A bounded-hop-count deflection scheme for Manhattan-street networks

Włodek Dobosiewicz* Paweł Gburzyński†

December 22, 1994

*Supported in part by NSERC Grant No. OGP9110. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1. email: dobo@cs.ualberta.ca.

†Supported in part by NSERC Grant No. OGP9183. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1. email: pawel@cs.ualberta.ca.

Abstract

We present a new deflection method suitable for Manhattan-street networks (MSN), which, without dropping packets, limits the number of hops travelled by a packet on its way from source to destination. This allows the destination to bound the size of the reassembly buffer needed to reconstruct the original order of packets that may arrive out of sequence. The proposed routing scheme is intended for bidirectional networks—with four incoming and four outgoing links per switch. In terms of average performance measures, our method performs not worse than the best local routing schemes when the offered load is light or moderate. In particular, when there is no contention, packets move along the shortest routes to their destinations. Routing decisions are made based on locally-available information: the coordinates of the routing switch are compared with the coordinates of the packet's destination. Unlike other deflection schemes proposed for MSN, our method is inherently asynchronous. Every incoming packet is processed independently of other packets that may arrive at (approximately) the same time. This means that packets arriving from different input links don't have to be aligned at the switch. This approach reduces the complexity of the routing algorithm and simplifies the switch design. The network doesn't have to compensate for the slight discrepancies in the transmission rates of different switches. This is the most painful implementation problem with the original synchronous approach and its becomes more severe for larger and/or faster networks.

1 Introduction

Network backbones based on meshed topologies are generally better scalable than unidimensional architectures (e.g., busses, rings, or stars). Unfortunately, the existence of multiple paths between nodes complicates the communication protocols by introducing the problem of packet routing—the very problem that the unidimensional networks were meant to avoid. The efficiency of routing depends on the following factors:

1. The criteria used to determine the preference of an output link for a packet to be routed.
2. The rules to resolve conflicts when two or more packets bid for the the same output link.
3. The options available when a routing decision is made, each option corresponding to a possible mapping between the packets to be retransmitted and the number of output ports available at the moment.

To be fast, robust, flexible, and reliable, meshed networks have to base their operation on very simple routing algorithms, preferably avoiding buffering packets before they are relayed.[1] Manhattan-street networks introduced and analysed by Maxemchuk [10, 11, 13, 14, 15] fulfill this postulate by *deflecting* the packets that cannot be routed via their preferred paths to suboptimal routes. Although with some deflection techniques applicable to Manhattan-street networks a packet may be buffered at an intermediate station for a limited time, a station never risks losing a packet due to the limited buffer space.

The common property of most deflection methods discussed in the literature (e.g., see [1, 2, 5, 6, 8, 9, 10, 11, 13, 14, 15]) is their synchronous character (packets arriving on different input ports of a switch must be aligned before a routing decision can be made). Another attribute of deflection is the inherent stochastic character of the datagram service offered by the network: there is generally no bound on the maximum number of hops that a packet may have to take on its way to the destination. As it has been pointed out by Maxemchuk [14], to make packet delivery more reliable (by avoiding livelocks), one has to **increase** the randomness of the routing function. In consequence, different packets of the same message may suffer unpredictable delays and they may arrive at the destination out of sequence. Even if the traffic is not isochronous, message reassembly at the destination may pose

[1] This general approach is called *wormhole routing* [3, 4].

problems because there is no way to guarantee that the message can be reassembled in any buffer of a finite size. Thus, although a deflection protocol may not lose packets in principle, it may lose them *in fact*, due to the limited size of reassembly buffers.

We propose a deflection scheme applicable to bidirectional Manhattan-street networks [2] which limits the number of hops travelled by a packet on its way from source to destination. This limit is roughly N—the number of stations in the network. Under light and moderate loads, our proposed scheme performs no worse than other local schemes [2, 11, 13, 14] (in terms of average measures). In particular, when there is no contention, packets propagate along the shortest routes to their destinations. The proposed routing method can also be applied to unidirectional networks (as introduced by Maxemchuk in [10]). In unidirectional MSN, our deflection scheme also bounds the maximum number of hops per packet, but in terms of average measures it performs significantly worse than the standard routing methods applicable to MSN.

2 Manhattan-street networks

A unidirectional Manhattan-street network consists of a number of unidirectional rings organised as shown in Figure 1. Every switch receives packets (slots) on two input ports and relays them on two output ports. In the first stage of its operation cycle, the switch aligns a pair of incoming slots such that they both can be processed at the same time. If a slot happens to be addressed to the switch, it is received and emptied. If the switch has its own packet to transmit, it can deposit the packet into an empty slot. Then, for each nonempty slot, the switch ranks the output ports according to the packet's preference. There exist locally applicable rules that determine how this ranking should be carried out (see [11]).

If both outgoing packets clearly prefer different output ports, or one outgoing slot is empty, the routing decision is obvious. In [14], Maxemchuk argues that conflicts should be resolved in the following way:

1. If both packets prefer only one and the same output port, one packet is selected at random and deflected; the other packet is relayed on its preferred port.

2. If one of the packets prefers one output port and the other packet has no preference, the packet with a clear preference is relayed on its preferred port and the other packet is relayed via the

other port.

3. If neither packet has a clear preference, the packets are relayed at random on different output ports.

Although one can think of a number of different strategies of resolving conflicts (e.g., see [14, 13]), the above simple rules have the advantage of avoiding livelocks and, as they offer no worse average performance than other, less randomised rules, we will be using them as a reference point.

In a bidirectional network (see Figure 2), every station has four input and four output ports. In the most extreme case, a routing decision must allocate output ports to four outgoing packets. In line with the postulates of Maxemchuk [14], we opt for the natural strategy discussed in [2] which is randomised and offers no worse average performance than other realistic methods. With this strategy, the multiple packets arriving at a switch within one slot interval are routed in such a way as to maximise the reduction of their combined distance to destinations. Whenever several decisions would lead to the same maximum reduction, one of them is made at random.

The two standard routing strategies described above will be collectively referred to as *random routing* and denoted by *RR* for brevity. They can be viewed as a single strategy applied to two networks with different degrees of switch connectivity. Their common properties are:

- **Synchronised nature.** Every routing decision is made synchronously for all the packets/slots that arrive on all the input ports of a switch within the current slot-time.

- **Local optimality.** When there is no contention, packets propagate along the shortest paths to their destinations (we will call this property of a routing scheme the *shortest path property*). In the case of a conflict, the global "satisfaction" of all outgoing packets with the routing decision is maximised. Routing decisions are based on local information statically available at the switch and they don't account for the history of routed packets.

- **Nondeterminism.** Whenever a number of routing decisions result in the same average quality of service, all these decisions have the same likelihood of being made.

- **Unbounded hop count.** There is no guarantee that the number of hops travelled by a given packet will be less than any number specified in advance.

The local optimality criteria are defined under the assumption that the incoming packets/slots are never buffered at the switch, except for the unavoidable delay required to align the multiple slots

arriving at the switch within the current slot-time and to carry out the routing operation. One can consider a variation of *RR* in which a limited buffer space is available at the switch and packets that cannot be routed along their preferred paths are stored temporarily. The role of buffering in such a case is to enlarge the pool of packets used as the input to the local optimisation problem. As determined experimentally by Maxemchuk in [13], a network with no buffers achieves between 55% and 79% of the throughput achievable by the same network using infinite buffers.

The alignment of the incoming slots at a switch can be implemented by placing insertion buffers between the input ports and the routing logic, as shown in Figure 3. Slots arriving from different rings are delayed for different durations and, at the beginning of each routing cycle, all input slots are always available. In a realistic environment, one should not assume that the transmission rate of different switches is exactly the same. To account for this discrepancy, the insertion buffers may have to be much larger than one-slot per input port. In a moderately-sized network, this problem can be solved by adding "safety gaps" around slots. As the size of these gaps tends to grow with the number of switches in the network, in a large network one may consider using backpressure techniques or the method described in [12] to keep the slot arrival rate steady.

3 Vertical routing in unidirectional MSN

We will denote switches by pairs of numbers representing their <*column*,*row*> coordinates in the MSN grid. By convention, we will assume that rows and columns are numbered from zero up, starting from the left upper corner of the grid. For reasons that will become clear shortly, we call our routing method *vertical routing*, which is abbreviated as *VR*. We start from the unidirectional case.

Assume that the network operates in a slotted mode—in the same synchronised fashion as the standard MSN. Suppose that a packet p addressed to destination $s_d =< c_d, r_d >$ arrives at switch $s =< c, r >$. If $s_d = s$, the packet has reached its destination and it will be received by the switch. Otherwise, p must be relayed on one of the two output ports. In this routing scenario, p is called a *column packet* if $c_d = c$, i.e., the destination switch of p is located in the same column as s. If p is not a column packet (i.e., $c_d \neq c$) then we will call it a *remote packet*. If $r_d = r$ then, besides being a remote packet, p is also called a *row packet*.

In essence, *vertical routing* is based on the following rules:

1. Whenever a column packet arrives at a switch from the row, the packet is routed to the col-

umn and the packet arriving from the column (if the current slot arriving from the column is nonempty) is directed to the row.

2. In any other circumstances, a (remote) packet arriving from the row is routed to the row and a packet arriving from the column is routed to the column.

A switch willing to insert its own packet into the network awaits a moment when at least one of the outgoing slots is empty. Then:

- If the empty slot has arrived from the row, the switch is allowed to fill it with any packet (i.e., a remote packet as well as a column packet).
- If the empty slot has arrived from the column, the switch is allowed to fill it with a column packet only.

The packet will be routed in the same way as if it just arrived from the network—according to the routing rules listed above. Note that a column packet inserted into an empty slot arriving from the row will be immediately directed to the column.

In the absence of contention, a remote packet inserted by its source switch will propagate along the row until it finds the column of its destination. Then the packet will be directed to the column and stay there until it hits its target. Notably, this may not be the optimum route. In the MSN grid, with alternating directions of adjacent rows/columns, a shorter number of hops can be achieved by making a "U-turn," i.e., changing the row/column direction by jumping to an adjacent row/column. Although such paths are natural with RR, they are illegal with VR. Consequently, VR (in its unidirectional version discussed in the present section) loses significantly to RR in terms of average performance measures.

An advantageous property of VR is that every packet reaches its destination after a bounded number of hops. To see this, let us consider the fate of a packet that just departs from its source. If the packet is remote, it will be routed to the row and, regardless of the traffic conditions at intermediate switches, it will stay in the row until it reaches the destination column. Then the packet will be guaranteed to make at least one hop towards its destination switch—along the destination column. At the next switch along that column, the packet may be deflected to the row,[2] but after making a

[2]If it happens to collide with a column packet arriving from the row.

full (necessarily undisturbed) circle through the row, it will arrive back at the column and will be guaranteed again at least one hop towards the destination. Let $s_s =< c_s, r_s >$ and $s_d =< c_d, r_d >$ denote the source and destination switch of the packet, respectively. If N_c and N_r represent the number of columns and rows in the network ($N = N_c \times N_r$), then:

$$\delta_c(s_s, s_d) = \begin{cases} (c_d - c_s + N_c) \bmod N_c & \text{if } r_s \text{ is even} \\ (c_s - c_d + N_c) \bmod N_c & \text{otherwise} \end{cases}$$

gives the number of columns separating the source from the destination along the source's row r_s. Similarly:

$$\delta_r(s_s, s_d) = \begin{cases} (r_d - r_s + N_r) \bmod N_r & \text{if } c_d \text{ is even} \\ (r_s - r_d + N_r) \bmod N_r & \text{otherwise} \end{cases}$$

tells how many rows separate the source from the destination along the destination's column. The maximum number of hops travelled by the packet (assuming a deflection at every intermediate switch along the destination's column) is equal to:

$$h_{max}(s_s, s_d) = \delta_c(s_s, s_d) + (\delta_r(s_s, s_d) - 1) \times N_c + \delta_r(s_s, s_d)$$

In the worst case, $\delta_c(s_s, s_d) = N_c - 1$ and $\delta_r(s_s, s_d) = N_r - 1$. Consequently, regardless of the location of the source and destination switches, the number of hops is bounded by:

$$h_{max} = (N_r - 1) \times N_c + N_r - 2$$

which for a square grid equals $N - 2$.

Another merit of *VR* (besides bounding the number of hops per packet) is that every routing decision can be made on a per-packet basis. Indeed, there is a single "preferred" output port per every incoming packet. The only situation when this preference cannot be fulfilled concerns a packet arriving from the column—when there also is a column packet that arriving from the row. But there is no flexibility in such a case: the packet arriving from the row **must** be routed to the column. Thus, when a packet arrives from the row, we can tell its fate—as well as the fate of the other packet arriving from the column—without having to look at the other packet.

4 Buffer space

Assume that all switches transmit slots at exactly the same rate. With this assumption, the minimum amount of transient buffer space needed to implement the routing rules of MSN is one slot. This space

is needed to align the incoming slots that may arrive "out of phase," with the maximum misalignment bounded by the slot size. If the vertical routing rules are implemented in a synchronised fashion, the amount of buffer space needed to make them work is the same as for MSN. The same amount of transient buffer space is sufficient to implement *VR* in an asynchronous way—without aligning the incoming slots. To see this suppose that the beginnings of slots arriving from the row and the column are shifted by Φ bits—as shown in Figure 4. Clearly, we have $\Phi < \Delta$, where Δ is the total slot length expressed in bits.

As long as the slots arriving from the row are *remote*, the two streams of slots are separated. Let t_1 represent the moment of the first arrival of a column slot from the row (see Figure 4). Thus, the row slot s_0^r that arrived at time t_0 was a remote slot and it was relayed to the row, but slot s_1^r arriving at time $t_1 = t_0 + \Delta$ must be directed to the column. This redirection cannot be performed immediately (at time t_1) as the output port of the column is busy retransmitting slot s_0^c that arrived at the switch at time $t_0 + \Phi$. Consequently, slot s_1^r must be buffered before it can be relayed, for the amount of time equal to $\Delta - \Phi$. While the switch is waiting to redirect slot s_1^r to the column, it has nothing to transmit to the row. To avoid introducing a gap that would result in a bandwidth wastage, the switch will emit an empty slot to the row.[3] Before the first bit of slot s_1^r can be transmitted to the column (which will happen at time $t_1 + \Phi$), the transient buffer will be filled with the initial Φ bits of that slot. From then on, the buffer is emptied at the slot's arrival rate and its contents do not grow any more. At the same time $t_1 + \Phi$ slot s_1^c arrives from the column. This slot must be deflected to the row, but this cannot happen until the switch finishes inserting the empty slot to the row. Thus, slot s_1^c has to be buffered until time t_2, when the switch will be able to transmit its first bit. At that time, the length of the stored portion of s_1^c will reach $\Delta - \Phi$, which will result in the situation depicted in Figure 5.

The total amount of buffer space needed at time t_2 is equal to Δ. It is easy to see that as long as all subsequent slots arriving from the row will have to be directed to the column, nothing will change in the buffer space assignment. Every slot arriving from the row will be relayed to the column with the lag of Φ; similarly, a slot arriving from the column will be relayed to the row $\Delta - \Phi$ bits after its arrival. Thus, without loss of generality, we can assume that slot s_2^r (arriving at time t_2) is remote, i.e., it has to be relayed to the row. This slot will be inserted into the left portion of the buffer (Figure 5)

[3]Note that the switch can fill this empty slot with its own payload, according to the rules introduced above.

and at time $t_2 + \Phi$ its first bit will reach the end of this portion. At the same time a new slot (s_2^c) arrives from the column. The insertion point of the right portion of the buffer (containing the last column slot being deflected to the row) is now at a slot boundary. At this moment, the switch directs slot s_2^c to the column and removes the boundary between the two portions, effectively merging them into one contiguous area of size Δ.

From now on, for as long as the subsequent slots arriving from the row are remote, the column slots will be relayed directly to the column and the remote slots will be relayed to the row with a one-slot lag. The switch will be able to eliminate this lag (and cut through the buffer) when it detects an empty slot to be relayed to the row. If this happens before the next arrival of a column slot from the row, this arrival will get us to the already considered scenario. Otherwise, at the first slot boundary in the stream of slots arriving from the column, the buffer will be partitioned into two portions—as before.

In a realistic network, one cannot assume that all slots arrive at exactly the same rate. Independent clocks at different switches may tick at slightly different pace and even a single clock may drift a bit, depending on the external factors like temperature and air pressure.

In the asynchronous version of *VR*, the problem of compensating for the variability in the slot arrival rate can be solved locally with very limited additional buffer space independent of the network size and its transmission rate. As long as all slots arriving from the row are remote, the row traffic does not interfere with the column traffic and the two streams can propagate at different rates. With the first arrival of a column slot from the row, the redirection mechanism described above works correctly regardless of how badly the incoming slots are misaligned at the moment (the value of Φ). Then, for as long as the slots arriving from the row are addressed to destinations in the column, the two slot streams are again well separated and they don't have to proceed at the same rate. The only problem occurs at the subsequent arrival of the first remote slot from the row (slot s_2^r in Figure 4). As before, this slot is inserted in the left portion of the transient buffer with the intention of appending it at the left end of the right portion when the column traffic gets synchronised. In contrast to the previous idealistic scenario, this moment (the arrival of slot s_2^c) need not coincide with the moment when the first bit of slot s_2^r hits the end of the left portion of the transient buffer. If the latter event happens earlier than the resynchronisation of the column traffic, the row traffic will resynchronise with a lag shorter than Δ. Otherwise, additional space is needed to accommodate the outstanding bits of s_2^r until the arrival of the first bit of s_2^c. The amount of this additional space is bounded by $\Delta' = \Delta \times \frac{R_{max}}{R_{min}}$,

where R_{max} and R_{min} are the maximum and the minimum clock rates occurring in the network. In that case, the row traffic will resynchronise with a lag longer than Δ' but shorter than $2\Delta'$. Note that a lag longer than or equal to Δ can be reduced by Δ at the first occurrence of an empty slot in the outgoing row stream. A shorter lag will stay until the next arrival of a column slot from the row. If a slot boundary in the stream arriving from the column does not occur before the column slot arriving from the row hits the end of the lag buffer, the switch will issue an empty slot to the row—to avoid creating a gap. This will increase the amount of storage needed to handle the deflection by Δ, but in any case $2\Delta'$ bits of buffer space will suffice to process any routing scenario.

One can generalise *VR* by assuming that a switch can be equipped with some buffer space in excess of the indispensable minimum. With this approach, a packet arriving from the column need not be immediately deflected when it meets a column packet arriving from the row, but it can be stored for some time in the available buffer space. If additional buffer space is used, the asynchronous deflection rules for *VR* can be re-formulated as follows:

1. Incoming packets are stored in the transient buffer. If the buffer is empty, the packets cut through it. In particular, a retransmission of an incoming packet can be started before the packet has been completely received.

2. Whenever a column packet arrives at a switch from the row, the packet is tagged with a flag called *CR* (for "column route") indicating that the packet **must** be routed to the column. At the same time a counter (*CRCount*) is incremented by one to indicate that one packet arriving from the column can be deflected to the row.

3. An outgoing packet to the column is selected as the first packet that either has arrived from the column or has been tagged with the *CR* flag.

4. An outgoing packet to the row is selected as the first packet that has arrived from the row and has not been tagged with *CR*. If there is no such packet and the buffer space is fully used and *CRCount* is greater than zero, the first column packet not tagged with *CR* is deflected to the row.

Notably, with the right organisation of the buffer space, packets can be of variable length. A switch willing to insert its own packet into the network is allowed to do so whenever enough free buffer space is available at the switch.

5 Vertical routing in bidirectional MSN

Unidirectional *VR* doesn't have the *shortest path property*: even under very light load packets are not routed along the shortest paths to their destination because they are not allowed to change rows unless deflected. This disadvantage of *VR* disappears when we switch to the bidirectional grid.

Every switch in the bidirectional network has four input and four output ports. This increase in the connectivity degree of a switch inflates the size of the problem space for *RR*. When four non-empty slots arrive at the switch within one cycle, the switch has to consider all 24 possible assignments of these slots to the output ports and select the one that maximises the reduction of the total remaining number of hops for all these slots [2]. In the case of *VR*, the routing problem is only slightly more complicated than in the unidirectional version. To see why it has to be more complicated at all, imagine that a packet arrives at a switch on one of the two row ports. If the packet has reached the column of its destination, it has to be routed to the column. To fulfill the shortest paths postulate, the switch should route the packet in the direction that offers the fewer number of hops to the destination. Note, however, that the switch may not have this much flexibility. Namely, it may turn out that the output port for the preferred column direction is busy relaying another packet routed that way. No limited buffer space can offer a safe solution as it may happen that there is a continuous supply of packets addressed to destinations in the same column and preferring the same direction. Consequently, the switch may sometimes have to route a packet to the column of its destination in a non-preferred direction. On the other hand, once a packet is sent in some direction along a row or column, its direction should not change because we want to make sure that the packet never returns to a once-visited switch. If this happens (and if the routing rules are local) we cannot guarantee that the packet will reach its destination after a bounded number of hops.

One simple way to make the path of every packet as predictable as in the unidirectional network is to partition the bidirectional grid into two independent subnets, e.g., one subnet oriented right and down and the other oriented left and up. With this approach, a packet inserted into one subnet would be confined to that subnet for its entire path to the destination. This solution would correspond to two parallel unidirectional networks, each network operating according to the rules outlined in the previous section. The only added flexibility would be the selection of the subnet upon a packet's departure from the source. This would reduce the average length of the path travelled by a packet with respect to the unidirectional network, but the resultant routing scheme would still lack the shortest path property.

To take care of both ends, i.e., to guarantee packet delivery after a bounded number of hops and, at the same time, fulfill the shortest path property, we propose that each packet be confined to one subnet, but not from the very beginning of its path. Each packet carries a single binary flag, denoted AS, which tells whether the packet has been **assigned** to its subnet or not. When a packet is launched at the source, its AS flag is cleared meaning that the packet has not be assigned to a subnet. If the packet is a remote packet (i.e., its destination is in another column), the source will transmit the packet in the row direction that offers the smaller number hops to the destination column. When the packet reaches the column of its destination the routing switch will decide in which direction the packet should be retransmitted. Then the switch will set the AS flag of the packet to one.

For a packet with the AS flag set, its port of arrival determines the subnet to which the packet is confined. For example, if the packet has arrived from the row traveling to the right, it can only be relayed to the row in the same direction or, if its destination happens to be located in the same column as the routing switch, to the column in the direction down. Similarly, when such a packet is deflected from the column, it can only be re-transmitted in one specific direction to the row. Consequently, the routing rules for "assigned" packets are the same as in the unidirectional case.

An "unassigned" packet can only arrive at a switch from a row ring because as soon as a packet is directed to a column ring its AS flag is set. A switch receiving an unassigned packet determines the packet's preferred direction along the column. In the synchronised version of the protocol without transient buffers, the following scenarios are possible:

1. The other slot to be relayed to the column is empty, is assigned to the other direction, or is unassigned but it prefers the other direction. In such a case there is no conflict. The unassigned packet is relayed along its preferred direction.

2. The other slot to be relayed to the column is assigned to the preferred direction of the unassigned packet. In such a case, the unassigned packet is relayed in its unpreferred direction.

3. The other slot is unassigned but it prefers the same direction. In that case, the packet whose destination is closer is relayed along its preferred direction and the other packet is re-transmitted on the opposite port. This way the extra number of hops incurred by the conflict is minimised.

With the asynchronous version of the routing scheme, every switch is equipped with a buffer pool. The minimum amount of buffer space is two slots (packets) which is the same as the amount needed in

the synchronised version—for implementing slot alignment. A switch receiving an unassigned column packet is allowed to route it in the direction inconsistent with the port of its arrival (i.e., from one subnet to the other) only if the amount of free buffer space makes it possible to accommodate one maximum length packet. This is needed because the switch is not allowed to route an assigned column packet (that may arrive while the first packet is being re-transmitted) anywhere else but to the column along the direction determined by its subnet assignment (i.e., its port of arrival). Thus, when such a packet arrives, it must be stored until the output port becomes available. Consequently, if the buffer space at a switch is the minimum required to make the switch operable, the switch is not able to relay packets between the two subnets and the routing scheme lacks the shortest path property. This disadvantage does not occur in the synchronised version of the protocol.

6 Performance

We have investigated the performance of *VR* by simulation and compared it to the performance of *RR*. First, we built detailed models of our networks and protocols in SMURPH [7]. These models were very accurate and they reflected the operation of the networks and their protocols at the implementation level. In particular, they accounted for specific propagation delays of links and the limited accuracy of clocks used by different switches. The *VR* protocol was implemented in its unsynchronised version. In the case of *RR*, elastic input buffers were used to compensate for the variability in the slot arrival rate on different input ports.

Unfortunately, but not surprisingly, the detailed models exhibited rather high execution times, especially for larger networks and longer intervals of simulated time. Therefore, we decided to program simplified slotted models and compare the results produced by them with the results obtained from the elaborate models. These simplified models operated in a slotted fashion. Within one-slot time every switch would receive one slot from every input port and determine the fate of the set of outgoing slots—one outgoing slot per every output port. These outgoing slots were delivered to their next intermediate switches in the next cycle. Although the model was synchronous, the routing rules for *VR* were not, i.e., the incoming slots were processed independently. This way we in fact modelled the asynchronous implementation of *VR*—in a somewhat simplified manner.

The results (throughput versus hop count) produced by the simplified models were surprisingly close to those obtained from the accurate models—the difference was always much less than 1%—

and, consequently, we decided to base our investigation on the simplified models. Every single point of a performance curve was produced as an average of six independent experiments with different initialisation of the random number generator. This precaution turned out to be exaggerated because all the investigated networks and protocols were very stable in this respect. We have not observed a single case when basing a curve on just one (any) set of experiments rather than six independent sets would result in a perceptible difference.

We assumed that an undersaturated network has reached a steady state when its snapshot throughput, taken every 1000 cycles, ceased to change (grow) significantly for ten consecutive cycles. Let $T_0, \ldots, T_9$ be the last ten snapshot values of the throughput. The experiment was assumed to have reached its steady state when the following condition was fulfilled:

$$\frac{|(T_0 + \ldots + T_4) - (T_5 + \ldots + T_9)|}{T_0 + \ldots + T_9} < 0.0001$$

From this moment, the network was simulated for further 10,000 cycles and during that time the proper measurements were taken. Depending on the network size and load, the number of slots received within that interval at their final destinations was between $10,000$ and $40,000,000$.

Figures 6—11 compare the performance of *VR* and *RR* in square MSN networks for the uniform traffic pattern. The first three figures refer to unidirectional networks. As predicted, *VR* operating on such a network suffers larger delays than *RR*. Figure 6 shows the relationship between the observed throughput and average hop count for a network consisting of 64 switches organised into an 8×8 square grid. Four variants of *VR* are included, with a different amount of additional buffer space available at every switch. This amount was 0, 1, 2, and 4 slots. Although with 2 or 4 additional slot buffers per switch *VR* achieves a higher maximum throughput than *RR* (operating with zero extra buffers), it never gets close to *RR* with respect to the average hop count travelled by a packet on its way to the destination (although it provides an upper bound on this count). Note that adding buffers to *RR* increases the maximum throughput achieved by that protocol by $30 - 50\%$ (see [13]).

The difference between *VR* and *RR* applied to the unidirectional networks tends to become more pronounced as the number of switches grows. Note that the deflection penalty for *VR* grows with the increasing row length, whereas it is constant for *RR*. This is visible in Figure 7 which makes a similar comparison as Figure 6, but for a network consisting of 32×32 switches. The general pattern is the same as in Figure 6, but the gap between *VR* and *RR* is wider. In particular, regardless of the number of additional slot buffers per switch, the maximum throughput achieved by *VR* is always less

than that of RR with no extra buffer space.

Figure 8 relates the maximum throughput achieved by five variants of VR (as before the digit stands for the number of extra slot buffers per switch) to the maximum throughput of RR for square networks with different size. The very small distance between the curves for 4 and 8 buffers indicates that little, if anything, can be gained by increasing the number of buffers beyond 4.

The situation looks somewhat different for VR and MS implemented on bidirectional networks. Figures 9—11 illustrate the performance of the two protocols in the bidirectional environment. We couldn't find in the available literature ready results for RR with extra buffer space operating on bidirectional MSN grids. Therefore, these results (obtained with our simulators) are included in the figures.

Notably, the extra buffers used with RR do not improve its performance as drastically as it happens in the case of the unidirectional networks. With the improved network connectivity there are more alternative routes to the same destinations, there are fewer deflections, and the deflection penalty for the harmful ones is smaller. Thus, less can be gained by trying to reduce the number of deflection by storing the packets that cannot be relayed optimally in intermediate buffers. This reasoning applies to RR but not to VR, where the deflection penalty is relatively high and equal to the row length. Therefore, the performance of RR improves quite a bit with more buffer space. With a reasonable number of extra buffers (4 or 8), the protocol can perform only slightly worse (on the average) than RR with the same amount of additional buffer space.

7 Conclusions

We have discussed a deflection protocol applicable to MSN topologies which bounds the number of hops travelled by packets on their way to the destinations without discarding any packets. Besides this advantage, our protocol can be implemented in a completely asynchronous way, which simplifies the routing scheme both technically and in terms of its computational complexity.[4] The protocol is applicable to unidirectional and bidirectional networks, but the bidirectional architecture is preferred: the average performance of our proposed protocol is then comparable to the performance of the standard MSN deflection scheme.

Our protocol can also be implemented on networks with irregular topologies. Although it uses the

[4]The latter issue is important for bidirectional networks.

concept of a *column* and *row*, it doesn't require that all rows or columns be of the same length (consist of the same number of switches). The only information needed by an intermediate switch to relay a packet is the <*column*,*row*> pair of coordinates of the destination node.

References

[1] A. Acampora, S. Shah, and Z. Zhang. Performance analysis of hot-potato routing for multiclass traffic in multihop lightwave networks. In *Proceedings of IEEE INFOCOM'92*, pages 644–654, Florence, Italy, May 1992.

[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] W. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[4] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

[5] M. Decina, V. Trecordi, and G. Zanolini. Throughput and packet loss in deflection routing multichannel-metropolitan area networks. In *Proceedings of GLOBECOM'91*, pages 1200–1208, 1991.

[6] M. Decina, V. Trecordi, and G. Zanolini. Performance analysis of deflection routing multichannel-metropolitan area networks. In *Proceedings of IEEE INFOCOM'92*, pages 2435–2443, Florence, Italy, May 1992.

[7] W. Dobosiewicz and P. Gburzyński. SMURPH: An object oriented simulator for communication networks and protocols. In *Proceedings of MASCOTS'93, Tools Fair Presentation*, pages 351–352, Jan. 1993.

[8] A. Greenberg and J. Goodman. Sharp approximate models of adaptive routing in mesh networks. In O. Boxma, J. Cohen, and H. Tijms, editors, *Teletraffic Analysis and Computer Performance Evaluation*, pages 255–270. Elsevier Science Publishers B.V. (North-Holland), 1986.

[9] A. Krishna and B. Hajek. Performance of shuffle-like switching networks with deflection. In *Proceedings of IEEE INFOCOM'90*, pages 473–480, San Francisco, CA, June 1990.

[10] N. Maxemchuk. The Manhattan street network. In *Proceedings of GLOBECOM'85*, pages 255–261, 1985.

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

[12] N. Maxemchuk. Distributed clocks in slotted networks. In *Proceedings of IEEE INFOCOM'88*, pages 119–125, 1988.

[13] N. Maxemchuk. Comparison of deflection and store-and-forward techniques in Manhattan-street network and shuffle-exchange networks. In *Proceedings of IEEE INFOCOM'89*, pages 800–809, 1989.

[14] N. Maxemchuk. Problems arising from deflection routing. In Pugolle, editor, *High Capacity Local and Metropolitan Networks*, pages 209–233. Springer Verlag, 1991.

[15] N. Maxemchuk and R. Krishnan. A comparison of linear and mesh topologies–DQDB and the manhattan street network. *IEEE Journal on Selected Areas in Communications*, 11(8):1278–1301, Oct. 1993.

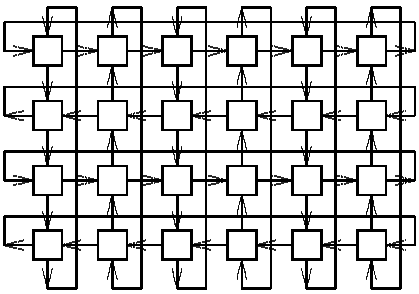

Figure 1: A 6×4 unidirectional Manhattan-street network.

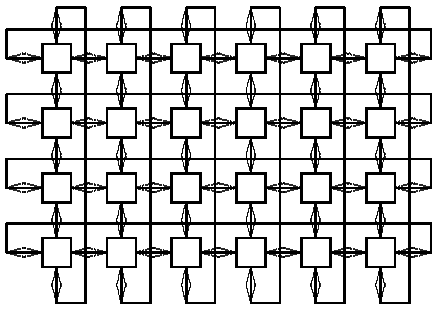

Figure 2: A 6×4 bidirectional Manhattan-street network.

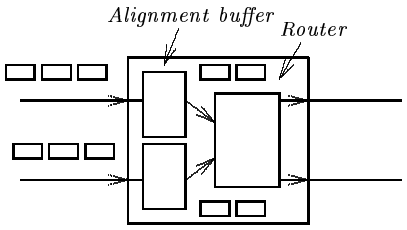

Figure 3: Alignment buffers in a synchronous unidirectional MSN switch.

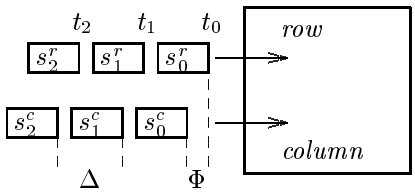

Figure 4: Slots arriving out of phase.

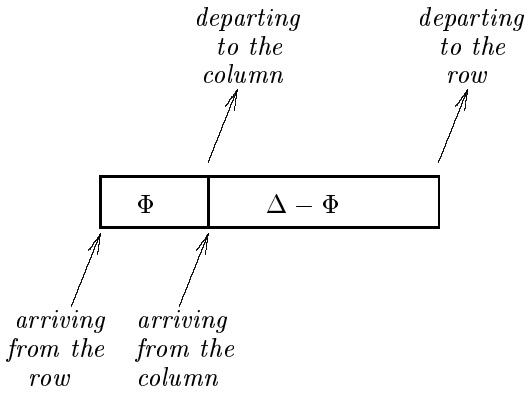

Figure 5: Buffer space assignment after processing the first column slot arriving from the row.

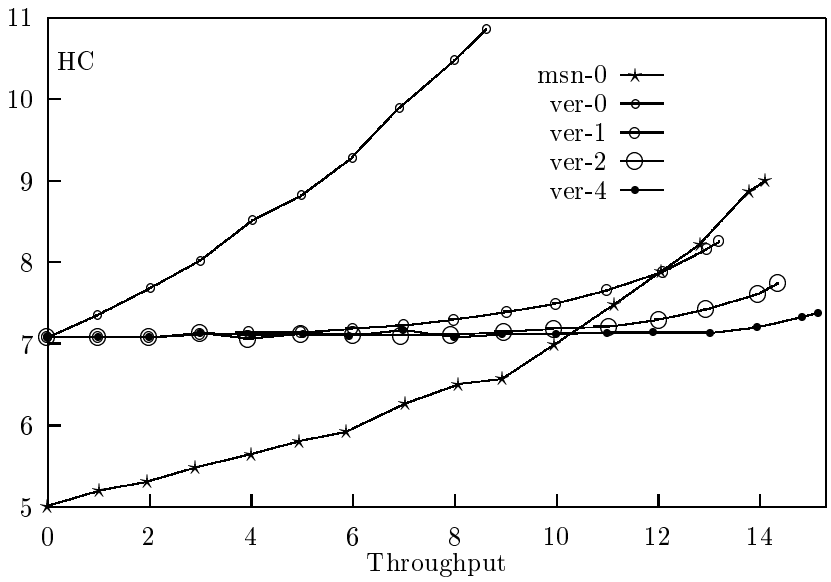

Figure 6: Connectivity–2 networks, 8×4 switches.

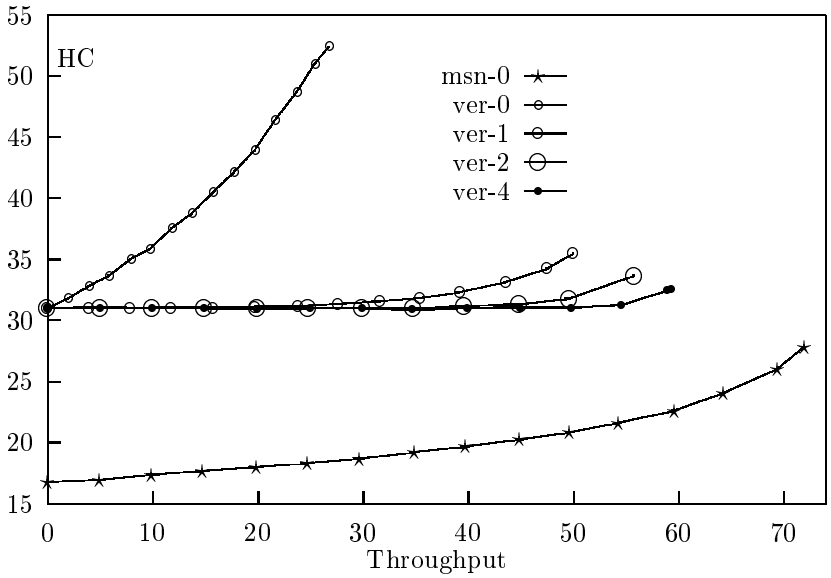

Figure 7: Connectivity–2 networks, 32×32 switches.

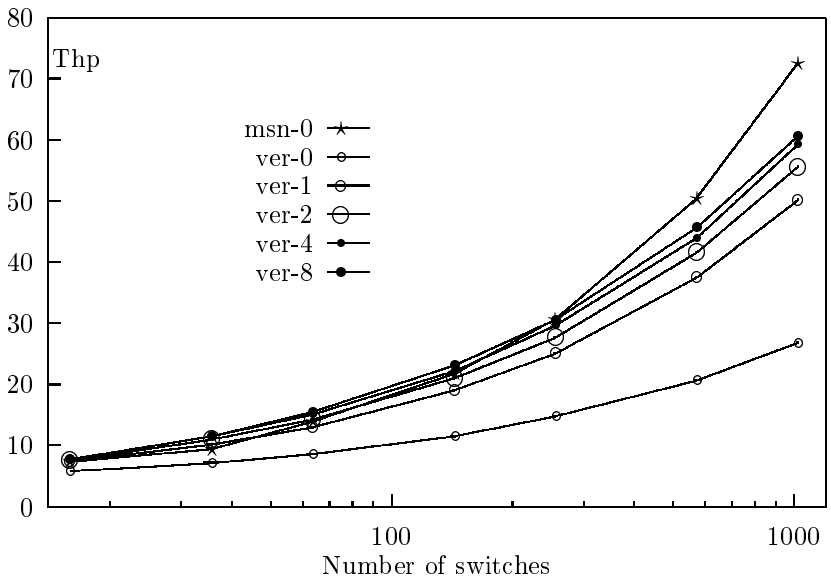

Figure 8: Connectivity–2 networks, maximum throughput versus network size.

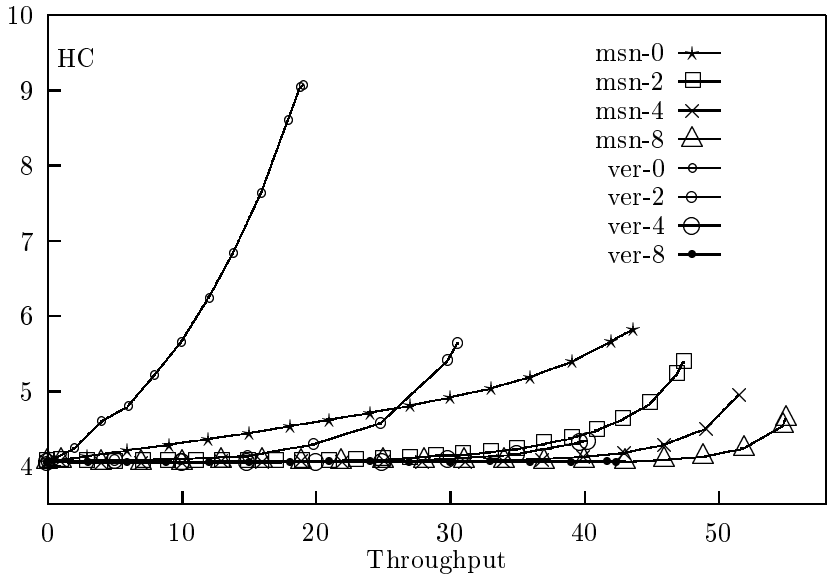

Figure 9: Connectivity–4 networks, 8×4 switches.

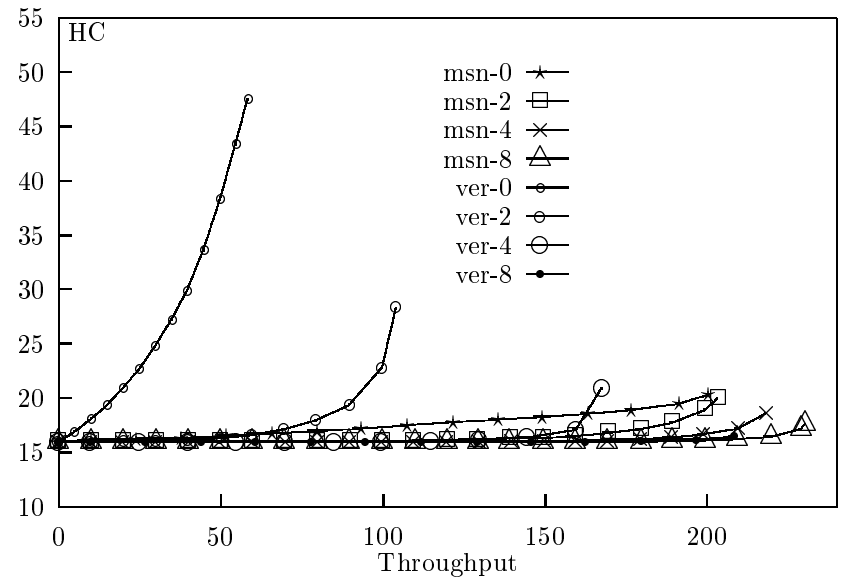

Figure 10: Connectivity–4 networks, 32×32 switches.

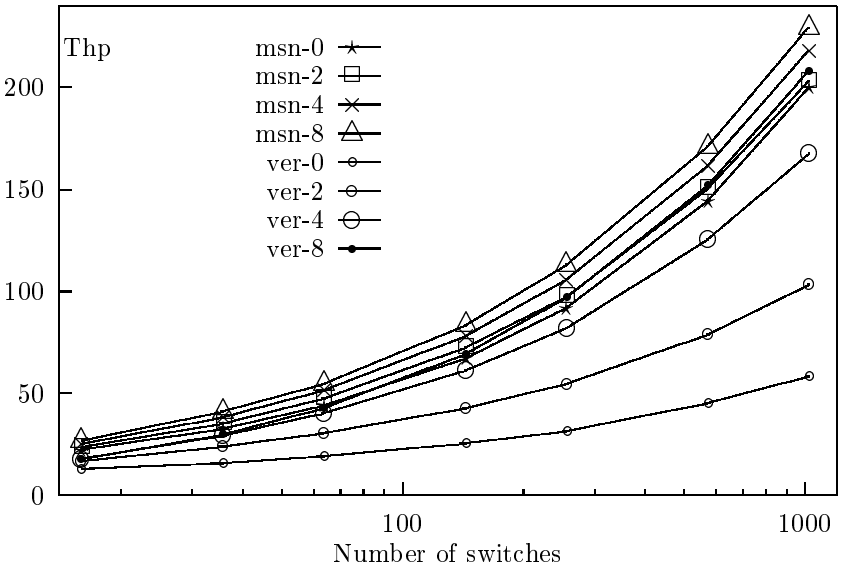

Figure 11: Connectivity–4 networks, maximum throughput versus network size.