Multicast in Deflection Networks

W. Olesinski and P. Gburzynski
Department of Computing Science
615 GSB, University of Alberta, Edmonton, AB, Canada T6G 2H1
tel: (403) 492-2347 (office), (403) 492-1071 (fax)
e-mail: [wladek,pawel]@cs.ualberta.ca
URL: http://www.cs.ualberta.ca/~[wladek,pawel]

Abstract

Multicasting in deflection networks is more difficult than in store-and-forward networks, because of the difficulties in sending multiple copies of the same packet on several output ports. On the other hand, deflected packets stray from their optimal paths and may visit the multicast recipients "accidentally," not necessary in the order envisioned by the sender. We compare a number of simple multicast strategies for deflection networks based on a few simple rules and assumptions. In particular, we never buffer packets that cannot be relayed immediately.

1. Introduction

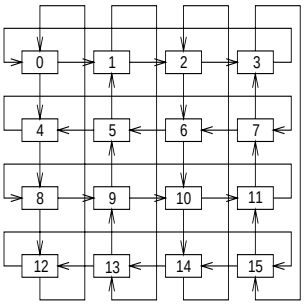
We consider *pure* deflection networks—ones that never lose packets because of a limited buffer space. Packets that at the moment of arrival cannot be relayed via their preferred routes (because those routes are busy) are deflected, i.e., relayed via suboptimal routes. This concept, traditionally illustrated by the Manhattan-street networks (MSN) introduced and analyzed by Maxemchuk in [5, 6], works under the assumption that the amount of buffer space available at a switch is sufficient to determine the route preference of an incoming packet before the packet is relayed via one of the outgoing links. No buffer space is required for storing packets that cannot be immediately relayed via their optimal routes, although it may make sense to store such packets temporarily before deflecting them right away [6].

In a deflection network, a source connected to a switch may send its packet, only if at least one incoming slot is empty. In a heavily loaded network, many slots will be non-empty and the source may not be allowed to transmit a packet immediately.

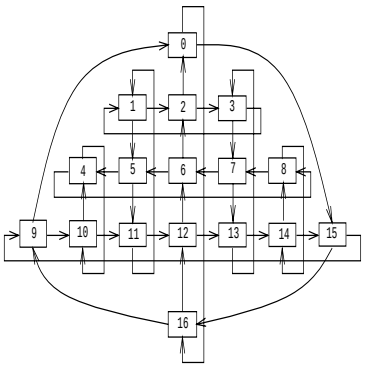
Deflection networks are good candidates for high-speed MANs. In [1] and [8], we argue that their performance for synchronous and isochronous traffic may be quite acceptable, despite the apparent unpredictability of routes. One important feature of a contemporary network is the multicast capability, widely used in many applications, like video teleconferencing, video distribution, distributed computing and control. Unfortunately, deflection networks do not support multicast directly. The fact that packets cannot be buffered at the switches and the unpredictability of paths traveled by those packets make it impossible to apply multicast algorithms used in store-and-forward networks. For example, distance-vector multicast routing [3, 4], link-state multicast routing [3, 7], or other schemes presented by Deering in [3], cannot be easily used in deflection networks because they are based on replication. If copies of a replicated packet cannot be sent over the desired outgoing links immediately, they are simply stored in buffers.This is difficult to do and unnatural in a deflection network, even if it uses some intermediate buffer space, because such space is usually limited [6]. Providing extra buffers just to accommodate multicasting would contradict the spirit of deflection; besides, this approach would produce rather unpredictable results depending on the background load.

Methods based on reverse path forwarding cannot be directly used in deflection networks, not only because they rely on replication. Such methods assume that packets from a given source arrive at the given switch on the same input link. This assumption is used to discard multicast duplicates recognized as those packets that arrive on links different than the one appropriate for the given source. Of course, this simple trick cannot be used in a deflection network. Even in the complete absence of contention, a packet is free to choose among several paths with the same length.

It is thus clear that deflection networks require a different approach to the multicast problem than store-and-forward networks. In this paper, we present a number of simple multicast schemes that can be used with deflection. With our schemes, packet replication is unnecessary, although, as we shall see, it may improve their performance. Our repli-



(a) Torus, size 16, connectivity 2



(b) Triangle, size 17, connectivity 2

Figure 1. Network topologies.

cation avoids the problem of duplicate recognition: multicast packets may be replicated only under specific conditions and each copy survives exactly one hop.

2. The model

We consider several network topologies, including torus and triangle (see Figure 1), as well as some other irregular/biased topologies. The results (at least in relative terms) were highly consistent across the different setups; therefore, we only present in this paper the results for the torus network. Note that 2-connected torus is in fact a Manhattan Street Network.

Each switch is equipped with delay buffers whose purpose is not to store packets before they are forwarded, but to align packets arriving at the switch and to give the switch ample time to make a routing decision. The networks operate in a slotted manner, in a way similar to MSN [6]. In one routing cycle, the switch accepts incoming slots from all its input ports and routes them to the output ports. If an incoming slot is nonempty and addressed to the current switch, the switch receives the contents of the slot and marks it as empty. If an incoming slot is empty, the switch is free to fill it with its own outgoing packet (we assume that every switch has a host capable of contributing some traffic to the network).

The routing strategy assigns output ports to all incoming slots that appear nonempty after the above operations. Packets are assigned to outgoing links such that the sum of the shortest distances to their destinations is minimized. If several possible selections give the same minimum sum, one of them is made randomly. We say that a routing scheme with these properties is *locally optimal* [2]).

In our model, we assume that slots are never buffered at a switch, except for the alignment and routing, and that the total delay involved in a single hop in the network is the same for all links and equal to a single slot. Be expressing all delays in hops, we normalize the results to the (average) hop size. Our experiments with networks using different (also non-identical) distances between neighboring switches have confirmed the validity of this simplification.

There is a distinguished source $S = 0$ that generates multicast packets (also called *M-packets*). In a regular network, exemplified by the torus topology, the selection of that source is immaterial. An M-packet is addressed to some number D of multicast destinations. We assume that the list of those destinations is stored in the packet's header. [1]

For the sake of measurement, the source will send a new M-packet only if the previous one was received by *all* D destinations. This is unrealistic in a real network because a source cannot know exactly when a multicast packet has been received by its last destination. However, this way we can investigate the behavior of individual M-packets without making them compete among themselves.

Our primary performance measure is the average delay suffered by a multicast packet on its way to the *last* recipient. This delay is expressed in hops, i.e., normalized to the (average) link length in the network.

First, we measure the delay versus the number D of multicast destinations in an empty network, i.e., in which the transmission of M-packets is the only activity. Then, we set D to some value and introduce a Poisson background traffic to the network. Under this scenario, every switch other than S generates packets addressed to a single, randomly selected destination. Numerically, this background load is expressed as the number of new packets generated in the network during one slot time. Generated packets are stored in a queue at the source switch, and they are extracted from the queue at every opportunity, i.e., whenever an empty slot happens to be passing through the switch. No performance measures are collected for the background traffic—we only observe how this traffic affects the performance of the proposed multicast schemes.

[1] In Section 4 we discuss methods of encoding this list efficiently.

3. Basic multicast schemes

Let us start from the following generic multicast strategy. The packet is transmitted by the distinguished multicast source S to the switch whose number is at the head of the list of destinations. This first destination is dubbed the *bounce destination*. Every intermediate switch on the path from S to the bounce destination checks if it is listed in the packet's header. If so, the switch receives the packet without removing it from the network, i.e., it clears its entry in the list of destinations stored in the header. If the list is still nonempty, the packet is relayed on the outgoing link, according to the routing scheme. If the list is empty, the packet is removed from the network.

When the packet arrives at the bounce destination and the list of recipients is still nonempty, a new bounce destination is selected and the packet continues its trip. Note that bounce destination may change at most $D - 1$ times.

The above generic strategy will be modified by changing the ways of selecting subsequent bounce destinations. Also, we will occasionally allow the packet to be replicated at a switch. Some of the possible variations are listed below.

- **B_Random**

  This is exactly the generic scheme presented above, in which the bounce destination is selected at random.

- **B_Shortest**

  The bounce destinations are ordered from the closest to the source to the most distant from the source.

- **B_Furthest**

  This approach is opposite to B_Shortest. The bounce destinations are ordered from the furthest from the source to the nearest to the source. The idea is to let the packet visit a few recipients "accidentally" before arriving at the first bounce destination.

- **B_Bounce**

  This scheme is similar to B_Shortest, except that the next bounce destination is chosen as the one being the closest to the current bounce destination. The first bounce destination is chosen as the one being the closest to the source.

- **B_All**

  This scheme resembles B_Bounce (and is identical to B_Bounce if there are no deflections). Any recipient (including "accidental recipients") is promoted to the status of a bounce destination. The next bounce destination is determined as the one being the closest to the current recipient.

4. Implementation

The above strategies assume no packet replication and, at least at first sight, can be viewed as having been presented in the increasing order of complexity. Clearly, B_Random is the simplest strategy, as it requires practically no processing at the source or at the intermediate recipients.

To implement any of the basic strategies, we have to augment the packet header by two fields: multicast session identifier and destination set. We assume that the header also includes two standard fields needed by regular traffic, i.e., the source and destination address.

Upon the setup of a multicast session, the source selects a session identifier, which is supposed to be unique in the network. One part of that identifier can be the source Id, another part can be a source-selected number telling apart different multicast sessions carried out by this source.

Then the source notifies the recipients of the multicast session. Each recipient is assigned a multicast Id between 0 and D, where D is the total size of the recipient pool. This way, the range of the recipient Id is confined to the size of the population of recipients (rather than the number of stations in the network), which makes it easier and more economical to represent recipient sets as bit patterns.

We assume that a set of recipients is represented in a fixed-width binary field (e.g., 32, 64, 128 bits) with every bit position corresponding to one recipient. This solution imposes a limit on D. In Section 8 we discuss methods of overcoming this restriction without inflating the size of the destination set beyond a reasonable value. In asynchronous deflection networks, where packets can be of variable length, the size of the destination set may vary, depending on the actual population of the recipient pool.

The destination address field of an M-packet contains the address of the next bounce destination. This address is first determined by the source and later at every subsequent bounce destination. For the strategies B_Bounce and B_All, this operation is very natural. For the first two strategies, the current bounce destination determines the next bounce destination based on the source address extracted from the packet header. If the network is regular (or almost regular)—see [6]—this involves simple calculations on the row/column coordinates of the respective stations. With the assistance of a dedicated hardware, these calculations can be performed in parallel for all destinations remaining in the set. This is yet another reason for limiting the maximum size of the destination set to a "reasonable" number.

In this context, schemes B_Shortest and B_Furthest appear in fact more complex than the last two schemes. Our results (Section 6) indicate that B_Bounce and B_All are significantly better than the other proposed solutions; therefore, one can view B_Shortest and B_Furthest as only providing two more reference points for evaluation. In fact,

B_Random is the only sensible competitor. Although it generally performs much worse than B_Bounce and B_All, it requires much less processing at the bounce destinations.

5. Limited replication

Since replication schemes in deflection networks are not very useful, we only incorporate a very simple replication technique in our schemes. Namely, if it happens that the neighbor of the current switch appears on the list of destinations, and a free slot is available on the link connecting the current switch to that neighbor, the packet is replicated and addressed specifically to the neighbor. We know that such a packet won't be deflected and that it will reach the destination in one hop.

6. Results

In this section we present some of our simulation results obtained for a deflection network built on the torus topology. The network has $N = 256$ switches, and its connectivity (the number of link pairs per switch) is $k = 2$ or $k = 4$.

6.1. Connectivity-2 networks

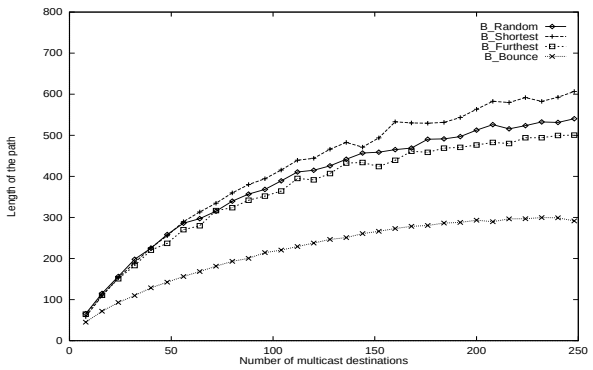


Figure 2. Delay vs D in basic schemes, k=2.

Figure 2 shows the delay vs the number D of multicast destinations when the background load is 0.

As expected, B_Bounce is much better than other schemes. Destinations in the packet's header are reordered from the nearest to the furthest at every bounce switch, with respect to that switch. If there is no contention, this allows a packet to move from one destination to another over the shortest path from among all the strategies considered in this paper.[2] It obviously assures that the delay achieved in this scheme is the lowest.

[2] Note that B_All is identical to B_Bounce if there are no deflections.

Differences between the first three, simpler schemes are rather small. They are caused mainly by what an M-packet can do on its way to the first bounce switch. For a larger number of multicast destinations, B_Furthest is surprisingly the best among the simpler schemes. The reason for this is that the packet tends to visit "accidentally" a few other destinations on the list before it reaches the bounce destination to which it was addressed.

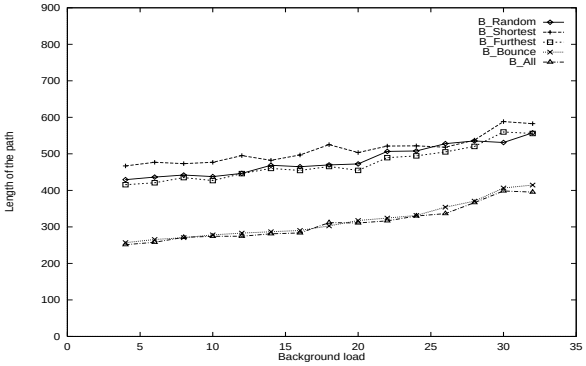


Figure 3. Delay vs background load in basic schemes, k=2.

Now, let us look at the results showing what happens when a background load is present in the network while the number of multicast destinations remains fixed at $D = 128$ (Figure 3). These figures also present B_All, in which destinations are reordered at *all* intermediate destinations, not only at the bounce ones.

For low background loads, the differences between the simplest schemes are similar to what we have seen already. Then, when the load gets heavier, they tend to dissipate. Under such conditions, the number of deflections is so high that it is rather irrelevant where an M-packet is sent first—to the nearest, furthest, or random destination.

B_Bounce is still better than the three simple schemes. However, especially under heavy loads, it is slightly worse than B_All. The reason is clear. When the load is low, M-packets rarely stray from their shortest paths between the destinations, and both schemes are almost identical. When the number of deflections increases, M-packets are sometimes deflected to other destinations on the list. With B_Bounce, such a packet will try to return to the previous path leading to the bounce destination it was addressed to, even if another destination from the list is just one hop away. With B_All, multicast destinations will be reordered and the packet will be forwarded to the nearest destination.

Among all the multicast algorithms presented above, B_Bounce significantly outperforms the others. B_All is slightly better in a heavily loaded network.

6.2. Connectivity 4

Similarly to what we have seen for the low connectivity network, B_Bounce offers the best performance (for the sake of brevity, we have omitted the figures). However, B_Furthest is no longer the best among the simple schemes. Its place in this category was surprisingly taken by B_Random. This can be explained as follows.

In a network with a high connectivity, the average path is shorter compared to the average path length in the same network with a lower connectivity. An M-packet can move faster from one destination to another, so the initial ordering of destinations is less important. The chaotic path of an M packet that made B_Random worse for $k = 2$ is now beneficial. Because of this, the packet has more chances to be received by destinations other than the intended bounce one. But why is it better than B_Furthest and B_Shortest?

In B_Furthest, we rely on "accidental" visits of an M-packet to some of the multicast destinations. Here, such visits are less probable because the paths are shorter. The M-packet gets to the first destination in a fewer number of hops which decreases the probability of being received by some other destinations. Thus, the M-packet tends to be received by more distant destinations first, approaching gradually the source. The probability of visiting "accidentally" destinations other than the bounce one is further decreased in B_Shortest, which again makes this scheme poor.

In a small connectivity network, where an average packet has to make more hops, the unordered path of a packet may overcome the benefits described above.

Among the multicast algorithms applied to the network with connectivity 4, B_Bounce significantly outperforms the simple schemes, and its performance is very close to B_All, even in a heavily loaded network. Since B_Bounce is less complex than B_All, B_Bounce appears to be better. If the low complexity of the scheme is important, B_Random seems to be a not so bad choice. This time, it outperforms the remaining two simple (but more complex) schemes.

6.3. Replication schemes

The schemes presented above are now enhanced with the simple replication feature introduced in Section 5. We expect that this scheme will decrease the time spent by the packet in network, thereby decreasing the delay, particularly under light loads. If the load is high, replications will occur less often because the outgoing links will be more likely to be busy.

The increase of the algorithm complexity is not high. As before, every switch must scan the list of destinations found in the M-packet's header. This time, it also has to find out if any destination on this list is among the switch's neighbors.

The relations between the replication schemes (not shown here) are similar to the relations between the basic schemes discussed in the previous section (figures 2 and 3). The first letter of each scheme's name is now `R' to indicate that the scheme has been augmented by the replication feature.

The difference between R_Random and R_Furthest is smaller than the difference between B_Random and B_Furthest. It seems that the chaotic path of an M-packet in a random scheme is even more beneficial than before. Note that in a replication scheme, a packet may be received by one of its destinations not only if it actually arrives at that destination, but also when it passes in its vicinity. With R_Random, a packet has more opportunities to pass in the neighborhood of one of its destinations, and get replicated. However, this phenomenon loses its impact when the background load is significantly greater than zero. In such a case, R_Furthest performs better, although the differences dissipate for higher loads—as it was the case with the basic schemes.

The relations between particular schemes in the connectivity-4 network are similar to what we have observed for connectivity 2. Notably, the difference between the performance of R_Random and R_Furthest becomes even more significant.

7. Comparison

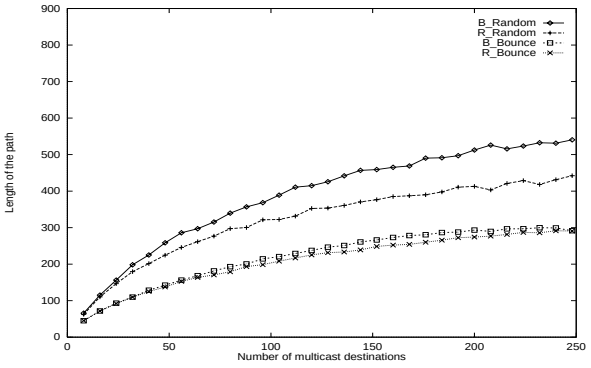


Figure 4. Delay vs D. Comparison, k=2.

Figure 4 shows a comparison between the schemes presented in the preceding sections when the network connectivity is 2 and load is 0 (relations between the schemes are similar when the load is non-zero) The comparison is made between the best simple schemes (i.e., B_Random, R_Random), and the schemes that require packet ordering from some (or all) destinations (i.e., B_All, R_All). Note that B(R)_Bounce is shown rather than B(R)_All—when the background load is 0, these schemes are equivalent.

Among both the simple and more complex schemes, the ones based on replication are better. However, they involve some additional processing at every switch, and their benefits tend to diminish in heavily loaded networks.

If the processing at switches is to be kept at a minimum, the basic schemes still perform quite well. Their most complex member (B_All) also involves processing in some of the intermediate switches, but its complexity is still lower than that of the R schemes.

Comparison between the schemes presented in preceding sections when the network connectivity is 4 gives some surprising results. Namely, if the background load is 0 and D is big, R_Random is better than B_Bounce. When the connectivity is high (and the average path length is small) it is more beneficial to let the packet wander around in the network from destination to destination without any specific order. The path is short, so it will not take the packet too much time to visit all destinations. At the same time, it will be more likely to visit some other destinations "accidentally," particularly if replication is in effect. Thus, we confront two mechanisms: potential length increase of the packet's path, but at the same time an increase in the probability of "accidental" arrivals due to the replication mechanism (R_Random); and potential length decrease of the packet's path, but at the same time a decrease in the number of destinations that may be visited "accidentally" (B_Bounce).

The experiments indicate that the first mechanism gives better results when the background load is low and the network connectivity is high. Of course, the scheme using the advantages of both mechanisms (R_Bounce) performs best.

We have also observed that the performance of the schemes converges to that of the basic schemes when the background load increases. Clearly, with the increasing load, M-packets have fewer chances for being replicated which makes the basic and random schemes behave alike.

8. Multicast groups

As we already mentioned in Section 4, the size of an M-packet's header in the multicast algorithms presented in this paper increases with the increasing limit on D—the number of multicast destinations. To avoid inflating the packet header without limiting the maximum number of multicast destinations too severely, it may make sense to divide a large deflection network into a number of *groups*.

Suppose that the maximum size of a multicast group is n and the number of multicast recipients is $m > n$. Upon the setup of a multicast session, the population of recipients can be logically divided into $G = \lfloor \frac{m}{n} \rfloor$ groups. A multicast source has to send G copies of a single M-packet, each copy addressed to destinations within one group.

9. Summary

We have presented a number of simple multicast schemes for deflection networks. Some of those schemes use a limited replication which doesn't depend on extra buffer space at a switch. The replication schemes outperform other schemes introducing relatively low overhead. Notably, among the simpler schemes, the one involving the least amount of processing (i.e., R_Random) turns out to be the best. In the network with $k = 4$, it is even better than the best basic scheme. Still, if the multicast algorithm complexity is to be kept very low, the basic scheme B_Bounce is a good candidate, particularly in a network with k=2.

The proposed schemes seem to be especially well suited for relatively small multicast groups because the length of an M-packet's header increases with the increasing size of the group. This increase may be reduced by partitioning the multicast recipients—as suggested in Section 8.

In a future study, we are going to investigate the performance of our schemes in some real applications, like videoconferencing and transmission of video films.

References

[1] C. Baransel, W. Dobosiewicz, and P. Gburzynski. Routing in multi-hop switching networks: Gbps challenge. *IEEE Network Magazine*, (3):38–61, 1995.

[2] F. Borgonovo and E. Cadorin. Locally-optimal deflection routing in the bidirectional Manhattan network. In *Proceedings of IEEE INFOCOM'90*, pages 458–464, 1990.

[3] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[4] A. Ford and D. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[5] N. Maxemchuk. The Manhattan Street Network. In *Proceedings of GLOBECOM`85*, pages 255–261, 1985.

[6] N. Maxemchuk. Routing in the Manhattan Street Network. *IEEE Transactions on Communications*, 35(5):503–512, May 1987.

[7] J. McQuilan, I. Richer, and E. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, 28:711–719, May 1980.

[8] W. Olesinski and P. Gburzynski. Real-time traffic in deflection networks. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)*, pages 23–28, San Diego, California, Jan. 1998.