Computer Systems | D. Siewiorek Editor

Reverse Path Forwarding of Broadcast Packets

Yogen K. Dalal and Robert M. Metcalfe
Xerox Corporation and
Stanford University

A broadcast packet is for delivery to all nodes of a network. Algorithms for accomplishing this delivery through a store-and-forward packet switching computer network include (1) transmission of separately addressed packets, (2) multidestination addressing, (3) hot potato forwarding, (4) spanning tree forwarding, and (5) source based forwarding. To this list of algorithms we add (6) reverse path forwarding, a broadcast routing method which exploits routing procedures and data structures already available for packet switching. Reverse path forwarding is a practical algorithm for broadcast routing in store-and-forward packet switching computer networks. The algorithm is described as being *practical* because it is not *optimal* according to metrics developed for its analysis in this paper, and also because it can be implemented in existing networks with less complexity than that required for the known alternatives.

Key Words and Phrases: reverse path forwarding, broadcast packets, routing, computer networks, store-and-forward packet switching, broadcast protocols
CR Categories: 3.81, 4.32, 5.32

1. Introduction

Store-and-forward packet switching computer networks provide communication *using* computers as well as communication *among* computers. The delay versus throughput characteristics of these networks make them exceptionally suitable for general purpose computer-to-computer communication [10]. Such networks use so-called routing algorithms, which direct a packet from its source node to its destination node. Store-and-forward packet switching networks based on point-to-point circuits, like the Arpanet [14], use either static or dynamic routine tables to minimize the delivery delay due to store-and-forward transmissions.

While *point-to-point* packet switching has proven very useful in computer resource sharing, certain distributed computing applications requiring *multipoint* communication have been suffering. *Broadcast* multipoint communication is the delivery of messages to *all* destinations, while *multicast,* or multidestination delivery, is the delivery of messages to some specified subset of all the destinations. Ethernet [11], the DCS Ring [6] and other broadcast networks are ideally suited for such communication. *Broadcast routing* is defined to be the routing procedures by which broadcast is achievable in inherently nonbroadcast communication networks [4]. Broadcast routing is a special case of multidestination routing.

Some applications in distributed computing environments require broadcast while others require multicast. For example, in the Arpanet user authentication and billing scheme [3], a Tip must locate an RSExec server to process and verify a user's password, and then periodically record the user's charges. Multicast helps in locating one of several RSExec servers faster, and can help in redundantly storing accounting data. Autodin I supports multidestination addressing and the average address multiplicity per message is 1.75 [13]. On the other hand, broadcast routing could be used in the case where the headquarters of a corporation each day issues directives and news to all its branch offices which are connected together by, say, a private store-and-forward communication network.

Algorithms for broadcast routing through a store-and-forward packet switching computer network include (1) transmission of separately addressed packets, (2) multidestination addressing, (3) hot potato forwarding, (4) spanning tree forwarding, and (5) source based forwarding. To this list of algorithms, we present in this paper (6) reverse path forwarding, a broadcast routing method which exploits routing procedures and data structures already available for packet switching.

Reverse path forwarding is a practical algorithm for broadcast routing in store-and-forward packet switching computer networks. The algorithm is described as being *practical* because it is not *optimal* according to metrics developed for its analysis in this paper, and also because it can be implemented in existing networks with less complexity than that required for the known alternatives.

Section 2 of this paper proposes a simple abstract model with which to view store-and-forward packet routing. Section 3 describes alternatives for broadcast

This work was done at the Stanford University Digital Systems Laboratory, supported in part by the Advanced Research Projects Agency, Department of Defense, under Arpa Order Number 2494, Contract Number MDA903-76C-0093, and in part by the National Science Foundation, under Research Grant MCS73-07973-A1,2.

Authors' address: Xerox Systems Development Department, Xerox Corporation, 3408 Hillview Avenue, Palo Alto, CA 94304.

routing. Reverse path forwarding is described in Section 4, and its extension in Section 5. The performance of reverse path forwarding as compared to the other known alternatives is discussed in Section 6. Section 7 discusses the reliability of broadcast routing algorithms, and Section 8 presents our conclusions.

2. Store-and-Forward Routing

Efficient techniques for store-and-forward routing in packet switching networks are vital to achieve low delay, congestion free communication. This subject has received considerable attention and analysis [7, 8, 12].

Typically, each node in the network has a *routing function* that determines the appropriate outgoing link that an arriving packet should take toward its destination. The routing function is based on *route maintenance* and *link maintenance* algorithms. Route maintenance is the updating of the *routing table,* used by the routing function, based on estimates of delay between nodes of the network. Link maintenance is the determination of expected delays along links connected to a node. This information is used to influence the route maintenance algorithm. Link maintenance in a node involves that node and its links, while route maintenance involves all nodes and links.

Route maintenance can be performed *statically* or *dynamically.* Static route maintenance involves setting up the routing tables at the start, once and for all, with no maintenance at all. Dynamic route maintenance is sensitive to traffic patterns and network topology changes. Dynamic route maintenance can be achieved by a *centralized* or *distributed* algorithm. Centralized route maintenance involves some centralized measurement and control authority. This authority collects statistics on internode delays and then updates the routing tables at each node. Distributed route maintenance involves each node in updating its own routing tables based on communication with other nodes (typically the number is much smaller than the total number of nodes in the network). A detailed taxonomy of route maintenance algorithms can be found in [12].

The output of the routing function can be a link or multiple alternative links. The latter kind of output may be used to transmit successive packets over different links in order to distribute packets in the network uniformly, thereby not producing localized points of congestion, or causing the route maintenance algorithm to oscillate.

2.1 Abstract Model

Store and forward routing algorithms can be modeled abstractly in the following manner:

A *network* consists of *nodes* labeled 1, 2, 3, ..., N. Each node is connected to some other set of nodes by *links* that are labeled 1, 2, 3, ..., L relative to that node. A node's *neighbors* are those nodes connected to it by links. Each node knows its own identity, referred to as *self,* but not necessarily the identity of its neighbors. We assume, without loss of generality, that there is exactly one *host* computer connected to each packet-switching node through its link 0. Since each node has only one host connected to it, node addresses are synonymous with host addresses.

A *packet* is defined by *SourceNode,* the source of the packet, by *DestinationNode,* the intended destination of the packet, and by *Text,* the contents of the packet. At each node, a packet is associated with an *IncomingLink,* the link on which it arrived, and an *OutgoingLink,* the link over which it is to be transmitted. The OutgoingLink is determined by applying the routing function, *routing,* to the DestinationNode of the arriving packet. Stated more formally, for the arriving packet, OutgoingLink ← Routing[DestinationNode]. If OutgoingLink is 0, then the packet is destined to the host connected to the node. The DestinationNode for such packets is equal to self. If OutgoingLink is −1, then the DestinationNode is unreachable, and the packet is discarded.

We assume that route maintenance attempts to minimize delivery delay. We assume further that delay is measured in hops, the number of links over which a packet is transmitted to reach its destination. These assumptions allow the reader to work examples simply with small integers evident from topology.

3. Broadcast Routing Algorithms

In this section we briefly describe the five other methods for routing broadcast packets [2, 4], and in the following sections we describe reverse path forwarding. A special destination address, *AllNodes,* is reserved for packets that are to be broadcast. Some algorithms will use the distinguished AllNodes address, while others will not.

These are metrics by which the broadcast routing algorithms are evaluated:

(1) $C(n)$, the number of packet copies transmitted to broadcast a packet from the host connected to node n. The unit for $C(n)$ is *packet-hops.* The minimum value for $C(n)$ is $2N - 1$, where N is the number of nodes. This is so because exactly one packet copy is transmitted from the source node per destination node, accounting for $N - 1$ packet-hops, and there are an additional N packet-hops to and from hosts and nodes. Each of the $N - 1$ destination hosts requires its own copy from its node, and one copy is required from the source host to source node.

(2) $A(n)$, the average delay across the $N - 1$ destination hosts, before a host receives a broadcast packet from the host connected to node n.

(3) $M(n)$, the maximum delay across the $N - 1$ destination hosts, before a host receives a broadcast packet from the host connected to node n.

Other nonquantitative measures are (1) the changes to packet format, (2) the amount of table space in each node specifically for broadcast routing, (3) algorithm complexity in the node, and (4) reliability of the broadcast.

Network *flooding* is defined as the excessive retransmission and forwarding of a broadcast packet. Broadcast routing algorithms should be stable enough never to cause flooding.

Since broadcast routing algorithms route broadcast packets to many destinations, many of the parameters in the abstract model described in Section 2.1 will become sets. For example, OutgoingLink will generalize to *OutgoingLinkSet.* If OutgoingLinkSet is empty, then the packet is discarded. *AllLinks* is the set of links {0, 1, 2, ..., L} at each node. We introduce such generalizations or modifications as they arise.

3.1 Separately Addressed Packets

The simplest method for achieving broadcast is to make a copy of the broadcast packet, at the source, one for each destination, and to use the normal routing mechanism for delivering each one. This is how multicast is achieved in the Arpanet.

The disadvantages of this method are that many more packet copies are transmitted than necessary, the average and maximum delays are quite large because of queueing delays, and the level of congestion within the network increases with the large number of packet copies and their interference with one another.

On the positive side, no changes are required to the routing algorithm or packet format, and the reliability of broadcast is as much as the underlying routing mechanism.

3.2 Multidestination Addressing

If packets could carry multiple destination addresses, then existing routing mechanisms can be used to achieve broadcast routing with the optimal number of packet copies and minimum delay. DestinationNode would generalize to *DestinationNodeSet.* Packets would then have a variable length destination address field to carry multiple addresses, or perhaps a fixed length bit map.

Copies of the broadcast packet are made at a node when the different destinations in DestinationNodeSet imply different outgoing links. Copies partition the *IncomingDestinationNodeSet* which is the set of destination nodes in an incoming packet. A modified copy of the arriving packet is generated for each member of the OutgoingLinkSet. An outgoing packet lists in its DestinationNodeSet a partition of the DestinationNodeSet of the incoming packet. For each Link in the OutgoingLinkSet, a copy is made of the incoming packet such that its *OutgoingDestinationNodeSet* (the set of destination nodes in an outgoing packet) is the set of all nodes such that the Node is in IncomingDestinationNodeSet, and Routing[Node] is equal to the outgoing Link. Stated more formally:

```
OutgoingLinkSet ← {Routing[IncomingDestinationNodeSet]}
FOR Link ∈ OutgoingLinkSet DO
  BEGIN
  OutgoingDestinationNodeSet ← { }
  FOR Node ∈ IncomingDestinationNodeSet DO
    IF Link = Routing[Node] THEN
      OutgoingDestinationNodeSet ← OutgoingDestinationNodeSet
        + {Node}
  copy packet on Link
  END.
```

This algorithm requires a modification to the routing algorithm to extend it to handle multiple destinations in a packet. No additional tables are needed by the algorithm. This algorithm permits multicast, by just restricting the OutgoingDestinationNodeSet at the source.

3.3 Hot Potato Forwarding

This algorithm and the remaining ones described in this section use the reserved destination address AllNodes.

During hot potato forwarding, each node copies the arriving broadcast packet on all links except the IncomingLink [1]. This simple scheme produces network flooding very quickly, and steps must be taken to prevent it.

The simplest method to prevent flooding is to have a *SequenceNumber* field in each broadcast packet. Each node remembers for some time the sequence numbers of broadcast packets copied and forwarded, from various sources, and does not make copies of any broadcast packets it has already seen. This method is not attractive because it is hard to define how many sequence numbers must be remembered and for how long.

Alternately broadcast packets could have a *HopCount* field, which is incremented each time copies are made of a broadcast packet. Upon receiving a broadcast packet whose HopCount field has exceeded a *threshold,* the packet is discarded thereby abating network flooding. In terms of the abstract model, the actions performed at each node on the arrival of a packet are:

```
IF DestinationNode = AllNodes THEN
  BEGIN
  HopCount ← HopCount + 1
  IF HopCount = threshold THEN OutgoingLinkSet ← {0}
  ELSE IF HopCount > threshold THEN OutgoingLinkSet ← { }
    ELSE OutgoingLinkSet ← AllLinks − IncomingLink
  END
ELSE OutgoingLinkSet ← {Routing[DestinationNode]}
```

This method requires modification of format for broadcast packets, and some more processing within the node. Using HopCounts still generates far too many packets for a safe value of threshold, for example, the diameter of the network. The *diameter* of a network is the length of any longest geodesic. A *geodesic* is the shortest path between any two nodes [9]. The large number of packet copies generated may produce congestion within the network. Further, packet copies will be in existence for a time larger than the maximum delay before a node receives the broadcast packet, and therefore the delay for subsequent packet transmissions will

also increase. The reliability of hot potato routing is very high, and this is the very reason that Baran proposed it in 1965.

3.4 Spanning Tree Forwarding

A spanning tree for a network is a tree that spans the nodes of the network, such that if any branch of this tree were removed, then the two remaining subtrees would be disconnected. A network can have more than one spanning tree.

In graph theoretical terms, a spanning tree is defined as follows: Consider a connected, undirected graph, G, with vertex set V, and edge set, E (E is a subset of $V \times V$). A *spanning tree* is a subset of E, such that there is a unique path between any two vertices in V. A spanning tree is also a one-connected graph. Suppose there is a cost associated with every edge in E; a *minimal spanning tree* of G is a spanning tree that minimizes the sum of the cost of the edges.

If a single spanning tree were overlayed on the network topology, then every node would lie on it. Each node would know which of its links were branches of the spanning tree. A broadcast could be initiated from any node. At each node, the broadcast packet will be copied and transmitted along the branches of the spanning tree, except along the branch on which it arrives. If the broadcast packet arrives on a link not in the spanning tree, then it is discarded, since its transmission can produce flooding. For a packet to arrive on a link not in the spanning tree, there must be an inconsistency in the routing information which might lead to flooding, and therefore discarding the packet is safe. Stated in terms of our model:

```
IF DestinationNode = AllNodes
THEN OutgoingLinkSet ← BroadcastRouting[IncomingLink]
ELSE OutgoingLink ← Routing[DestinationNode].
```

This scheme is optimal in the number of packet copies, but not delay. The delay is a function of the position of the node within the spanning tree from where the broadcast was initiated.

The algorithm requires no change to the packet format and makes use of the reserved destination address AllNodes. There is a small table at each node. The spanning tree could be static and set up initially. Alternately the spanning tree could be dynamically updated using a centralized computation. If the spanning tree was a minimal spanning tree then a somewhat expensive distributed computation could be used to dynamically change it [4].

3.5 Source Based Forwarding

A broadcast packet that arrives at a node is copied along a subset of the links according to the data structure in each node which lists the outgoing links to use as a function of packet source. This data structure is based on the shortest path trees from each node to all the others, or in other words the algorithm is based on having a delay-minimizing spanning tree for each source node rather than only one for all nodes. This algorithm makes use of the reserved destination address AllNodes as well. In terms of the abstract model, the actions a node performs on receiving a packet is:

```
IF DestinationNode = AllNodes
THEN OutgoingLinkSet ← BroadcastRouting[SourceNode]
ELSE OutgoingLink ← Routing[DestinationNode].
```

The characteristics of this algorithm are that the arriving packet is copied but not modified, the optimal number of packet copies is generated, and the transmission delay is minimum. There is, however, a large table at each node, and updating this table in a dynamic manner involves a complex computation.

4. The Reverse Path Forwarding Algorithm

The reverse path forwarding algorithm broadcasts packets with the reserved destination address AllNodes based on the source of the packet. This is achieved without any extra tables and routing procedures beyond those described in the abstract model for point-to-point routing.

When a broadcast packet arrives at a node on an IncomingLink, the node will copy this packet on AllLinks except the IncomingLink, if and only if the node believes that the best way to get to SourceNode is via IncomingLink. If this condition is not met, then the broadcast packet is discarded, thereby preventing network flooding. This algorithm has the flavor of both hot potato and source based forwarding. In terms of the abstract model, the actions a node would perform on receiving a packet are:

```
IF DestinationNode = AllNodes THEN
BEGIN
  IF IncomingLink = Routing[SourceNode]
  THEN OutgoingLinkSet ← AllLinks − {IncomingLink}
  ELSE OutgoingLinkSet ← { }
  END
ELSE OutgoingLink ← Routing[DestinationNode].
```

The algorithm derives its name, reverse path forwarding, from the observation that a broadcast packet is accepted for forwarding only if it arrives along a link that is part of the *shortest path tree* to SourceNode from all the other nodes. This is the *shortest reverse path tree* from SourceNode, and is in general different from the shortest path tree from SourceNode to all the other nodes. In the case that delays over links are the same in both directions, as with hop counts, the shortest path tree from SourceNode and the shortest reverse path tree from SourceNode are isomorphic.

The shortest reverse path trees are very easily derivable from the shortest path trees from a node, which is what the route maintenance algorithm uses to build the routing table. Figure 1 illustrates how the reverse path forwarding takes place from node 8, along the reverse path tree from node 8. The packet copies are labeled $P_{i,j,\ldots,k}$, where i is the source of the packet and k its immediate desti-

Fig. 1. Reverse path forwarding initiated from node 8.

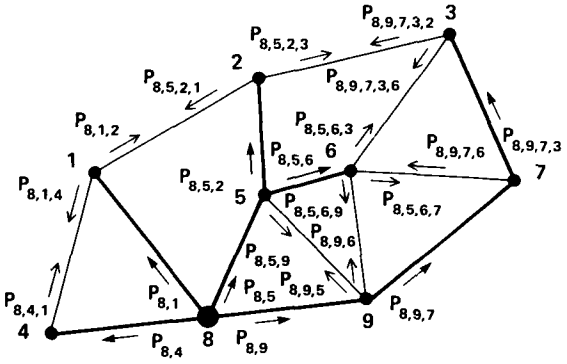
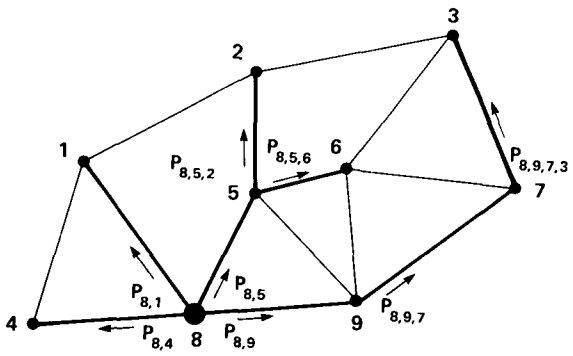


Fig. 2. Extended reverse path forwarding initiated from node 8.



nation. The sequence $i, j, \ldots, k$ defines the genealogy of the packet. $P_{8,9}$ is the first copy arriving at a node 9 after departure from the source 8. Likewise for $P_{8,9,7}$. Note that more packet copies are transmitted than necessary. 31 packet copies are transmitted while the optimal number is 17. (Remember to count the N copies needed between nodes and their hosts.) The number of packet copies transmitted is equal to the sum of the cardinality of AllLinks at each node, minus $(N - 1)$, because each node forwards the broadcast packet on |AllLinks| − 1 links, except the source which has packet copies transmitted on |AllLinks|. For most network topologies, this number is larger than the optimal value.

Although extra packet copies are generated, the extra packets are *immediately* discarded, thereby preventing network flooding.

There is no change in the packet format or data structures at each node. The extension to the routing function in each node is very simple, and could be made very easily to the program in the Imps and Tips in the Arpanet [12].

The delays to perform broadcast are minimum if the shortest reverse path trees are isomorphic to the shortest path trees, as would be the case if hop count was the measure of delay. The two trees would also be isomorphic if communication traffic was symmetric over a link thereby producing equal delay in both directions, an assumption that is generally not true [5].

5. The Extended Reverse Path Forwarding Algorithm

It is the property of reverse path forwarding that extra packet copies are deleted immediately, by the neighbors of the node generating them. The extension is proposed wherein a periodic exchange of routing information among neighbors allows nodes to avoid the generation of extra packets.

The extended reverse path forwarding algorithm transmits the optimal number of packet copies per broadcast by using tables similar to that used by the source based forwarding algorithm, in conjunction with the basic control structure described in Section 4. The tables list the outgoing links over which an acceptable broadcast packet should be forwarded. The algorithm makes sure that packet copies are transmitted only along the branches of the reverse path tree. The novel feature about this extension to the reverse path forwarding algorithm, is that this data structure can be computed very easily in a distributed fashion, unlike the one used by the source based forwarding algorithm. In terms of the abstract model, the actions performed at a node are:

```
IF DestinationNode = AllNodes THEN
BEGIN
  IF IncomingLink = Routing[SourceNode] THEN
  OutgoingLinkSet ← BroadcastRouting[SourceNode] + {0}
  ELSE OutgoingLinkSet ← { }
  END
ELSE OutgoingLink ← Routing[DestinationNode].
```

Copies of a broadcast packet are made only along the branches of the reverse path tree by using a data structure consisting of a table, *LinkBroadcastTable,* for each of the L links. Each table is indexed by SourceNode and results in a boolean, indicating whether the broadcast packet from SourceNode should be copied on that link or not. A packet addressed to AllNodes should be copied on a link if LinkBroadcastTable[SourceNode] = TRUE, for that link. Figure 2 illustrates the flow of broadcast packets from node 8 along the reverse path tree using this extension.

Consider now the process by which these tables at each node may be updated dynamically, in a distributed manner. In terms of the reverse path forwarding model, in order to construct such a table, a node must know to which destinations a given link will be used by its neighbors to normally transmit a packet. Hence, if a broadcast packet arrived from one of these "destinations," then the node will know along which links to make copies. Periodically, each node will transmit an appropriate LinkBroadcastTable to each of its neighbors. Upon receiving such a table a node merely replaces its old copy by the new one. A LinkBroadcastTable can be computed for a Link directly from the routing function at that node:

```
LinkBroadcastTable[SourceNode] ← Routing[SourceNode] = Link.
```

This algorithm is optimal in the delay metrics if the delays over a link are equal in both directions.

Table I. Performance of the Broadcast Routing Algorithms for the Network of Figure 3.

Performance measure	$C(n)$ packet hops, n				$A(n)$ hops, n				$M(n)$ hops, n				$\overline{C(n)}$	$\overline{A(n)}$	$\overline{M(n)}$
Broadcast routing algorithm	1	2	3	4	1	2	3	4	1	2	3	4			
Separately addressed packets	10	9	10	9	3.6	3.3	3.6	3.3	5	5	5	5	9.5	3.5	5
Multidestination addressing	7	7	7	7	3.3	3	3.3	3	4	3	4	3	7	3.2	3.5
Source based forwarding	7	7	7	7	3.3	3	3.3	3	4	3	4	3	7	3.2	3.5
Hot potato forwarding (threshold = diameter = 2)	13	15	13	15	3.3	3	3.3	3	4	3	4	3	14	3.2	3.5
Spanning tree forwarding	7	7	7	7	3.3	3.3	4	4	4	4	5	5	7	3.7	4.5
Reverse path forwarding	11	11	11	11	3.3	3	3.3	3	4	3	4	3	11	3.2	3.5
Extended reverse path forwarding	7	7	7	7	3.3	3	3.3	3	4	3	4	3	7	3.2	3.5

6. Performance Evaluation

Figure 3 illustrates a simple four node network. Table I indicates the performance of the various broadcast routing algorithms for this network. For each broadcast routing algorithm we determine $C(n)$ which is the number of packet copies transmitted to broadcast to all hosts from a host connected to node n, $A(n)$ which is the average delay, across the $N-1$ destinations, before a broadcast packet from a host connected to node n reaches a destination host, and $M(n)$ which is the maximum delay, across the $N-1$ destinations, before a destination host receives a broadcast packet from a host connected to node n. We also determine $\overline{C(n)}$, $\overline{A(n)}$ and $\overline{M(n)}$ which are averages over all N sources. This simple network is presented so the reader may understand the metrics by checking them in his head. Figure 4 illustrates the logical map for the Arpanet as of August 1976. The network contains 59 nodes. Table II indicates the performance of the algorithms for this network. We only display $\overline{C(n)}$, $\overline{A(n)}$ and $\overline{M(n)}$ which are averages over the N sources for the Arpanet, since the table would have otherwise become extremely large!

The performance was calculated keeping some other issues in mind:

(i) The effect of queueing delays has not been taken into account for any of the other algorithms except separately addressed packets for which it is assumed that a source host can only transmit a packet to the source node after the packet previously transmitted from the source host to its node has reached the node; a delay of one hop.

(ii) When determining $M(n)$ for separately addressed packets it is assumed that the source host transmits packets to the source node in an order that minimizes $M(n)$. The host transmits a packet to the furthest host first and the closest host last. In general, hosts will not have knowledge of the communication network topology, and so this feature provides optimistic values for $M(n)$.

(iii) The spanning tree used for spanning tree forwarding was the minimal spanning tree. This minimal spanning tree is unique since the network is transformed into one with distinct link delays using a simple tie-breaking algorithm that can be computed distributedly [4].

(iv) For the network representing the Arpanet, it was assumed that there was only one host connected to each node, in order to be consistent with the abstract model used in this paper. The metrics would have had much larger values if the actual number was taken into account.

This analysis shows that reverse path forwarding is

Fig. 3. A network and its minimal spanning tree.

Fig. 4. Arpanet logical map, August 1976.

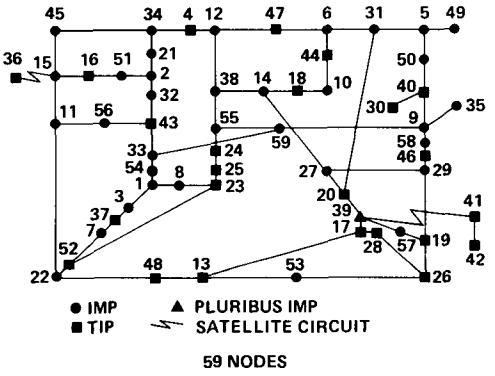


Table II. Performance of the broadcast routing algorithms for the Arpanet.

Performance measure / Broadcast routing algorithm	$\overline{C(n)}$ packet hops	$\overline{A(n)}$ hops	$\overline{M(n)}$ hops
Separately addressed packets	425	35.3	60
Multidestination addressing	117	7.3	11.1
Hot potato forwarding (threshold = diameter = 11)	1238	7.3	11.1
Spanning tree forwarding	117	14.5	27.6
Source based forwarding	117	7.3	11.1
Reverse path forwarding	145	7.3	11.1
Extended reverse path forwarding	117	7.3	11.1

an optimal algorithm under the assumptions made, and even in an operational network in which the reverse path tree from a node is not isomorphic to the shortest path tree from that node its suboptimality under the $C(n)$ and $A(n)$ metric is probably acceptable.

7. Reliability of Broadcast

The preceding performance evaluation misses an important consideration, namely whether broadcast routing within the network can be made reliable, and what the implications of unreliable broadcast is on the higher level applications that rely on this capability. Reliability here is taken to mean the ability to deliver exactly one copy of the broadcast packet to all destinations, under the assumption of a perfectly reliable network; i.e. one that is not partitioned and therefore has a route from every node to every other.

We first examine the reverse path, and extended reverse path forwarding algorithms. If the route maintenance algorithm is static, then exactly one copy of the broadcast packet will be delivered to each host as we have seen in the previous two sections. If, however, the route and link maintenance algorithms are dynamic, as is usually the case, then both the reverse path and extended reverse path forwarding algorithms cannot guarantee to deliver exactly one copy of a broadcast packet to all destinations, as we shall show.

In the reverse path forwarding algorithm, a broadcast packet is transmitted over every link of the network. At any instant a node has one and only one branch corresponding to a given reverse path tree. The reverse path tree for a given SourceNode may, however, change cates or no packets to be delivered to some of the hosts.

For example, in Figure 1, assume that node 2 has just received packet $P_{8,5,2}$ along link (2, 5) which is a branch, but $P_{8,1,2}$ from node 1 has not yet arrived at node 2. Node 2 will deliver $P_{8,5,2}$ to its host and copy the packet appropriately along the other links. The routing function at node 2 may now be updated to reflect that the best way to get to node 8 from node 2 is over link (1, 2) and not (2, 5). Therefore when $P_{8,1,2}$ arrives at node 2 from 5, it too will be delivered to the host connected to node 2 and copied appropriately over the other links. Therefore, the host connected to node 2 received two copies of the broadcast packet.

Similarly, if $P_{8,9,7,3,2}$ and $P_{8,1,2}$ arrived at node 2 before $P_{8,5,2}$, they would be discarded since link (2, 5) is the best way to get to node 8, the SourceNode. The routing function at node 2 may now be updated to reflect the fact that link (2, 5) is not the best way to get to node 8, but (1, 2) is the best way. Subsequently, when $P_{8,5,2}$ arrives, it too will be discarded resulting in the delivery of no broadcast packet to the host connected to node 2.

Consider now the extended reverse path forwarding algorithm. In Figure 2, assume that $P_{8,5,6}$ has arrived at node 6 from 5. It will be delivered to the host connected to 6 and not copied further. Further assume that at this time $P_{8,9,7}$ has not yet arrived at node 7 from 9. Now the LinkBroadcastTables at node 3 are updated to reflect that the best way to get to node 8 is via link (3, 6) rather than (3, 7). Thus, when $P_{8,9,7}$ arrives at node 7, it will be delivered to the host connected to 7 but will not be forwarded. As a consequence, the host connected to node 3 will never receive the broadcast packet.

Similarly, if $P_{8,9,7}$ and $P_{8,9,7,3}$ had arrived respectively at nodes 7 and 3 before $P_{8,5,6}$ reached node 6, and then the very same changes were made to the LinkBroadcastTables at node 3, then the host connected to node 3 will receive duplicate copies of the broadcast packet.

Note that the extended reverse path forwarding algorithm, as stated in Section 5, copies a broadcast packet only if it arrived on the "correct link", i.e. the same link that the node would have used to transmit a packet to the SourceNode using the normal routing function. This test was necessary in the reverse path forwarding algorithm to prevent network flooding. In the extended version of the algorithm, this test is not necessary, if the reverse path tree did not dynamically change, because packets would only be copied along branches of the reverse path tree and therefore arrive only on the "correct link." If the reverse path tree can dynamically change during the course of a broadcast, then a node may copy a broadcast packet on a link it thinks is a branch, but its neighbor at the other end thinks is not. For example, in Figure 2, node 7 may forward $P_{8,9,7}$ to node 3, while node 3 has sent node 7 a LinkBroadcastTable that does not include link (3, 7) for node 8 as SourceNode. Thus, node 3 will conclude that $P_{8,9,7}$ did not arrive on the "correct link" and will discard the packet. This may result in node 3 never receiving an acceptable broadcast packet. Alternatively, if nodes accepted and forwarded broadcast packets based on the source irrespective of the link it arrived on, then more packets would be transmitted than necessary, and some nodes may receive duplicates.

Therefore these algorithms cannot guarantee that a node will receive one and only one acceptable broadcast packet. This is because during the course of a broadcast the reverse path tree changes, and as a result decisions made during the beginning of the broadcast process are incorrect.

Separately addressed packets and multidestination addressing guarantee to deliver exactly one copy of the broadcast packet to all nodes even if adaptive routing mechanisms are used in the network. This is because each destination explicitly appears as a destination address in a packet. Of course, the adaptive routing mechanism may cause a larger number of packet copies to be transmitted than the optimal, and as a result may increase the delay. Source based forwarding and spanning tree forwarding, as described in this paper, guarantee to

deliver exactly one copy to all nodes since they use static routing mechanisms. Static techniques reduce the performance of all routing, broadcast included. Hot potato forwarding guarantees to deliver at least one copy to all nodes if the threshold is chosen correctly.

In reality, the network may partition, since it is not perfectly reliable, and so any of the algorithms described cannot guarantee to deliver exactly one copy of the broadcast packet to all nodes.

8. Conclusions

The extended reverse path forwarding algorithm is practical and optimal if the delays in both directions of a link are identical. This algorithm can be implemented in existing networks using existing routing procedures, and with simple extensions to the data structures. The suboptimality of reverse path forwarding under the $C(n)$ and actual delay metrics is probably acceptable in operational networks as well.

Both reverse path and extended reverse path forwarding algorithms have the tragic flaw, that if the underlying dynamic routing mechanism changes the routing tables used by the nodes during the course of a broadcast, then a broadcast packet may not be delivered to a node even if a path to it exists. This flaw is shared to a greater or lesser degree by the alternatives except for separately addressed packets, multidestination addressing and hot potato forwarding. Duplicate copies may also be delivered.

We conjecture that it is useful to divide broadcast routing algorithms into two classes, one oriented towards sending a message to every host on the network, and the other oriented towards sending a message to a relatively small percentage of all hosts.

Algorithms of the first class will have a reasonable cost for transmitting a message to all hosts, but essentially the same cost for sending a message to even a small group. Those of the second class will have a reasonable cost for sending a message to a small group, but extravagant cost for sending the message to all hosts. Examples of the first class are hot potato forwarding, spanning tree forwarding, source based forwarding and reverse path forwarding, while examples of the second are separately addressed packets and multidestination addressing.

If algorithms that support efficient broadcast routing are used to deliver messages to a subset of the destinations, then a number of hosts will receive packets that they will subsequently discard. This is a wasteful use of a critical resource—the communication channel connecting the host to the network. In such cases the channel may become the bottleneck. Further, each host will have to process the packet in order to decide if it is intended for it.

Of course, techniques of the first class can be modified to be used for restricted broadcast. For example, different spanning trees could be used for different restricted broadcast groups. Alternatively, a node that is on the spanning tree, but whose host does not belong to a restricted broadcast group, will forward the packet to other nodes as per the conventions of the protocol without delivering the packet to its host. They can be useful, however, if communication networks wish to provide special services.

Our division of broadcast routing algorithms into two groups is based on what we imagine communication networks will be used for. The most usual use of broadcast will be to reach a small fraction of the hosts, and therefore there is the need for efficient multicast routing algorithms. However, for limited purpose networks, it is easier to imagine the need for communication with all hosts, and therefore it is desirable to use broadcast routing algorithms.

We have shown that it is, in general, not possible to achieve reliable broadcast routing, but this is true of routing in general. In some applications one might take the position that unreliable broadcast is a prelude to reliable point-to-point communication; for example when a Tip in the Arpanet locates an RSExec server. There will be other applications in which one and only one copy of the broadcast packet must be delivered to all hosts. The *broadcast protocol* within the network or the hosts must sequence broadcast packets (messages) generated from the source, so that duplicates may be filtered at the destinations. Further, responses to a broadcast packet can contain the sequence number of the broadcast, to identify with which message the response is associated. It may be necessary for the recipients to acknowledge the broadcast packet (message), so that the source may retransmit only to hosts that did not respond, thus reducing the bandwidth used in the reliable broadcast. The design of reliable broadcast protocols analogous to reliable point-to-point interprocess communication protocols in computer networks is a subject for future research.

Acknowledgments. An earlier treatment of these algorithms appears in Yogen Dalal's Stanford Ph.D. dissertation supervised by Vinton G. Cerf, Robert M. Metcalfe, Susan S. Owicki, and Philip M. Spira.

Received October 1977

References

1. Baran, P., Boehm, S., and Smith, P. On Distributed Communication. Tech. Rep., Vol. 1-9, Rand Corp., Santa Monica, Calif., 1964.
2. Cerf, V.G. Obtaining broadcast communication from non-broadcast transmission media. Private communication, Sept. 1976.
3. Cosell, B.P., Johnson, P.R., Malman, J. H., Schantz, R.E., Sussman, J., Thomas, R.H., and Walden, D.C. An operational system for computer resource sharing. Proc. Fifth Symp. Operating Syst. Principles, Nov. 1975, 75-81 (available from ACM, New York).
4. Dalal, Y.K. Broadcast protocols in packet switched computer networks., Ph.D. Th., Stanford, DSL Techn. Rep. 128, April 1977; available as R78-64, IEEE Comptr. Repository, IEEE-CS, Long Beach, Calif.
5. Danthine, A.A.S., and Eschenauer, E.C. Influence on packet node behavior of the internode protocol. *IEEE Trans. Comm. COM-24*, 6 (June 1976), 606-614.

6. Farber, D.J., and Larson, K.C. The structure of a distributed computing system—the communication system. Proc. Symp. Computer-Communications Networks and Traffic, Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., April 1972, pp. 21–27.
7. Fultz, G.L. Adaptive routing techniques for message switching computer communication networks. Ph.D. Th., UCLA-ENG-7252, U. of California, Los Angeles, July 1972.
8. Gerla, M. The design of store- and forward (S/F) networks for computer communication. Ph.D. Th., UCLA-ENG-7319, U. of California, Los Angeles, 1973.
9. Harary, F. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.
10. Metcalfe, R.M. Packet Communication. Ph.D. Th., Harvard, Proj. Mac Tech. Rep. No. 114, M.I.T., Cambridge, Mass., Dec. 1973.
11. Metcalfe, R.M., and Boggs, D.R., Ethernet: Distributed packet switching for local computer networks. *Comm. ACM 19,* 7 (July 1976), 395–404.
12. McQuillan, J.M. Adaptive routing algorithms for distributed computer networks. Ph.D. Th., Harvard, BBN Rep. 2831, May 1974; available as AD781467, N.T.I.S., Springfield, Va.
13. Paoletti, L.M. AUTODIN. In *Computer Communication Networks,* R.L. Grimsdale and F.F. Kuo, Eds. (Proc. NATO Advanced Study Inst. Comptr. Comm. Networks, Sussex, U.K., Sept. 1973), Noordoff Int. Publ., Leyden, 1975.
14. Roberts, L.G., and Wessler, B.D. The ARPA computer network. In *Computer Communication Networks,* N. Abramson and F. Kuo, Eds., Prentice-Hall, Englewood Cliffs, N.J., 1972.

Corrigendum. Programming Languages

David Gries, An Exercise in Proving Parallel Programs Correct, *Comm. ACM 20,* 12 (Dec. 1977), 921–930.

Dr. Leslie Lamport detected what appeared to be a methodological mistake in the proof of the on-the-fly garbage collector. The assignment *atleastgray*(*m*[*i*].*left*) of the Collector (see the algorithm labeled (3.6) on page 925) contains references to the *two* shared variables *m*[*i*].*left* and *m*[*m*[*i*].*left*].*color*, and this clearly violates the restriction (2.10) found on page 923.

The problem is not a methodological error but a missing footnote. The statement *atleastgray*(*m*[*i*].*left*) in (3.6) does have a footnote number 3 attached to it, and an earlier version of the paper [Springer Lecture Notes in Computer Science 46, 1976, 57–81] contained the footnote

This should be written as "*t*:= *m*[*i*].*left*; *atleastgray*(*t*)" where *t* is a local variable. Since the mutator never tests the color of a node and only grays a node using also *atleastgray*, the single statement *atleastgray*(*m*[*i*].*left*) is equivalent under parallel operation to this sequence of two operations.

Dr. Lamport also noted that the informal discussion of noninterference of assertions (4.5.1) and (4.5.2) in the first four paragraphs of Section 4.5 could be interpreted as using circular reasoning, but that a formal proof of noninterference does indeed work.

My thanks to Dr. Lamport for pointing out these problems and my apologies for any inconvenience they have caused the reader.

Programming Languages
J.J. Horning Editor

Abstract Data Types and Software Validation

John V. Guttag, Ellis Horowitz, and David R. Musser
University of Southern California

A data abstraction can be naturally specified using algebraic axioms. The virtue of these axioms is that they permit a representation-independent formal specification of a data type. An example is given which shows how to employ algebraic axioms at successive levels of implementation. The major thrust of the paper is twofold. First, it is shown how the use of algebraic axiomatizations can simplify the process of proving the correctness of an implementation of an abstract data type. Second, semi-automatic tools are described which can be used both to automate such proofs of correctness and to derive an immediate implementation from the axioms. This implementation allows for limited testing of programs at design time, before a conventional implementation is accomplished.

Key Words and Phrases: abstract data type, correctness proof, data type, data structure, specification, software specification
CR Categories: 4.34, 5.24

1. Introduction

The key problem in the design and validation of large software systems is reducing the amount of com-

This research was supported in part by the National Science Foundation under Grant MCS76-06089, by the Defense Research Projects Agency under Contract DAHC15 72 C 0308, and by the Joint Services Electronics Program Monitored by the Air Force Office of Scientific Research under Contract F44 620-76-C-0061.

Authors' addresses: J.V. Guttag and E. Horowitz, Computer Science Department, University of Southern California, Los Angeles, CA 90007; D.R. Musser, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90291.