. All restorations are completed within 0.5s with message processing delay at the nodes being 5ms.

5. Conclusion

We pointed out problems associated with adapting a restoration algorithm based on flooding to recover from multiple-link and node failures. The main problem is to position the chooser nodes correctly. We proposed multi-destination flooding and path route monitoring. We simulated the algorithm with a mesh network and verified that the algorithm can handle multiple-link and node failures as well as single-link failures.

The message delay within a node depends on the architecture of the DCS and the processing load. The next step will be to analyze these delays and to include restoration time.

Acknowledgment

The authors thank Dr. Takanashi, Dr. Murano, and Mr. Yamaguchi of Fujitsu Laboratories Ltd., and Mr. Tokimasa of Fujitsu Ltd. for their encouragement and advice.

References

[1] W. Falconer, "Services Assurance in Modern Telecommunications Networks," IEEE Communications Magazine, Vol. 28, No. 6, pp. 32-39, June 1990.

[2] W. D. Grover, "The Selfhealing™ Network: A FAST DISTRIBUTED RESTORATION TECHNIQUE FOR NETWORKS USING DIGITAL CROSSCONNECT MACHINES", Globecom'87, pp. 28.2.1-28.2.6, Nov. 1987.

[3] C. H. Yang and S. Hasegawa, "FITNESS: Failure Immunization Technology for Network Service Survivability", Globecom'88, pp. 47.3.1-47.3.6, Dec. 1988.

[4] T. Chujo, T. Soejima, H. Komine, K. Miyazaki, and T. Ogura, "The Design and Simulation of an Intelligent Transport Network with Distributed Control", NOMS'90, pp. 11.4-1 - 11.4-12, Feb. 1990.

[5] A. S. Tanenbaum, "Computer Networks", pp. 298-299, Prentice-Hall International, 1988.

[6] T. Chujo, T. Soejima, H. Komine, K. Miyazaki, and T. Ogura, "The Modeling and Simulation of an Intelligent Transport Network with Distributed Control", ITU-COM'89, VII.1, pp. 343 - 347, Oct. 1989.

A distributed restoration algorithm for multiple-link and node failures of transport networks

Komine, H. Chujo, T. Ogura, T. Miyazaki, K. Soejima, T.
Fujitsu Lab. Ltd., Kawasaki, Japan ;

This paper appears in: **Global Telecommunications Conference, 1990, and Exhibition. 'Communications: Connecting the Future', GLOBECOM '90., IEEE**

Meeting Date: 12/02/1990 - 12/05/1990
Publication Date: 2-5 Dec. 1990
Location: San Diego, CA USA
On page(s): 459 - 463 vol.1
Reference Cited: 6
Inspec Accession Number: 3976310

Abstract:
Fast restoration of broadband optical fiber networks from multiple-link and node failu as well as single-link failures, is addressed. A **distributed restoration algorithm** b on message flooding is described. The algorithm is an extension of a previously prop algorithm for single-link failure. It restores the network from multiple-link and node failures, using multidestination flooding and path route monitoring. Computer simula of the algorithm verified that it can find alternate paths within 0.5 s, whenever the message processing delay at a node is 5 ms

Index Terms:
broadband networks optical links broadband optical fiber networks **distributed restoration algorithm** message flooding message processing delay multidestination flooding multiple-li failures node failures path route monitoring single-link failures transport networks

Documents that cite this document
Select link to view other documents in the database that cite this one.

Search Results [PDF FULL-TEXT 364 KB] PREV NEXT DOWNLOAD CITATION

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ| Terms | Back to Top

On Four-Connecting a Triconnected Graph[†] (Extended Abstract)

Tsan-sheng Hsu
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188
tshsu@cs.utexas.edu

Abstract

We consider the problem of finding a smallest set of edges whose addition four-connects a triconnected graph. This is a fundamental graph-theoretic problem that has applications in designing reliable networks.

We present an $O(n\alpha(m,n)+m)$ time sequential algorithm for four-connecting an undirected graph G that is triconnected by adding the smallest number of edges, where n and m are the number of vertices and edges in G, respectively, and $\alpha(m,n)$ is the inverse Ackermann's function.

In deriving our algorithm, we present a new lower bound for the number of edges needed to four-connect a triconnected graph. The form of this lower bound is different from the form of the lower bound known for biconnectivity augmentation and triconnectivity augmentation. Our new lower bound applies for arbitrary k, and gives a tighter lower bound than the one known earlier for the number of edges needed to k-connect a $(k-1)$-connected graph. For $k=4$, we show that this lower bound is tight by giving an efficient algorithm for finding a set of edges with the required size whose addition four-connects a triconnected graph.

1 Introduction

The problem of augmenting a graph to reach a certain connectivity requirement by adding edges has important applications in network reliability [6, 14, 28] and fault-tolerant computing. One version of the augmentation problem is to augment the input graph to reach a given connectivity requirement by adding a smallest set of edges. We refer to this problem as the *smallest augmentation* problem.

Vertex-Connectivity Augmentations
The following results are known for solving the smallest augmentation problem on an undirected graph to satisfy a vertex-connectivity requirement.

For finding a smallest biconnectivity augmentation, Eswaran & Tarjan [3] gave a lower bound on the smallest number of edges for biconnectivity augmentation and proved that the lower bound can be achieved. Rosenthal & Goldner [26] developed a linear time sequential algorithm for finding a smallest augmentation to biconnect a graph; however, the algorithm in [26] contains an error. Hsu & Ramachandran [11] gave a corrected linear time sequential algorithm. An $O(\log^2 n)$ time parallel algorithm on an EREW PRAM using a linear number of processors for finding a smallest augmentation to biconnect an undirected graph was also given in Hsu & Ramachandran [11], where n is the number of vertices in the input graph. (For more on the PRAM model and PRAM algorithms, see [21].)

For finding a smallest triconnectivity augmentation, Watanabe & Nakamura [33, 35] gave an $O(n(n+m)^2)$ time sequential algorithm for a graph with n vertices and m edges. Hsu & Ramachandran [10, 12] developed a linear time algorithm and an $O(\log^2 n)$ time EREW parallel algorithm using a linear number of processors for this problem. We have been informed that independently, Jordan [15] gave a linear time algorithm for optimally triconnecting a biconnected graph.

For finding a smallest k-connectivity augmentation, for an arbitrary k, there is no polynomial time algorithm known for finding a smallest augmentation to k-connect a graph, for $k>3$. There is also no efficient parallel algorithm known for finding a smallest augmentation to k-connect any nontrivial graph, for $k>3$.

[†]This work was supported in part by NSF Grant CCR-90-23059.

The above results are for augmenting undirected graphs. For augmenting directed graphs, Masuzawa, Hagihara & Tokura [23] gave an optimal-time sequential algorithm for finding a smallest augmentation to k-connect a rooted directed tree, for an arbitrary k. We are unaware of any results for finding a smallest augmentation to k-connect any nontrivial directed graph other than a rooted directed tree, for $k > 1$.

Other related results on finding smallest vertex-connectivity augmentations are stated in [4, 19].

Edge-Connectivity Augmentations
For the problem of finding a smallest augmentation for a graph to reach a given edge connectivity property, several polynomial time algorithms and efficient parallel algorithms are known. These results can be found in [1, 3, 4, 5, 8, 9, 13, 16, 19, 24, 27, 30, 31, 34, 37].

Augmenting a Weighted Graph
Another version of the problem is to augment a graph, with a weight assigned to each edge, to meet a connectivity requirement using a set of edges with a minimum total cost. Several related problems have been proved to be NP-complete. These results can be found in [3, 5, 7, 20, 22, 32, 33, 36].

Our Result
In this paper, we describe a sequential algorithm for optimally four-connecting a triconnected graph. We first present a lower bound for the number of edges that must be added in order to reach four-connectivity. Note that lower bounds different from the one we give here are known for the number of edges needed to biconnect a connected graph [3] and to triconnect a biconnected graph [10]. It turns out that in both these cases, we can always augment the graph using exactly the number of edges specified in this above lower bound [3, 10]. However, an extension of this type of lower bound for four-connecting a triconnected graph does not always give us the exact number of edges needed [15, 17]. (For details and examples, see Section 3.)

We present a new type of lower bound that equals the exact number of edges needed to four-connect a triconnected graph. By using our new lower bound, we derive an $O(n\alpha(m,n)+m)$ time sequential algorithm for finding a smallest set of edges whose addition four-connects a triconnected graph with n vertices and m edges, where $\alpha(m,n)$ is the inverse Ackermann's function. Our new lower bound applies for arbitrary k, and gives a tighter lower bound than the one known earlier for the number of edges needed to k-connect a $(k-1)$-connected graph. The new lower bound and the algorithm described here may lead to a better understanding of the problem of optimally k-connecting a $(k-1)$-connected graph, for an arbitrary k.

2 Definitions

We give definitions used in this paper.

Vertex-Connectivity
A graph[1] G with at least $k+1$ vertices is ***k*-connected**, $k \geq 2$, if and only if G is a complete graph with $k+1$ vertices or the removal of any set of vertices of cardinality less than k does not disconnect G. The *vertex-connectivity* of G is k if G is k-connected, but not $(k+1)$-connected. Let $\mathcal{U}$ be a minimal set of vertices such that the resulting graph obtained from G by removing $\mathcal{U}$ is not connected. The set of vertices $\mathcal{U}$ is a *separating k-set*. If $|\mathcal{U}| = 3$, it is a *separating triplet*. The *degree* of a separating k-set S, $d(S)$, in a k-connected graph G is the number of connected components in the graph obtained from G by removing S. Note that the degree of any separating k-set is ≥ 2.

Wheel and Flower
A set of separating triplets with one common vertex c is called a *wheel* in [18]. A wheel can be represented by the set of vertices $\{c\} \cup \{s_0, s_1, \ldots, s_{q-1}\}$ which satisfies the following conditions: (*i*) $q > 2$; (*ii*) $\forall i \neq j$, $\{c, s_i, s_j\}$ is a separating triplet except in the case that $j = ((i+1) \bmod q)$ and (s_i, s_j) is an edge in G; (*iii*) c is adjacent to a vertex in each of the connected components created by removing any of the separating triplets in the wheel; (*iv*) $\forall j \neq (i+1) \bmod q$, $\{c, s_i, s_j\}$ is a degree-2 separating triplet. The vertex c is the *center* of the wheel [18]. For more details, see [18].

The *degree* of a wheel $W = \{c\} \cup \{s_0, s_1, \ldots, s_{q-1}\}$, $d(W)$, is the number of connected components in $G - \{c, s_0, \ldots, s_{q-1}\}$ plus the number of degree-3 vertices in $\{s_0, s_1, \ldots, s_{q-1}\}$ that are adjacent to c. The degree of a wheel must be at least 3. Note that the number of degree-3 vertices in $\{s_0, s_1, \ldots, s_{q-1}\}$ that are adjacent to c is equal to the number of separating triplets in $\{(c, s_i, s_{(i+2) \bmod q}) \mid 0 \leq i < q$, such that $s_{(i+1) \bmod q}$ is degree 3 in $G\}$. An example is shown in Figure 1.

A separating triplet with degree > 2 or not in a wheel is called a *flower* in [18]. Note that it is possible that two flowers of degree-2 $f_1 = \{a_{1,i} \mid 1 \leq i \leq 3\}$ and $f_2 = \{a_{2,i} \mid 1 \leq i \leq 3\}$ have the property that $\forall i$, $1 \leq i \leq 3$, either $a_{1,i} = a_{2,i}$ or $(a_{1,i}, a_{2,i})$ is an edge in G. We denote $f_1 \mathcal{R} f_2$ if f_1 and f_2 satisfy the above

[1]Graphs refer to undirected graphs throughout this paper unless specified otherwise.

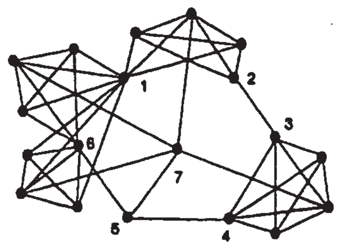


Figure 1: Illustrating a wheel $\{7\} \cup \{1,2,3,4,5,6\}$. The degree of this wheel is 5, i.e. the number of components we got after removing the wheel is 4 and there is one vertex (vertex 5) in the wheel with degree 3.

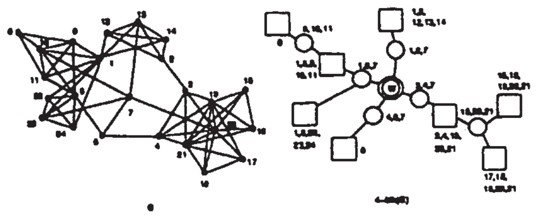


Figure 2: Illustrating a triconnected graph and its 4-$blk(G)$. We use rectangles, circles and two concentric circles to represent R-vertices, F-vertices and W-vertices, respectively. The vertex-numbers beside each vertex in 4-$blk(G)$ represent the set of vertices corresponding to this vertex.

condition. For each flower f, the *flower cluster* $\mathcal{F}_f$ for f is the set of flowers $\{f_1, \dots, f_x\}$ (including f) such that $f\mathcal{R}f_i$, $\forall i$, $1 \le i \le x$.

Each of the separating triplets in a triconnected graph G is either represented by a flower or is in a wheel. We can construct an $O(n)$-space representation for all separating triplets (i.e. flowers and wheels) in a triconnected graph with n vertices and m edges in $O(n\alpha(m,n)+m)$ time [18].

K-Block
Let $G = (V, E)$ be a graph with vertex-connectivity $k-1$. A ***k-block*** in G is either (i) a minimal set of vertices B in a separating $(k-1)$-set with exactly $k-1$ neighbors in $V \setminus B$ (these are ***special k-blocks***) or (ii) a maximal set of vertices B such that there are at least k vertex-disjoint paths in G between any two vertices in B (these are ***non-special k-blocks***). Note that a set consisting of a single vertex of degree $k-1$ in G is a k-block. A ***k-block leaf*** in G is a k-block B_l with exactly $k-1$ neighbors in $V \setminus B_l$. Note also that every special k-block is a k-block leaf. If there is any special 4-block in a separating triplet $\mathcal{S}$, $d(\mathcal{S}) \le 3$. Given a non-special k-block B leaf, the vertices in B that are not in the flower cluster that separates B are ***demanding vertices***. We let every vertex in a special 4-block leaf be a demanding vertex.

Claim 1 *Every non-special k-block leaf contains at least one demanding vertex.* □

Using procedures in [18], we can find all of the 4-block leaves in a triconnected graph with n vertices and m edges in $O(n\alpha(m,n)+m)$ time.

Four-Block Tree
From [18] we know that we can decompose vertices in a triconnected graph into the following 3 types: (i) 4-blocks; (ii) wheels; (iii) separating triplets that are not in a wheel. We modify the decomposition tree in [18] to derive the ***four-block tree*** 4-$blk(G)$ for a triconnected graph G as follows. We create an R-vertex for each 4-block that is not special (i.e. not in a separating set or in the center of a wheel), an F-vertex for each separating triplet that is not in a wheel, and a W-vertex for each wheel. For each wheel $W = \{c\} \cup \{s_0, s_1, \dots, s_{q-1}\}$, we also create the following vertices. An F-vertex is created for each separating triplet of the form $\{c, s_i, s_{(i+1) \bmod q}\}$ in W. An R-vertex is created for every degree-3 vertex s in $\{s_0, s_1, \dots, s_{q-1}\}$ that is adjacent to c and an F-vertex is created for the three vertices that are adjacent to s. There is an edge between an F-vertex f and an R-vertex r if each vertex in the separating triplet corresponding to f is either in the 4-block H_r corresponding to r or adjacent to a vertex in H_r. There is an edge between an F-vertex f and a W-vertex w if the the wheel corresponding to w contains the separating triplet corresponding to f. A ***dummy R-vertex*** is created and adjacent to each pair of flowers f_1 and f_2 with the properties that f_1 and f_2 are not already connected and either $f_1 \in \mathcal{F}_{f_2}$, $f_2 \in \mathcal{F}_{f_1}$ (i.e. their flower clusters contain each other) or their corresponding separating triplets are overlapped. An example of a 4-block tree is shown in Figure 2.

Note that a degree-1 R-vertex in 4-$blk(G)$ corresponds to a 4-block leaf, but the reverse is not necessarily true, since we do not represent some special 4-block leaves and all degree-3 vertices that are centers of wheels in 4-$blk(G)$. A special 4-block leaf $\{v\}$, where v is a vertex, is represented by an R-vertex in 4-$blk(G)$ if v is not the center of a wheel w and it is in one of separating triplets of w. The degree of a flower F in G is the degree of its corresponding vertex in 4-$blk(G)$. Note also that the degree of a wheel W in

G is equal to the number of components in 4-$blk(G)$ by removing its corresponding W-vertex w and all F-vertices that are adjacent to w. A wheel W in G is a *star wheel* if $d(W)$ equals the number of leaves in 4-$blk(G)$ and every special 4-block leaf in W is either adjacent to or equal to the center. A star wheel W with the center c has the property that every 4-block leaf in G (not including $\{c\}$ if it is a 4-block leaf) can be separated from G by a separating triplet containing the center c. If G contains a star wheel W, then W is the only wheel in G. Note also that the degree of a wheel is less than or equal to the degree of its center in G.

K-connectivity Augmentation Number
The *k-connectivity augmentation number* for a graph G is the smallest number of edges that must be added to G in order to k-connect G.

3 A Lower Bound for the Four-Connectivity Augmentation Number

In this section, we first give a simple lower bound for the four-connectivity augmentation number that is similar to the ones for biconnectivity augmentation [3] and triconnectivity augmentation [10]. We show that this above lower bound is not always equal to the four-connectivity augmentation number [15, 17]. We then give a modified lower bound. This new lower bound turns out to be the exact number of edges that we must add to reach four-connectivity (see proofs in Section 4). Finally, we show relations between the two lower bounds.

3.1 A Simple Lower Bound

Given a graph G with vertex-connectivity $k-1$, it is well known that $\max\{\lceil \frac{l_k}{2} \rceil, d-1\}$ is a lower bound for the k-connectivity augmentation number where l_k is the number of k-block leaves in G and d is the maximum degree among all separating $(k-1)$-sets in G [3]. It is also well known that for $k=2$ and 3, this lower bound equals the k-connectivity augmentation number [3, 10]. For $k=4$, however, several researchers [15, 17] have observed that this value is not always equal to the four-connectivity augmentation number. Examples are given in Figure 3. Figure 3.(1) is from [15] and Figure 3.(2) is from [17]. Note that if we apply the above lower bound in each of the three graphs in Figure 3, the values we obtain for Figures 3.(1),

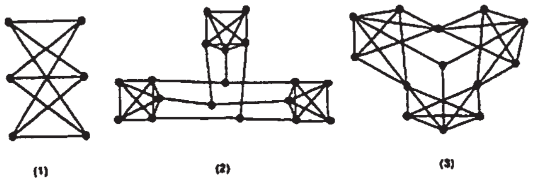


Figure 3: Illustrating three graphs where in each case the value derived by applying a simple lower bound does not equal its four-connectivity augmentation number.

3.(2) and 3.(3) are 3, 3 and 2, respectively, while we need one more edge in each graph to four-connect it.

3.2 A Better Lower Bound

Notice that in the previous lower bound, for every separating triplet S in the triconnected graph $G = \{V, E\}$, we must add at least $d(S)-1$ edges between vertices in $V \setminus S$ to four-connect G, where $d(S)$ is the degree of S (i.e. the number of connected components in $G-S$); otherwise, S remains a separating triplet. Let the set of edges added be $\mathcal{A}_{1,S}$. We also notice that we must add at least one edge into every 4-block leaf B to four-connect G; otherwise, B remains a 4-block leaf. Since it is possible that S contains some 4-block leaves, we need to know the minimum number of edges needed to eliminate all 4-block leaves inside S. Let the set of edges added be $\mathcal{A}_{2,S}$. We know that $\mathcal{A}_{1,S} \cap \mathcal{A}_{2,S} = \emptyset$. The previous lower bound gives a bound on the cardinality of $\mathcal{A}_{1,S}$, but not that of $\mathcal{A}_{2,S}$. In the following paragraph, we define a quantity to measure the cardinality of $\mathcal{A}_{2,S}$.

Let Q_S be the set of special 4-block leaves that are in the separating triplet S of a triconnected graph G. Two 4-block leaves B_1 and B_2 are *adjacent* if there is an edge in G between every demanding vertex in B_1 and every demanding vertex in B_2. We create an *augmenting graph for S*, $\mathcal{G}(S)$, as follows. For each special 4-block leaf in Q_S, we create a vertex in $\mathcal{G}(S)$. There is an edge between two vertices v_1 and v_2 in $\mathcal{G}(S)$ if their corresponding 4-blocks are adjacent. Let $\overline{\mathcal{G}(S)}$ be the complement graph of $\mathcal{G}(S)$. The seven types of augmenting graphs and their complement graphs are illustrated in Figure 4.

Definition 1 *The* **augmenting number** $a(S)$ *for a separating triplet S in a triconnected graph is the number of edges in a maximum matching $\mathcal{M}$ of $\overline{\mathcal{G}(S)}$ plus the number of vertices that have no edges in $\mathcal{M}$ incident on them.*

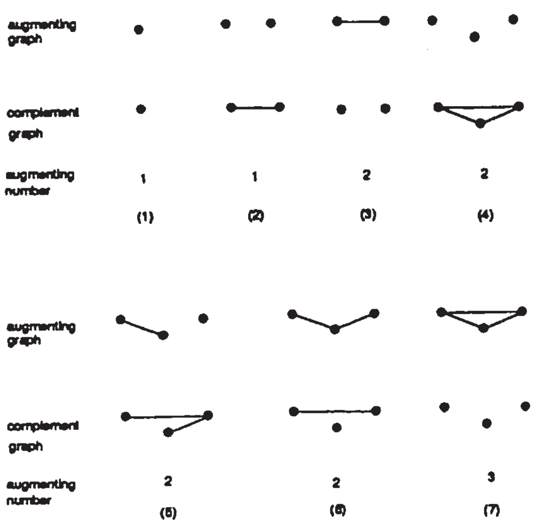


Figure 4: Illustrating the seven types of augmenting graphs, their complement graphs and augmenting numbers that one can get for a separating triplet in a triconnected graph.

The augmenting numbers for the seven types of augmenting graphs are shown in Figure 4. Note that in a triconnected graph, each special 4-block leaf must receive at least one new incoming edge in order to four-connect the input graph. The augmenting number $a(S)$ is exactly the minimum number of edges needed in the separating triplet S in order to four-connect the input graph. The augmenting number of a separating set that does not contain any special 4-block leaf is 0. Note also that we can define the *augmenting number* $a(C)$ for a set C that consists of the center of a wheel using a similar approach. Note that $a(C) \leq 1$.

We need the following definition.

Definition 2 *Let G be a triconnected graph with l 4-block leaves. The* **leaf constraint** *of G, $lc(G)$, is $\lceil \frac{l}{2} \rceil$. The* **degree constraint** *of a separating triplet S in G, $dc(S)$, is $d(S) - 1 + a(S)$, where $d(S)$ is the degree of S and $a(S)$ is the augmenting number of S. The degree constraint of G, $dc(G)$, is the maximum degree constraint among all separating triplets in G. The* **wheel constraint** *of a star wheel W with center c in G, $wc(W)$, is $\lceil \frac{d(W)}{2} \rceil + a(\{c\})$, where $d(W)$ is the degree of W and $a(\{c\})$ is the augmenting number of $\{c\}$. The wheel constraint of G, $wc(G)$, is 0 if there is no star wheel in G; otherwise it is the wheel constraint of the star wheel in G.*

We now give a better lower bound on the 4-connectivity augmentation number for a triconnected graph.

Lemma 1 *We need at least* $\max\{lc(G), dc(G), wc(G)\}$ *edges to four-connect a triconnected graph G.*

<u>**Proof:**</u> Let $\mathcal{A}$ be a set of edges such that $G' = G \cup \mathcal{A}$ is four-connected. For each 4-block leaf B in G, we need one new incoming edge to a vertex in B; otherwise B is still a 4-block leaf in G'. This gives the first component of the lower bound.

For each separating triplet S in G, $G - S$ contains $d(S)$ connected components. We need to add at least $d(S) - 1$ edges between vertices in $G - S$, otherwise S is still a separating triplet in G'. In addition to that, we need to add at least $a(S)$ edges such that at least one of the two end points of each new edge is in S; otherwise S contains a special 4-block leaf. This gives the second term of the lower bound.

Given the star wheel W with the center c, 4-*blk*(G) contains exactly $d(W)$ degree-1 R-vertices. Thus we need to add at least $\lceil \frac{d(W)}{2} \rceil$ edges between vertices in $G - \{c\}$; otherwise, G' contains some 4-block leaves. In addition to that, we need to add $a(\{c\})$ non-self-loop edges such that at least one of the two end points of each new edge is in $\{c\}$; otherwise $\{c\}$ is still a special 4-block leaf. This gives the third term of the lower bound. □

3.3 A Comparison of the Two Lower Bounds

We first observe the following relation between the wheel constraint and the leaf constraint. Note that if there exists a star wheel W with degree $d(W)$, there are exactly $d(W)$ 4-block leaves in G if the center is not degree-3. If the center of the star wheel is degree-3, then there are exactly $d(W) + 1$ 4-block leaves in G. Thus the wheel constraint is greater than the leaf constraint if and only if the star wheel has a degree-3 center. We know that the degree of any wheel is less than or equal to the degree of its center. Thus the value of the above lower bound equals 3.

We state the following claims for the relations between the degree constraint of a separating triplet and the leaf constraint.

Claim 2 *Let S be a separating triplet with degree $d(S)$ and h special 4-block leaves. Then there are at least $h + d(S)$ 4-block leaves in G.* □

Claim 3 *Let $\{a_1, a_2, a_3\}$ be a separating triplet in a triconnected graph G. Then a_i, $1 \leq i \leq 3$, is incident on a vertex in every connected component in $G - \{a_1, a_2, a_3\}$.* □

Corollary 1 *The degree of a separating triplet S is no more than the largest degree among all vertices in S.* □

From Corollary 1, we know that it is not possible that a triconnected graph has type (6) or type (7) of the augmenting graphs as shown in Figure 4, since the degree of their underling separating triplet is 1. We also know that the degree of a separating triplet with a special 4-block leaf is at most 3 and at least 2. Thus $dc(S)$ is greater than $d(S) - 1$ if $dc(S)$ equals either 3 or 4. Thus we have the following lemma.

Lemma 2 *Let $low_1(G)$ be the lower bound given in Section 3.1 for a triconnected graph G and let $low_2(G)$ be the lower bound given in Lemma 1 in Section 3.2. (i) $low_1(G) = low_2(G)$ if $low_2(G) \notin \{3,4\}$. (ii) $low_2(G) - low_1(G) \in \{0,1\}$.* □

Thus the simple lower bound extended from biconnectivity and triconnectivity is in fact a good approximation for the four-connectivity augmentation number.

4 Finding a Smallest Four-Connectivity Augmentation for a Triconnected Graph

We first explore properties of the 4-block tree that we will use in this section to develop an algorithm for finding a smallest 4-connectivity augmentation. Then we describe our algorithm. Graphs discussed in this section are triconnected unless specified otherwise.

4.1 Properties of the Four-Block Tree

Massive Vertex, Critical Vertex and Balanced Graph

A separating triplet S in a graph G is *massive* if $dc(S) > lc(G)$. A separating triplet S in a graph G is *critical* if $dc(S) = lc(G)$. A graph G is *balanced* if there is no massive separating triplet in G. If G is balanced, then its 4-$blk(G)$ is also *balanced*. The following lemma and corollary state the number of massive and critical vertices in 4-$blk(G)$.

Lemma 3 *Let S_1, S_2 and S_3 be any three separating triplets in G such that there is no special 4-block in $S_i \cap S_j$, $1 \le i < j \le 3$. $\sum_{i=1}^{3} dc(S_i) \le l + 1$, where l is the number of 4-block leaves in G.*

Proof: G is triconnected. We can modify 4-$blk(G)$ in the following way such that the number of leaves in the resulting tree equals l and the degree of an F-node f equals its degree constraint plus 1 if f corresponds to S_i, $1 \le i \le 3$. For each W-vertex w with a degree-3 center c, we create an R-vertex r_c for c, an F-vertex f_c for the three vertices that are adjacent to c in G. We add edges (w,f_c) and (f_c,r_c). Thus r_c is a leaf. For each F-vertex whose corresponding separating triplet S contains h special 4-block leaves, we attach $a(S)$ subtrees with a total number of h leaves with the constraint that any special 4-block that is in more than one separating triplet will be added only once (to the F-node corresponding to S_i, $1 \le i \le 3$, if possible). From Figure 4 we know that the number of special 4-block leaves in any separating triplet is greater than or equal to its augmenting number. Thus the above addition of subtrees can be done. Let 4-$blk(G)'$ be the resulting graph. Thus the number of leaves in 4-$blk(G)'$ is l. Let f be an F-node in 4-$blk(G)'$ whose corresponding separating triplet is S. We know that the degree of f equals $dc(S)+1$ if $S \in \{S_i \mid 1 \le i \le 3\}$. It is easy to verify that the sum of degrees of any three internal vertices in a tree is less than or equal to 4 plus the number of leaves in a tree. □

Corollary 2 *Let G be a graph with more than two non-special 4-block leaves. (i) There is at most one massive F-vertex in 4-$blk(G)$. (ii) If there is a massive F-vertex, there is no critical F-vertex. (iii) There are at most two critical F-vertices in 4-$blk(G)$.* □

Updating the Four-Block Tree

Let v_i be a demanding vertex or a vertex in a special 4-block leaf, $i \in \{1,2\}$. Let B_i be the 4-block leaf that contains v_i, $i \in \{1,2\}$. Let b_i, $i \in \{1,2\}$, be the vertex in 4-$blk(G)$ such that if v_i is a demanding vertex, then b_i is an R-vertex whose corresponding 4-block contains v_i; if v_i is in a special 4-block leaf in a flower, then b_i is the F-vertex whose corresponding separating triplet contains v_i; if v_i is the center of a wheel w, b_i is the F-vertex that is closet to $b_{(i \bmod 2)+1}$ and is adjacent to w. The vertex b_i is the *implied vertex* for B_i, $i \in \{1,2\}$. The *implied path P between B_1 and B_2* is the path in 4-$blk(G)$ between b_1 and b_2. Given 4-$blk(G)$ and an edge (v_1,v_2) not in G, we can obtain 4-$blk(G \cup \{(v_1,v_2)\})$ by performing local updating operations on P. For details, see [18].

In summary, all 4-blocks corresponding to R-vertices in P are collapsed into a single 4-block. Edges in P are deleted. F-vertices in P are connected to the new R-vertex created. We *crack* wheels in a way that is similar to the cracking of a polygon for updating 3-block graphs (see [2, 10] for details). We say that P is *non-adjacent* on a wheel W, if the cracking of W creates two new wheels. Note that it is possible that a separating triplet S in the original graph is no longer a separating triplet in the resulting graph by adding an edge. Thus some special leaves in the original graph are no longer special, in which case they must be added to 4-$blk(G)$.

Reducing the Degree Constraint of a Separating Triplet
We know that the degree constraint of a separating triplet can be reduced by at most 1 by adding a new edge. From results in [18], we know that we can reduce the degree constraint of a separating triplet S by adding an edge between two non-special 4-block leaves B_1 and B_2 such that the path in 4-$blk(G)$ between the two vertices corresponding to B_1 and B_2 passes through the vertex corresponding to S. We also notice the following corollary from the definitions of 4-$blk(G)$ and the degree constraint.

Corollary 3 *Let S be a separating triplet that contains a special 4-block leaf. (i) We can reduce $dc(S)$ by 1 by adding an edge between two special 4-block leaves B_1 and B_2 in S such that B_1 and B_2 are not adjacent. (ii) If we add an edge between a special 4-block leaf in S and a 4-block leaf B not in S, the degree constraint of every separating triplet corresponding to an internal vertex in the path of 4-$blk(G)$ between vertices corresponding to S and B is reduced by 1.* □

Reducing the Number of Four-Block Leaves
We now consider the conditions under which the adding of an edge reduces the leaf constraint $lc(G)$ by 1. Let *real degree* of an F-node in 4-$blk(G)$ be 1 plus the degree constraint of its corresponding separating triplet. The real degree of a W-node with a degree-3 center in G is 1 plus its degree in 4-$blk(G)$. The real degree of any other node is equal to its degree in 4-$blk(G)$.

Definition 3 (The Leaf-Connecting Condition) *Let B_1 and B_2 be two non-adjacent 4-block leaves in G. Let P be the implied path between B_1 and B_2 in 4-$blk(G)$. Two 4-block leaves B_1 and B_2 satisfy the* **leaf-connecting condition** *if at least one of the following conditions is true. (i) There are at least two vertices of real degree at least 3 in P. (ii) There is at least one R-vertex of degree at least 4 in P. (iii) The path P is non-adjacent on a W-vertex in P. (iv) There is an internal vertex of real degree at least 3 in P and at least one of the 4-block leaves in $\{B_1, B_2\}$ is special. (v) B_1 and B_2 are both special and they do not share the same set of neighbors.*

Lemma 4 *Let B_1 and B_2 be two 4-block leaves in G that satisfy the leaf-connecting condition. We can find vertices v_i in B_i, $i \in \{1,2\}$, such that $lc(G \cup \{(v_1, v_2)\}) = lc(G) - 1$, if $lc(G) \geq 2$.* □

4.2 The Algorithm

We now describe an algorithm for finding a smallest augmentation to four-connect a triconnected graph. Let $\delta = dc(G) - lc(G)$. The algorithm first adds 2δ edges to the graph such that the resulting graph is balanced and the lower bound is reduced by 2δ. If $lc(G) \neq 2$ or $wc(G) \neq 3$, there is no star wheel with a degree-3 center. We add an edge such that the degree constraint $dc(G)$ is reduced by 1 and the number of 4-block leaves is reduced by 2. Since there is no star wheel with a degree-3 center, $wc(G)$ is also reduced by 1 if $wc(G) = lc(G)$. The resulting graph stays balanced each time we add an edge and the lower bound given in Lemma 1 is reduced by 1. If $lc(G) = 2$ and $wc(G) = 3$, then there exists a star wheel with a degree-3 center. We reduce $wc(G)$ by 1 by adding an edge between the degree-3 center and a demanding vertex of a 4-block leaf. Since $lc(G) = 2$ and $wc(G) = 3$, $dc(G)$ is at most 2. Thus the lower bound can be reduced by 1 by adding an edge. We keep adding an edge at a time such that the lower bound given in Lemma 1 is reduced by 1. Thus we can find a smallest augmentation to four-connect a triconnected graph. We now describe our algorithm.

The Input Graph is not Balanced
We use an approach that is similar to the one used in biconnectivity and triconnectivity augmentations to balance the input graph [10, 11, 26]. Given a tree T and a vertex v in T, a *v-chain* [26] is a component in $T - \{v\}$ without any vertex of degree more than 2. The leaf of T in each v-chain is a *v-chain leaf* [26]. Let $\delta = dc(G) - lc(G)$ for a unbalanced graph G and let 4-$blk(G)'$ be the modified 4-block tree given in the proof of Lemma 3. Let f be a massive F-vertex. We can show that either there are at least $2\delta + 2$ f-chains in 4-$blk(G)'$ (i.e. f is the only massive F-vertex) or we can eliminate all massive F-vertices by adding an edge. Let λ_i be a demanding vertex in the ith f-chain leaf. We add the set of edges $\{(\lambda_i, \lambda_{i+1}) \mid 1 \leq i \leq 2\delta\}$. It is also easy to show that the lower bound given in Lemma 1 is reduced by 2δ and the graph is balanced.

The Input Graph is Balanced
We first describe the algorithm. Then we give its proof of correctness. In the description, we need the following definition. Let B be a 4-block leaf whose implied vertex in 4-$blk(G)$ is b and let B' be a 4-block leaf whose implied vertex in 4-$blk(G)$ is b'. B' is a *nearest* 4-block leaf of B if there is no other 4-block leaf whose implied vertex has a distance to b that is shorter than the distance between b and b'.

```
{* $G$ is triconnected with $\geq 5$ vertices; the algorithm finds
a smallest four-connectivity augmentation. *}
graph function aug3to4(graph $G$);
{* The algorithmic notation used is from Tarjan [29]. *}
  $T :=$ 4-$blk(G)$; root $T$ at an arbitrary vertex;
  let $\bar{l}$ be the number of degree-1 $R$-vertices in $T$;
  do $\exists$ a 4-block leaf in $G \rightarrow$
    if $\exists$ a degree-3 center $c \rightarrow$
1.    if $lc(G) = 2$ and $wc(G) = 3 \rightarrow$
        {* Vertex $c$ is the center of the star wheel $w$. *}
        $u_1 :=$ the 4-block leaf $\{c\}$;
        let $u_2$ be a a non-special 4-block leaf
      | $\exists$ another degree-3 center $c'$ non-adjacent to $c \rightarrow$
        let $u_2$ be the 4-block leaf $\{c'\}$
      | $\exists$ a special 4-block leaf $b$ non-adjacent to $u_1 \rightarrow$
        let $u_2 := b$
      | $\nexists$ (degree-3 center or special 4-block leaf)
      non-adjacent to $u_1 \rightarrow$
        let $u_2$ be a a 4-block leaf such that $\exists$ an internal
        vertex with real degree $\geq 3$ in their implies path
      fi
    | $lc(G) \neq 2$ or $wc(G) \neq 3 \rightarrow$
      if $\bar{l} > 2$ and $\exists$ 2 critical $F$-vertices $f_1$ and $f_2 \rightarrow$
2.      find two non-special 4-block leaves $u_1$ and $u_2$ such
        that the implied path between them passes through
        $f_1$ and $f_2$
      | $l > 2$ and $\exists$ only one critical $F$-vertex $f_1 \rightarrow$
        if $\exists$ two non-adjacent special 4-block leaves in the
        separating triplet $S_1$ corresponding to $f_1 \rightarrow$
3.        let $u_1$ and $u_2$ be two non-adjacent 4-block leaves
          in $S_1$
        | $\nexists$ two non-adjacent special 4-block leaves in the
        separating triplet $S_1$ corresponding to $f_1 \rightarrow$
4.        let $v$ be a vertex with the largest real degree
          among all vertices in $T$ besides $f_1$;
          if real degree of $v$ in $T \geq 3 \rightarrow$
            find two non-special 4-block leaves $u_1$ and $u_2$
            such that the implied path between them
            passes through $f_1$ and $v$
          fi
          {* The case when the degree of $v$ in $T < 3$ will
          be handled in step 8. *}
        fi
      | $\exists$ two vertices $v_1$ and $v_2$ with real degree $\geq 3 \rightarrow$
5.      find two non-special 4-block leaves $u_1$ and $u_2$ such
        that the implied path between them passes
        through $v_1$ and $v_2$
      | $\exists$ an $R$-vertex $v$ of degree $\geq 4 \rightarrow$
6.      find two non-special 4-block leaves $u_1$ and $u_2$ such
        that the implied path between them passes
        through $v$
      | $\exists$ a $W$-vertex $v$ of degree $\geq 4 \rightarrow$
7.      let $u_1$ and $u_2$ be two non-special 4-block leaves such
        that the implied path between them is
        non-adjacent on $v$
      | $\exists$ only one vertex $v$ in $T$ with real degree $\geq 3 \rightarrow$
        {* $T$ is a star with the center $v$. *}
8.      find a nearest vertex $w$ of $v$ that contains a 4-block
        leaf $v_1$;
        let $w'$ be a nearest vertex of $w$ containing a 4-block
        leaf non-adjacent to $v_1$;
        find two 4-block leaves $u_1$ and $u_2$ whose implied
        path passes through $w$, $w'$ and $v$
        {* The above step can always be done, since $T$ is a
        star. *}
      {* Note that $T$ is path for all the cases below. *}
      | $\exists$ two non-adjacent special 4-block leaves in one
      separating triplet $S \rightarrow$
9.      let $u_1$ and $u_2$ be two non-adjacent special 4-block
        leaves in $S$
      | $\exists$ a special 4-block leaf $u_1 \rightarrow$
10.     find a nearest non-adjacent 4-block leaf $u_2$
      | $\bar{l} = 2 \rightarrow$
        let $u_1$ and $u_2$ be the two 4-block leaves
        corresponding to the two degree-1 $R$-vertices in $T$
      fi
    fi;
    let $y_i$, $i \in \{1,2\}$, be a demanding vertex in $u_i$ such that
    $(y_1,y_2)$ is not an edge in the current $G$;
    $G := G \cup \{(y_1,y_2)\}$;
    update $T$, $\bar{l}$, $lc(G)$, $wc(G)$ and $dc(G)$
  od;
  return $G$
end aug3to4;
```

Before we show the correctness of algorithm aug3to4, we need the following claim and corollaries.

Claim 4 [26] *If 4-blk(G) contains two critical vertices f_1 and f_2, then every leaf is either in an f_1-chain or in an f_2-chain and the degree of any other vertex in 4-blk(G) is at most 2.* □

Corollary 4 *If 4-blk(G) contains two critical vertices f_1 and f_2 and the corresponding separating triplet S_i, $i \in \{1,2\}$, of f_i contains a special 4-block leaf, then its augmenting number equals the number of special 4-block leaves in it.* □

Corollary 5 *Let f_1 and f_2 be two critical F-vertices in 4-blk(G). If the number of degree-1 R-vertices in 4-blk$(G) > 2$ and the corresponding separating triplet of f_i, $i \in \{1,2\}$, contains a 4-block leaf B_i, we can add an edge between a vertex in B_1 and a vertex in B_2 to reduce the lower bound given in Lemma 1 by 1.* □

Theorem 1 *Algorithm aug3to4 adds the smallest number of edges to four-connect a triconnected graph.* □

We now describe an efficient way of implementing algorithm aug3to4. The 4-block tree can be computed in $O(n\alpha(m,n) + m)$ time for a graph with n vertices and m edges [18]. We know that the leaf constraint, the degree constraint of any separating triplet and the wheel constraint of any wheel in G can only be decreased by adding an edge. We also know that $lc(G)$, the sum of degree constraints of all separating triplets and the sum of wheel constraints of all wheels are all $O(n)$. Thus we can use the technique in [26] to maintain the current leaf constraint, the degree constraint for any separating triplet and the wheel constraint for any wheel in $O(n)$ time for the entire execution of the algorithm. We also visit each vertex and each edge in the 4-block tree a constant number of times before deciding to collapse them. There are $O(n)$ 4-block leaves and $O(n)$ vertices and edges in $4\text{-}blk(G)$. In each vertex, we need to use a set-union-find algorithm to maintain the identities of vertices after collapsing. Hence the overall time for updating the 4-block tree is $O(n\alpha(n,n))$. We have the following claim.

Claim 5 *Algorithm aug3to4 can be implemented in $O(n\alpha(m,n)+m)$ time where n and m are the number of vertices and edges in the input graph, respectively and $\alpha(m,n)$ is the inverse Ackermann's function.* □

5 Conclusion

We have given a sequential algorithm for finding a smallest set of edges whose addition four-connects a triconnected graph. The algorithm runs in $O(n\alpha(m,n) + m)$ time using $O(n + m)$ space. The following approach was used in developing our algorithm. We first gave a 4-block tree data structure for a triconnected graph that is similar to the one given in [18]. We then described a lower bound on the smallest number of edges that must be added based on the 4-block tree of the input graph. We further showed that it is possible to decrease this lower bound by 1 by adding an appropriate edge.

The lower bound that we gave here is different from the ones that we have for biconnecting a connected graph [3] and for triconnecting a biconnected graph [10]. We also showed relations between these two lower bounds. This new lower bound applies for arbitrary k, and gives a tighter lower bound than the one known earlier for the number of edges needed to k-connect a $(k-1)$-connected graph. It is likely that techniques presented in this paper may be used in finding the k-connectivity augmentation number of a $(k-1)$-connected graph, for an arbitrary k.

Acknowledgment

We would like to thank Vijaya Ramachandran for helpful discussions and comments. We also thank Tibor Jordan, Arkady Kanevsky and Roberto Tamassia for useful information.

References

[1] G.-R. Cai and Y.-G. Sun. The minimum augmentation of any graph to a k-edge-connected graph. *Networks*, 19:151–172, 1989.

[2] G. Di Battista and R. Tamassia. On-line graph algorithms with spqr-trees. In *Proc. 17th Int'l Conf. on Automata, Language and Programming*, volume LNCS # 443, pages 598–611. Springer-Verlag, 1990.

[3] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976.

[4] D. Fernández-Baca and M. A. Williams. Augmentation problems on hierarchically defined graphs. In *1989 Workshop on Algorithms and Data Structures*, volume LNCS # 382, pages 563–576. Springer-Verlag, 1989.

[5] A. Frank. Augmenting graphs to meet edge-connectivity requirements. In *Proc. 31th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 708–718, 1990.

[6] H. Frank and W. Chou. Connectivity considerations in the design of survivable networks. *IEEE Trans. on Circuit Theory*, CT-17(4):486–490, December 1970.

[7] G. N. Frederickson and J. Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, May 1981.

[8] H. N. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proc. 32th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 812–821, 1991.

[9] D. Gusfield. Optimal mixed graph augmentation. *SIAM J. Comput.*, 16(4):599–612, August 1987.

[10] T.-s. Hsu and V. Ramachandran. A linear time algorithm for triconnectivity augmentation. In *Proc. 32th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 548–559, 1991.

[11] T.-s. Hsu and V. Ramachandran. On finding a smallest augmentation to biconnect a graph. In *Proceedings of the Second Annual Int'l Symp. on Algorithms*, volume LNCS #557, pages 326–335. Springer-Verlag, 1991. *SIAM J. Comput.*, to appear.

[12] T.-s. Hsu and V. Ramachandran. An efficient parallel algorithm for triconnectivity augmentation. Manuscript, 1992.

[13] T.-s. Hsu and V. Ramachandran. Three-edge connectivity augmentations. Manuscript, 1992.

[14] S. P. Jain and K. Gopal. On network augmentation. *IEEE Trans. on Reliability*, R-35(5):541–543, 1986.

[15] T. Jordan, February 1992. Private communications.

[16] Y. Kajitani and S. Ueno. The minimum augmentation of a directed tree to a k-edge-connected directed graph. *Networks*, 16:181–197, 1986.

[17] A. Kanevsky and R. Tamassia, October 1991. Private communications.

[18] A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *Proc. 32th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 793–801, 1991.

[19] G. Kant. Linear planar augmentation algorithms for outerplanar graphs. Tech. Rep. RUU-CS-91-47, Dept. of Computer Science, Utrecht University, the Netherlands, 1991.

[20] G. Kant and H. L. Bodlaender. Planar graph augmentation problems. In *Proc. 2nd Workshop on Data Structures and Algorithms*, volume LNCS #519, pages 286–298. Springer-Verlag, 1991.

[21] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 869–941. North Holland, 1990.

[22] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. In *Proc. 19th Int'l Conf. on Automata, Language and Programming*, 1992, to appear.

[23] T. Masuzawa, K. Hagihara, and N. Tokura. An optimal time algorithm for the k-vertex-connectivity unweighted augmentation problem for rooted directed trees. *Discrete Applied Mathematics*, pages 67–105, 1987.

[24] D. Naor, D. Gusfield, and C. Martel. A fast algorithm for optimally increasing the edge-connectivity. In *Proc. 31th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 698–707, 1990.

[25] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*. Morgan-Kaufmann, 1992, to appear.

[26] A. Rosenthal and A. Goldner. Smallest augmentations to biconnect a graph. *SIAM J. Comput.*, 6(1):55–66, March 1977.

[27] D. Soroker. Fast parallel strong orientation of mixed graphs and related augmentation problems. *Journal of Algorithms*, 9:205–223, 1988.

[28] K. Steiglitz, P. Weiner, and D. J. Kleitman. The design of minimum-cost survivable networks. *IEEE Trans. on Circuit Theory*, CT-16(4):455–460, 1969.

[29] R. E. Tarjan. *Data Structures and Network Algorithms*. SIAM Press, Philadelphia, PA, 1983.

[30] S. Ueno, Y. Kajitani, and H. Wada. Minimum augmentation of a tree to a k-edge-connected graph. *Networks*, 18:19–25, 1988.

[31] T. Watanabe. An efficient way for edge-connectivity augmentation. Tech. Rep. ACT-76-UILU-ENG-87-2221, Coordinated Science lab., University of Illinois, Urbana, IL, 1987.

[32] T. Watanabe, Y. Higashi, and A. Nakamura. Graph augmentation problems for a specified set of vertices. In *Proceedings of the first Annual Int'l Symp. on Algorithms*, volume LNCS #450, pages 378–387. Springer-Verlag, 1990. Earlier version in Proc. 1990 Int'l Symp. on Circuits and Systems, pages 2861–2864.

[33] T. Watanabe and A. Nakamura. On a smallest augmentation to triconnect a graph. Tech. Rep. C-18, Department of Applied Mathematics, faculty of Engineering, Hiroshima University, Higashi-Hiroshima, 724, Japan, 1983. revised 1987.

[34] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *J. Comp. System Sci.*, 35:96–144, 1987.

[35] T. Watanabe and A. Nakamura. 3-connectivity augmentation problems. In *Proc. of 1988 IEEE Int'l Symp. on Circuits and Systems*, pages 1847–1850, 1988.

[36] T. Watanabe, T. Narita, and A. Nakamura. 3-edge-connectivity augmentation problems. In *Proc. of 1989 IEEE Int'l Symp. on Circuits and Systems*, pages 335–338, 1989.

[37] T. Watanabe, M. Yamakado, and K. Onaga. A linear time augmenting algorithm for 3-edge-connectivity augmentation problems. In *Proc. of 1991 IEEE Int'l Symp. on Circuits and Systems*, pages 1168–1171, 1991.

On four-connecting a triconnected graph

Hsu, T.
Dept. of Comput. Sci., Texas Univ., Austin, TX , USA;

This paper appears in: **Foundations of Computer Science, 1992. Proceedings., Annual Symposium on**

Meeting Date: 10/24/1992 - 10/27/1992
Publication Date: 24-27 Oct. 1992
Location: Pittsburgh, PA USA
On page(s): 70 - 79
Reference Cited: 37
Inspec Accession Number: 4488295

Abstract:
The author considers the problem of finding a smallest set of edges whose addition f
connects a triconnected graph. This is a fundamental graph-theoretic problem that h
applications in designing reliable **networks**. He presents an O(nα(m,n)+m) time
sequential algorithm for four-connecting an undirected graph G that is triconnected b
adding the smallest number of edges, where n and m are the number of vertices an
edges in G, respectively, and α(m, n) is the inverse Ackermann function. He present
new lower bound for the number of edges needed to four-connect a triconnected gra
The form of this lower bound is different from the form of the lower bound known fo
biconnectivity augmentation and triconnectivity augmentation. The new lower bound
applies for arbitrary k, and gives a tighter lower bound than the one known earlier f
number of edges needed to **k-connect** a (k-1)-connect graph. For k=4, he shows th
this lower bound is tight by giving an efficient algorithm for finding a set edges with
required size whose addition four-connects a triconnected graph

Index Terms:
computational complexity computational geometry four-connecting graph theory graph-theo problem inverse Ackermann function reliable **networks** triconnected graph computational complexity computational geometry four-connecting graph theory graph-theoretic problem inverse Ackermann function reliable **networks** triconnected graph

Documents that cite this document
There are no citing documents available in IEEE Xplore at this time.

Search Results [PDF FULL-TEXT 776 KB] PREV NEXT DOWNLOAD CITATION

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ| Terms | Back to Top

A Flexible Architecture for Multi-Hop Optical Networks

A. Jaekel, S. Bandyopadhyay and A. Sengupta

School of Computer Science, University of Windsor, Windsor, Ontario N9B 3P4, CANADA

Department of Computer Science, University of South Carolina, Columbia, SC 29208

Abstract

It is desirable to have low diameter logical topologies for multihop lightwave networks. Researchers have investigated regular topologies for such networks. Only a few of these (e.g., GEMNET [8]) are scalable to allow the addition of new nodes to an existing network. Adding new nodes to such networks requires a major change in routing scheme. For example, in a multistar implementation, a large number of retuning of transmitters and receivers and/or renumbering nodes are needed for [8]. In this paper, we present a scalable logical topology which is not regular but it has a low diameter. This topology is interesting since it allows the network to be expanded indefinitely and new nodes can be added with a relatively small change to the network. In this paper we have presented the new topology, an algorithm to add nodes to the network and two routing schemes.

Keywords: *optical networks, multihop networks, scalable logical topology, low diameter networks.*

1. Introduction

Optical networks [1] are interconnections of high-speed broadband fibers using *lightpaths*. Each lightpath provides traverses one or more fibers and uses one wavelength division multiplexed (WDM) channel per fiber. In a multihop network, each node has a small number of lightpaths to a few other nodes in the network. The physical topology of the network determines how the lightpaths get defined. For a multistar implementation of the physical topology, a lightpath $u \rightarrow v$ is established when node u broadcasts to a passive optical coupler at a particular wavelength and the node v picks up the optical signal by tuning its receiver to the same wavelength. For a wavelength routed network, a lightpath $u \rightarrow v$ might be established through one or several fibers interconnected by router nodes. The lightpath definition between the nodes in an optical network is usually represented by a directed graph (or digraph) G = (V, E) (where V is the set of nodes and E is the set of the edges) with each node of G representing a node of the network and each edge (denoted by $u \rightarrow v$) representing a lightpath from u to v. G is usually called the logical topology of the network. When the lightpath $u \rightarrow v$ does not exist, the communication from a node u to a node v occurs by using a (graph-theoretic) path (denoted by $u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_{k-1} \rightarrow v$) in G using k hops through the intermediate nodes $x_1, x_2, \ldots, x_{k-1}$. The information is buffered at intermediate nodes and, to reduce the communication delay, the number of hops should be small. If a shortest graph-theoretic path is used to establish a communication from u to v, the maximum hop distance is the *diameter* of G. Clearly, the lightpaths need to be defined such that G has a small diameter and low average hop distance. The indegree and outdegree of each node should be low to reduce the network cost. However, a reduction of the degree usually implies an increase in the diameter of the digraph, that is, larger communication delays. The design of the logical topology of a network turns out to be a difficult problem in view of these contradictory requirements. Several different logical topologies have been proposed in the literature. An excellent review of multihop networks is presented in [1].

Both regular and irregular structures have been studied for multihop structures [2], [3], [4], [5], [6], [7]. All the proposed regular topologies(e.g., shuffle nets, de Bruijn graphs, torus, hypercubes) enjoy the property of simple routing algorithms, thereby avoiding the need of complex routing tables. Since the diameter of a digraph with n nodes and maximum outdegree d is of $O(\log_d n)$, most of the topologies attempt to reduce the diameter to $O(\log_d n)$. One common property of these network topologies is the number of nodes in the network must be given by some well-defined formula involving network parameters. This makes the topology non-scalable. In short, addition of a node to an existing network is virtually impossible. In [8], the principle of shuffle interconnection between nodes in a shufflenet [4] is generalized (the generalized version can have any number of nodes in each column) to obtain a scalable network topology called GEMNET. A similar idea of generalizing

0-8186-9014-3/98 $10.00 © 1998 IEEE