Broadcast in Fast Networks

Ajei Gopal*
Cornell University
Ithaca, NY 14853
ajei@cs.cornell.edu

Inder Gopal Shay Kutten
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
gopal,kutten@ibm.com

Abstract

The current trend in network technology is to implement as much of the switching function as possible directly in specialized high-speed hardware. This paper presents a broadcast algorithm for such a network that is tolerant of failures in the form of message loss. The model used in this paper is based on the one introduced by Cidon et al; the hardware functions assumed are simple enough to be implemented in high speed logic. The basic idea is to forward broadcast messages directly in hardware, thereby avoiding software introduced delays. Software intervention (possible only *after* the broadcasted message has already been forwarded) is required only to ensure termination in case of failures. With high probability, the broadcast will terminate in time $O(n\tau_{max})$ where n is the number of nodes and τ_{max} is an upper bound on (variable) message delivery time across a link.

*This research was conducted when this author was visiting the IBM T. J. Watson Research Center. Additional support was provided by an IBM graduate student fellowship

1 Introduction

Broadcasting is the task of delivering a single message from a particular source node to all the other nodes in the network. Broadcast algorithms form a fundamental component of any computer network and are often used as subroutines in more complex algorithms. In this paper, we study broadcasting techniques in arbitrary topology, point-to-point networks; these techniques take advantage of the function provided by the low-level switching hardware, do not require knowledge of the network size or structure, and do not rely on pre-established routing tables.

The traditional point-to-point network model is of relatively fast processors interconnected by relatively slow links, with all of the packet[1] forwarding and processing functions performed by the processors. Typically, broadcast in such networks is implemented by *flooding* protocols [Tanne81] [MRR80]. In a common version of flooding [Segal83], the source node sends m, the message to be broadcast, to all its *neighbors* (nodes that are directly attached by a communication link). On receipt of m a node forwards it to all its neighbors, provided it has not received m earlier. To ensure termination of the flood, a node for-

[1] We use the term packet synonymously with message.

CH2826-5/90/0000/0338/$01.00 © 1990 IEEE

wards m only once, even if it receives m multiple times. This typically requires a node to maintain some history of prior messages received.

The current trend in packet switched networks is toward higher and higher link speeds [KMS87], and the accompanying trend has been to implement the packet switching function in specialized high speed hardware, rather than in general purpose processors. In order to achieve the required speed cost-effectively, the hardware must be simple; in particular, the hardware cannot access a variable length, unordered history. Thus, traditional flooding algorithms cannot be cheaply implemented directly in fast hardware, and hence any implementation of such an algorithm (and hence of traditional broadcast) will be significantly slower than the network speed.

This paper studies *fast* distributed broadcast protocols for a high speed network. The basic goal of our algorithm is to employ the simple hardware switching functions in order to perform a controlled flood of the network that proceeds at *hardware speeds.*

The network model we assume is realistic, and is similar to high-speed networks currently under development [CiGo88] [TuWy83]. Nodes are interconnected by bidirectional communications links. These links are typically very reliable but messages can be lost because of buffer overflow in the switching hardware (perhaps due to traffic congestion). It is important to note that the probability of such a message loss is *very small*[CGGS88]. Furthermore, we assume that since the switching time and number of buffers are known, there is a known upper bound on message delivery time between adjacent nodes.

The broadcast mechanisms described in this paper are intended for use in situations where no knowledge exists about the network structure or size. We do not envisage them being used to deliver high volume traffic (eg. video distribution). This kind of traffic will use optimized distribution trees that have been set up using information about the current loading conditions in the network [?]. A primary use for the broadcast mechanisms in this paper will be for the flow of control messages (such as topology update messages). These messages are not generated very frequently (in the case of topology update messages only when a link or node fails or recovers). However, when a message is generated, it is important that it be delivered quickly to every node in the network.

Cohen and Segall [CoSe88] have independently proposed a fast broadcast and query algorithm that is also designed for hardware implementation. The motivation for their work is very similar to ours though their node model is different. In particular, it uses more variables and operations within the hardware switching component.

1.1 Model

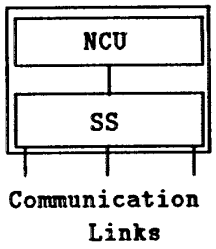

Figure 1: Switching Node

The network model assumed in this paper is based on the model introduced in [CGK88]. A communication network is represented by an arbitrary graph $G = (V, E)$, where V repre-

sents the set of nodes and E represents the set of bidirectional communication links that interconnect the nodes. A node (see Figure 1) is composed of two parts; a high speed switching hardware unit (Switching Subsystem (SS)) and a single (slow) processor (Network Control Unit (NCU)). Any operation performed by the NCU is potentially slow; any operation performed by the SS is considered to take negligible time. In this paper, the SS is frequently referred to as the 'hardware', and the NCU is referred to as the 'software'. All (bidirectional) communication links are attached directly to the SS. The SS and the NCU of the same node are also connected by a bidirectional communication link.

NCU involvement in message routing can significantly increase the message processing time at a node. However, sophisticated message processing cannot be done without NCU involvement, as the SS is a simple piece of hardware without any processing capabilities. Its only function is fast routing of messages. With respect to broadcast messages, the routing functions are simply to recognize a distinct broadcast address on an incoming message, and send this message on all outgoing links, including the one to the NCU[2]. More formally, we represent a message by the concatenation of two substrings, $p = x.y$, where x (the message *header*) is the first k bits of p Define BCAST-ID to be a fixed k bit sequence and a message of the form BCAST-ID.y to be a BROADCAST-message. Define the following operation performed by the SS (called a FORWARD): if a message BCAST-ID.y arrives on an input link, the message BCAST-ID.y is output over *each* output link. The SS also maintains a single bivalent variable called STATE, taking on the values "exception" and "normal". This variable will be used to disable the hardware FORWARD function of the SS in order to guarantee termination of the algorithm.

Each SS maintains a hardware clock that can be used to measure time intervals. As mentioned before, since buffering, switching, and link transmission delays are bounded, we assume that a message will either take less than τ_{max} time to traverse a link, or it will be lost. We say a *failure* has occured when a message is lost. Note that we do not assume links or nodes can permanently fail during the operation of the algorithm. The assumption is that failures are caused by random events such as bit errors or buffer overflows and that subsequent transmission of the same message over a link will have a high probability of success. We assume FIFO ordering of messages transmitted over the same link.

1.2 Goals and Overview of the Results

An efficient broadcast protocol must exploit the two-level model presented earlier, by pushing as much of the work as possible into the hardware (SS), resorting to software (NCU) intervention only when "there is a problem". The two requirements for a broadcast protocol are *termination* and *delivery* defined below:

Termination: A broadcast is said to terminate when all network events (message sends/receives) that were caused by that broadcast have ceased.

Delivery: A broadcast message is said to have been successfully delivered if it has been received by all the nodes in the network.

This paper introduces two algorithms (A and B). For simplicity of presentation, algorithms A and B are initially described with the assumption that broadcasts do not overlap – that is, two broadcasts are never active in the network at the same time. Section 4

[2] Other routing functions are described in [CGK88].

deals with overlaps by proposing randomized extensions to the algorithms, and addresses the issue of multiple simultaneous broadcasts.

Algorithm A consists only of actions executed by the SS at each node – that is, it can be implemented completely in hardware. When there are no failures, rapid (in time $O(D\tau_{max})$, where D is the diameter of the network) delivery and termination are guaranteed; however, the algorithm may not terminate when failures occur.

Algorithm B is generalization of A, that behaves similarly to A when no failures occur, guaranteeing rapid delivery and termination. In contrast to A, algorithm B terminates and guarantees delivery even if failures occur; this is ensured by some software intervention. After a message m is FORWARDed by the hardware, the software is then invoked to decide when the hardware can return to its original state. This software check begins concurrently with the hardware broadcast, and hence the broadcast is *not* held up. Any action that the software in the nodes take can only affect the *next* broadcast, as the current broadcast of m will have already completed.

Algorithm A is introduced in the next section, along with a proof of rapid termination and delivery in the no failure case. This algorithm is very simple, but may cycle in case of a failure. The non-terminating behavior of the algorithm in a failure case is also illustrated; this analysis suggests the basic idea that we generalize the algorithm to deal with failures. The subsequent section introduces algorithm B, along with a proof of termination and delivery.

2 A First Attempt

A simplistic use of hardware, such as performing a FORWARD at a node every time a broadcast message is received will lead to a non-terminating broadcast. Figure 2 presents an algorithm (A) designed to run at the SS at each node. A broadcast is initiated when the NCU at the source node sends out a broadcast message along the link to the node's SS. On receipt of a broadcast message while in the normal STATE, the SS executes a FORWARD, sets its STATE to exception, restoring it to the normal STATE after $2\tau_{max}$ time. Note that if a FORWARD message is received while the SS is in the exception state the message is discarded.[3]

The intuition behind this algorithm is that once node v's SS enters the exception state, it remains in that state long enough to ensure that any additional copies of the message m that are sent by any of v's neighbors are received by it while v is still in the exception state. Hence v will discard all duplicate copies of m.

Theorem 1 ***Assuming no failures and no other concurrent broadcasts, algorithm A ensures termination and delivery in time $O(D\tau_{max})$, where D is the diameter of the network.***

Proof: The proof will show delivery and termination separately.

Delivery: from connectivity.

Termination: from the following, that shows that no node FORWARDs the same message m twice. Let u be the first node that FORWARDs m more than once. Consider u's neighbor v that sent it the message that caused the second FORWARD at u; clearly this message must be received by u after it performed its first FORWARD.

[3] Some of the methods for dealing with multiple concurrent broadcasts (Section 4) may require that such messages are delivered to the NCU, rather than be discarded.

On receipt of broadcast message
(m = *BCAST-ID.p*) and (STATE = normal):
/* assume receipt at time t_{rcv} */
FORWARD
/* forward BCAST-ID.p on each link */
STATE := exception
/* disable further hardware broadcasts */
deliver m to NCU
set timeout to expire at time $t_{rcv} + 2\tau_{max}$

On expiration of the timeout:
STATE := normal
/* to enable further broadcasts */

Figure 2: Algorithm A – executed by the SS at each node

By assumption, that message must have been due to the first FORWARD at v. The latest time this first FORWARD could have been done is within time τ_{max} after u's first FORWARD (caused by v receiving a message from u). Note that v's FORWARD could be due to a message received from some other processor, in which case it will be performed even earlier. Thus, u must receive v's message within τ_{max} time (at the latest) after v FORWARDed the message – that is, within $2\tau_{max}$ of u's first FORWARD. But u must still be in the exception state when v's message arrives at u, contradicting the assumption that v caused the second FORWARD at u.

From the previous previous paragraph, and the fact that each node is reached by the broadcast in at most $D\tau_{max}$ time, we get that the termination time is proportional to $O(D\tau_{max})$. □

The above analyses for Algorithm A relied heavily on failure-freedom. However, there are simple examples that show how a single failure can result in a non-terminating broadcast.

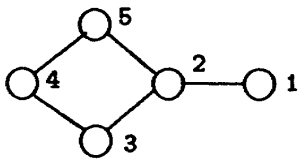

Figure 3: Error Scenario for Algorithm A

Lemma 2 ***Algorithm A may not terminate when one failure can occur.***

Proof: Consider the network shown in Figure 3. Assume that a broadcast originates at node 1 at time t when all the nodes are in the normal state, and that it takes a message x time units to traverse a link, where $2x > \tau_{max}$. Suppose that this broadcast message is received at node 2, where it is forwarded back to node 1 and to nodes 3 and 5[1], and that the $2 \rightarrow 5$ message is lost (the only failure in the scenario). It will take just $4x$ time for the message to return to node 2 via nodes 3, 4, and 5 (the message sequence $2 \rightarrow 3, 3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 2$). Since $4x > 2\tau_{max}$, node 2 will have returned to the normal state by the time it receives the message from node 5. Therefore, the message $5 \rightarrow 2$, will cause node 2 to FORWARD again. The forwarded message to node 5 will be received while node 5 is in the exception state, and hence will not be FORWARDed again. However, the $2 \rightarrow 3$ message will be received by node 3 in the normal state, causing 3 to FORWARD the message again. This reasoning can be repeated to

[1] Let $u \rightarrow v$ denote a message from node u to node v.

show that the broadcast will visit the nodes 2, 3, 4, 5 (in order) an infinite number of times. Such an infinite path is called a *cycle.* □

The presence of a failure is necessary but not sufficient for non-termination. In general, the effects of a single failure can be quite unpredictable, depending on the topology of the network and the actual time taken to traverse links.

3 Failure Resistant Protocol

To ensure termination despite failure, we modify algorithm A to include software intervention. Observe that a cycling broadcast will terminate if each node remains in the exception state for a sufficiently long period of time. Since the time is *a priori* unknown to each node, any time picked may be too low and thus useless as the node will be back in the normal state when the broadcast visits the node again

One trivial solution is for the SS to remain in the exception state indefinitely, disabling the fast hardware broadcast mechanism; hence the network may have to relay on the traditional slower methods. Blocking time is defined to be a measure of how much time is spent in the exception state by any node.

Blocking time: The blocking time is defined to be the maximum time spent by any node in the exception state during the course of a single broadcast.

Since we present algorithms A and B without consideration for multiple simultaneous broadcasts, the importance of the blocking time will be seen in Section 4 (on multiple broadcasts). Note that the smaller the blocking time, the better the service for a *subsequent* broadcast. By seeking to minimize the blocking time, we prevent trivial solutions such as the one given above.

Under fault free conditions, Algorithm A works because it ensures that a "wave" of exception state nodes is created from the source outwards, preventing a broadcast message from looping back and reentering a previously visited node. Consider a non-terminating broadcast shown in the example in the previous section. Node 1 receives the broadcast the second time because it leaves the exception state without ensuring that its neighbor has entered the exception state. This "breaks" the "wave" of exception state nodes and a broadcast message is permitted to reented a previously visited node. If each node stays in the exception state until it is certain that every one of its neighbors has entered the exception state, the "wave" is preserved and the broadcast should terminate.

Algorithm B implements this intuition. As in algorithm A, when a broadcast message m is received while node v's SS is in the normal state, the SS FORWARDs the message, and goes into the exception state. However, unlike Algorithm A, it remains in exception state until the NCU software informs it that it can reenter the normal state. When the NCU receives the first copy of a broadcasted message, it sends a copy of this message to all its neighbors. This message is sent reliably using a Data Link Protocol operating between neighboring NCU's to ensure reliable delivery despite failures. When it recieves a copy of the message from all of its neighbors, it then informs its SS that it can revert to the normal state and perform a hardware forwarding of the next broadcast.

We emphasize that the involvement of the NCU does not slow down the broadcast during normal failure free operation. The hardware still forwards the message as in Algorithm A. Involvement of the NCU occurs only after the message has been forwarded. Our primary objective, namely to deliver the message

to every node at hardware speeds is accomplished. However, the involvement of the NCU does place processing overheads on it which are comparable with the processing overheads required during a normal flooding algorithm.

A quasi formal code for Algorithm B is given in Figure 4.

A simple optimization of the algorithm which considerably reduces the message flow is for each NCU to send to its neighbor an indication of receipt of the message, rather than a copy of the the message itself. If the NCU does not receive an indication of receipt from a neighbor within a specified time-out interval, it sends to that neighbor a copy of the original message.

In the absence of failures, the results of Theorem 1 are valid for algorithm B, and it satisfies the requirement of rapid termination and delivery.

Theorem 3 *Algorithm B ensures eventual termination and delivery even in the presence of failures.*

Proof: The proof follows in a manner very similar to the proof for Algorithm A. As before, we show delivery and termination separately.

Delivery: follows from connectivity. Since each NCU sends a copy of the message reliably to its neighbor, it is possible to trace a chain of reliable message transfers from the source to any node in the connected network.

Termination: from the following, that shows that no node FORWARDs the same message m after it reenters the normal state. Let u be the first node that FORWARDs m after it reenters the normal state. Consider u's neighbor v that sent it the hardware broadcast message that caused the FORWARD at u. The NCU in node v must have sent (the NCU in) node u a reliable message and this message must have been received before node u can exit the exception state and reenter the normal state. However, a node sends a reliable message only after it enters the exception state. Thus when node v sent the reliable message it must have already have entered the exception state. The hardware broadcast message that caused u to perform the FORWARD was received after the receipt of the reliable message. Consequently, by the FIFO properties of the channel, the hardware broadcast message must have been sent after the reliable message. Node v must have been in the normal state in order to FORWARD a broadcast message. Consequently, node v performed a hardware broadcast after reentering the normal state and it performed this before u. This creates a contradiction. □

4 Extensions

Recall that we envisage this broadcast mechanism being used primarily for the flow of control traffic (such as topology update messages). These messages are not generated very frequently but when they are, it is important that they be delivered quickly. The time period between successive message generations is likely to be large. In addition, as the network is very fast, it is likely that algorithm B will terminate rapidly. Thus, the probability is small that two broadcasts will overlap (that is, be active in the network at the same time). This section discusses way of dealing with this low probability event.

4.1 Randomization

One heuristic measure of the likelihood of overlap is given by the *blocking time*. Recall that this is defined to be the sum total of all the time spent in the exception state by all the nodes during the course of a broadcast. Considering algorithm B again, notice that a

broadcast can be forced to terminate even if only a few of the nodes in the network were to remain in exception for the required time. The problem then becomes to decide in a distributed manner which node in the network ought to go into exception; this leads to the use of randomization as a symmetry breaking mechanism. After a node performs a FORWARD, it goes into the exception state, and with probability p it behaves as in Algorithm B, and with probability $1 - p$, it behaves as in Algorithm A. This leads to a randomized algorithm that terminates with probability 1 despite the occurrence of failures, with a lower blocking time than algorithm B. However, this algorithm does not guarantee delivery. A detailed analysis of this algorithm remains for future research.

4.2 Multiple Simultaneous Broadcasts

A generalization of the model allows us to extend the algorithms to deal with *num-bcasts* simultaneous broadcast. Define a set of states, STATE-SET, and a set of broadcast ids, BCAST-SET, such that |STATE-SET| = |BCAST-SET| = *num-bcasts*. Then, algorithms A and B can be extended in the natural fashion, by linking BCAST-ID[k] and STATE-SET[k]: a broadcast for a message with BCAST-ID[k] is performed if and only if the STATE-SET[k] is in the normal state. The source of a broadcast must choose the BCAST-ID for that broadcast.

However, even if some fixed number of concurrent broadcasts are allowed, the problem of overlapping broadcasts remains. One approach is to rely on some higher level of the communications hierarchy to preform retransmission if required. Another approach is to detect overlaps and ensure that every broadcast reaches every node. A slight modification of Algorithm A and (the hardware part of) Algorithm B will deliver messages received by the SS, while in the exception state, to the NCU. Based on its message history an NCU can determine which messages are new (and have to be forwarded as either a software flood or rebroadcast), and which have already been forwarded.

This procedure of overlap detection can be improved by another generalization to the model. Assume that a few context bits can be stored at each node. These bit are taken from the message header, and could be either a sequence of random bits generated by the broadcaster, or a geographic identifier - perhaps a name for some cluster of nodes. When a message is *FORWARDED*, the context bits are stored in the SS. When the SS notices a message arrival during exception, it alerts the software only if the context bits in the message are different from those already stored in the hardware. Thus, duplicate messages will never be delivered, although it becomes possible to lose messages. However the probability of message loss equals the probability that the header information for two different messages are identical. If there are k bits of context information, and the context bits are chosen randomly, the probability of message loss by this mechanism is $\leq (1/2)^k$.

5 Conclusions and Future Work

This paper is based on a communications model that is becoming increasingly important, with the introduction of high-speed switching networks. The key to the model is the separation of a node into a fast and simple hardware component that performs basic switching operations and a software component. The paper solves a fundamental prob-

lem in such networks – that of a broadcast at hardware speeds.

The first algorithm, A, is implemented exclusively in hardware and ensures termination and delivery in time proportional to the diameter of the network; however, it can cycle infinitely in the unlikely event of a failure in the form of a message loss. Algorithm B addresses this problem. It is based on involving the NCU to determine when the SS can move back into the normal state. A related work [GGK90], describes another approach to dealing with failures. This approach uses NCU involvement to determine the appropriate timeout interval without any additional reliable message exchange between neighboring NCU's. However, the guarantee of delivery is not provided by this algorithm.

Future work includes an analysis of a randomized variation of these algorithms, a probabilistic analysis of all the algorithms assuming other distributions on the link delays and an analysis of the generalizations to multiple broadcasts.

6 Acknowledgements

We are very grateful to Yehuda Afek for his contribution to the proof of Lemma 2, to Israel Cidon for conversations that motivated this work, to Samir Khuller for listening to and commenting on the results, to Micky Rodeh for drawing our attention to a reference and to Vasant Shanbhogue for reading a draft of this paper.

References

[BeYa76] J. L. Bentley and A. C. Yao, "An Almost Optimal Algorithm for Unbounded Searching," *Information Processing Letters* 5 (1976), pp. 82-87.

[CGGS88] I. Cidon, I. Gopal, G. Grover and M. Sidi, "Real Time Packet Switching: A Performance Analysis," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 9, December 1988.

[CGK88] I. Cidon, I. Gopal and S. Kutten, "New Models and Algorithms for Future Networks," *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, Totonto, Ontario, Canada, August 1988, pp. 75-89.

[Crist88] Flaviu Cristian, "Probabilistic Clock Synchronization," IBM Research Report RJ6432, October 1988.

[DGS85] S. B. Davidson, H. Garcia-Molina and D. Skeen, "Consistency in Partitioned Networks," *ACM Computing Surveys*, Vol. 17.3, September 1985.

[Even79] S. Even, "Graph Algorithms," Computer Science Press 1979.

[KMS87] P. Kaiser, J. Midwinter and S. Shimada, "Status and Future Trends in Terrestrial Optical Fiber Systems in North America, Europe and Japan," *IEEE Communications Magazine*, Vol. 25, No. 10, October 1987.

[MRR80] J. McQuillan, I. Richer and E. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, Vol COM-28, May 1980.

[Segal83] A. Segall, "Distributed Network Protocols," *IEEE Transactions on Information Theory*, January 1983.

[CoSe88] R. Cohen and A. Segall, "A Distributed Query Protocol for High Speed Networks," *Proceedings of the Ninth*

International Conference on Computer Communication, Tel-Aviv, Israel, October 1988, pp. 299-302.

[SrTo87] T. K. Srikanth and Sam Toueg, "Optimal Clock Synchronization," *Journal of the ACM*, Vol 34, Number 3, July 1987.

[CiGo88] I.Cidon and I.Gopal, "PARIS: An approach to private integrated networks," *Journal of Analog and Digital Cabled Systems*, June 1988.

[TuWy83] J.S. Turner and L.F. Wyatt, "A packet network architecture for integrated services," Proc. of Globecom'83, pp. 45-50, Nov. 1983.

[Tanne81] A. Tannenbaum, Computer Networks, Prentice Hall 1981.

[GGK90] A.S.Gopal, I.S.Gopal and S.Kutten, "Hardware Flooding," Manuscript in preparation, 1990.

ALGORITHM EXECUTED BY SS:

```
On receipt of broadcast message
    (m = BCAST-ID.p) and (STATE = normal):
    FORWARD
    /* forward BCAST-ID.p on each link */
    STATE := exception
    /* disable further hardware broadcasts */
    deliver message m to NCU
On instruction from NCU:
    change state as instructed
    /* enable or disable further
    hardware broadcasting*/
```

ALGORITHM EXECUTED BY NCU:

```
On receipt of a message
    (m = BCAST-ID.p):
    /* m rcvd. by hardware broadcast */
    if (first receipt of m)
        send m reliably
        to neighboring NCU's
    /* message sent over reliable
    point-to-point connection */
 On receipt of a message m
    from neighboring NCU:
    /* m received over reliable
    point-to-point connection */
    if (first receipt of m)
        send m reliably
            to neighboring NCU's
        instruct SS to enter
            exception state
    if (message m
        received from all neighbors)
        /* received over reliable connections */
        instruct SS to enter normal state
```

Figure 4: Algorithm B