Asynchronous Deflection with Transient Buffers

Wladek Olesinski and Pawel Gburzynski
Department of Computing Science
615 GSB, University of Alberta, Edmonton, AB, Canada T6G 2H1
e-mail: [wladek,pawel]@cs.ualberta.ca
URL: http://www.cs.ualberta.ca/~[wladek,pawel]

Abstract

We investigate experimentally several routing algorithms for asynchronous deflection networks, i.e., ones that operate in an unslotted manner. We determine the impact of an extra input buffer space on the quality of routing decisions. Finally, we compare the performance of asynchronous and synchronous deflection networks.

1. Introduction

Traditionally, deflection networks have been viewed as slotted systems. In such a network, packets arrive at a switch in orchestrated batches and are examined simultaneously. This way, a routing decision deals with all packets that arrive at the switch within the current *slot* and account for the combined preferences of all these packets [5, 6, 8, 9, 10]. This approach poses some implementation problems: in a realistic environment, transmission rates of different switches may not be exactly the same, and the synchronization of the network may be difficult. To keep the slot arrival rate steady, a switch may need additional buffer space and/or the network may have to resort to complicated backpressure mechanisms [7]. As the performance of such techniques depends on the (normalized) propagation delays across the network, they do not scale very well to the increasing network size and/or transmission rate. The fixed packet (slot) size is another drawback.

Alternatively, one may consider asynchronous networks, in which packets (not necessarily of the same length) may arrive at a switch spontaneously [2, 3, 4]. With the simplest implementation of this idea, the switch will make a routing decision for one packet at a time, as soon as the packet's header (destination) has been recognized. Asynchronous routing eliminates the drawbacks of synchronous networks but it tends to make significantly worse routing decisions, particularly at low connectivities (e.g., 2×2 switches). On the other hand, as pointed out in [2], the quality of asynchronous routing decisions improves with the increasing connectivity of a switch.

To see why in some cases asynchronous deflection routing may perform significantly worse than synchronous routing, consider the following scenario at a 2×2 switch. Packet P_1 arrives at the switch and both output ports are idle. Assume that the packet doesn't clearly prefer any output port, i.e., the switch is free to choose any of them to relay the packet. Suppose the switch chooses port p_1. A moment later, while P_1 is still being relayed, another packet, P_2, arrives at the same switch. This time P_2 prefers p_1, i.e., the port that is being occupied by P_1. There is no choice but to deflect the new packet, although if both packets were available at the same time, the switch would be able to avoid the deflection. Note that if another packet, say P_3, arrives while P_2 is being deflected, the switch will have absolutely no flexibility as to the fate of that packet. Consequently, it may get into a sustained scenario of unbounded duration in which all packets arriving at the switch are continuously being deflected. This phenomenon is particularly harmful in networks with small connectivities and under heavy load.

The above problem can be alleviated by equipping the switch with some transient buffer space. This space will make it possible to postpone a routing decision for a while, until it can be confronted with the routing decisions for other packets. In this paper we discuss a few possible implementations of this idea and their impact on the performance of asynchronous deflection routing.

2. The model

We consider the torus topology (Figure 1), which is the standard configuration for Manhattan-street networks (MSN) [5]. Notably, other topologies that we tried (including some biased topologies, e.g., a triangle) produced performance results highly consistent with those obtained for the torus. In relative terms, the results presented in our paper should hold for many other reasonable topologies.

Every switch is equipped with some buffer space; the

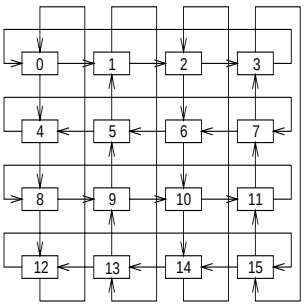

Figure 1. Torus network, size 16, connectivity 2

way of using these buffers is part of the routing strategy. The network operates in an asynchronous manner, in a way described in the introduction. Packets are not explicitly aligned at the switch, they may be of different length and they are allowed to arrive at any time.

Every switch has a *host* capable of contributing traffic to the network. The traffic is uniform which means that all sources and destinations are equally probable. Every host generates packets according to the Poisson distribution with a given mean. The load parameter determines how many new packets appear in the entire network during the time needed to transmit a single packet. To reduce the number of parameters and eliminate irrelevant degrees of freedom, we keep the packet length fixed, although the routing rules never pose this requirement. Our experiments carried out for variable packet length have produced similar results to those where the packet length was fixed. The actual packet length is immaterial: what matters is the ratio of the buffer size at a switch to the (average) packet length.

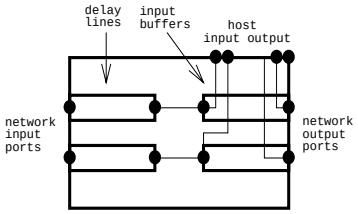

Figure 2. Simplified switch model

To avoid disrupting the relayed traffic by its own packets, every switch employs delay lines separating its input ports from the routing fabric (including the buffer space). If a switch whose host is backlogged finds at least one delay line free (see Figure 2), it removes the first outgoing packet from the host's queue and inserts it into the input buffer—as if the packet arrived from the network on the corresponding input port. With the delay line, the switch makes sure that the packet transmitted by the switch will not collide with a packet entering the buffer from the input port. The minimum size of a delay line must be L, where L is the maximum length of a packet.[1] With this approach, the switch is never forced to drop a packet, which we view as a fundamental property of deflection routing. If an incoming packet is addressed to the current switch (i.e., its host), the switch receives the packet and removes it from the network.

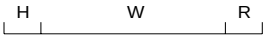

Figure 3. Input buffer

The input buffer consists of three logical parts corresponding to three stages of packet processing—as shown in Figure 3.

- Part H absorbs the packet for the amount of time needed by the switch to recognize the packet's header, more specifically, the packet's destination. A routing decision regarding the packet can only be made if its destination is known.
- Part W is the "waiting room." It allows the switch to defer the routing decision for some time, e.g., to coordinate it with routing decisions for other packets. This part may be empty, in which case the routing decision for the packet must be made as soon as the first bit of the packet has passed the H part.
- Part R corresponds to the amount of time needed by the switch to actually make a routing decision.

In the sequel, H, W, and R will be used to denote the amount of time spent by a packet in the three components of the input buffer. As is customary in homogeneous networks, we will use one bit as a time unit. We assume that H and R are fixed, but W can be varied. If $W = 0$, the total length of the input buffer is the shortest possible. Note that the delay lines used by the hosts are in fact separate buffers.

A routing decision made at time t is performed $H + W$ bits after the first bit of the first packet affected by this decision entered the input buffer. Let us denote this packet by P. If $W \neq 0$, the routing decision may involve packets that arrived later than P, assuming that their destinations have been recognized (the packets have passed through H) before time t. We say that such a routing decision is *initiated* by packet P.

[1] An internal switch, e.g., one that is not connected to a host, need not be equipped with delay lines.

Our primary performance criterion is the maximum throughput achievable by the network expressed as the total number of received bits per one bit of time. A single simulation experiment was run as long as the differences among the four consecutive snapshot values of throughput taken at intervals corresponding to 50 times the maximum propagation distance in the network were more than 1%.

For the results presented in this section, the size of a simulated network was $N = 100$ switches, packet length $L = 1024$ bits, $R = H = 102$ bits (i.e., 1/10 of L), link length $l = 4000$ bits (almost 4 packets), and delay line length $d \geq L$ (this was one of the variables).

Note that R and H are practically irrelevant—they just slightly inflate the propagation distance between a pair of neighboring switches. As it turns out (Section 3.2), sometimes it makes sense to increase the length of the delay lines (d) above the minimum (L); thus, this parameter was varied in some experiments. Retaining the torus topology, we have investigated networks with connectivities $k = 2, 4, 8$. If $k = 4$, all links in Figure 1 are bidirectional; with $k = 8$, four diagonal bidirectional links are added to every switch. The most important variable was W, i.e., the "useful" length of the input buffer.

3. The standard algorithm

The most natural routing algorithm routes the packets arriving on the same input port in the order of their arrivals. Thus, a packet can only be routed if no other packet precedes it in the input buffer. If $W > 0$, it is possible that more than one packet will compete for the same free outgoing port. In such a case, the *locally optimal* routing decision [1, 2] assigns free output ports to all competing packets in such a way that the sum of the shortest distances to their destinations is minimized. The routing algorithm operating according to the above rules will be called *standard*.

3.1. Performance of the standard algorithm

Figure 4 shows the maximum throughput achieved for a given connectivity and W.

One would expect that the maximum throughput will increase with increasing W, because the quality of a routing decision will tend to improve. However, this only holds for some ranges of W. In particular, if $k = 2$, the throughput for $W = 1024$ is higher than for $W = 0$, but it drops for larger W. If $k = 4$, the throughput reaches its maximum for $W = 256$ and then drops. If $k = 8$, the throughput reaches its peak for an even smaller value of $W = 128$.

Let us start with the 2-connected network. To understand why the throughput drops when $W > L$, consider the scenario illustrated in Figure 5. Lines A, B, and C show different left (input) boundaries of the buffer, corresponding

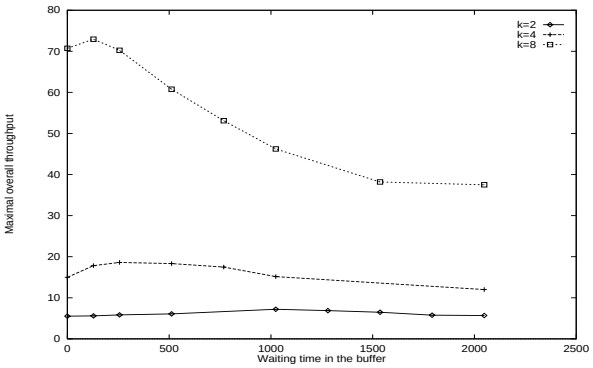

Figure 4. Maximum overall throughput vs waiting time W in the buffer for different connectivities

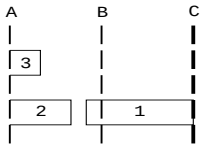

Figure 5.

to different values of W. C which is also the right boundary of the buffer, corresponds to $W = 0$. Without missing anything important, we can assume that $R = H = 0$.

Consider case A. Packet 1 arrives at the end of the buffer (C) first and competes with packet 3 (it does not compete with 2 because 2 is behind 1 in the same buffer). Thus, we have two competing packets and neither of the output ports is busy (notation $C(2, 0)$). After a while, packet 2 reaches C. There is no packet for it to compete with (the fate of 3 has already been determined), and there is only one available output port (the one freed by packet 1). We denote this situation by $C(1, 1)$. The history of the routing decisions is $C(2, 0)$, $C(1, 1)$.

In case B, when packet 1 reaches C, packet 3 is not yet available; thus, the first routing decision is $C(1, 0)$. When packet 2 gets to the end of the buffer, it will compete with 3 ($C(2, 0)$). The two routing decisions are $C(1, 0)$, $C(2, 0)$.

When $W = 0$ (case C), all three packets will be routed independently: $C(1, 0)$, $C(1, 0)$, $C(1, 1)$.

The best chance for all packets to be routed over their preferred links is in case B. The first packet gets its prefered port and at least one (but possibly both) of the remaining two packets gets its prefered port as well.

Next is case C. The reason why it is worse than B is that the last packet (3) must be routed via the only available

port, so its chances for getting the prefered port are smaller than in the previous case.

Case A (the largest buffer) turns out to be the worst of them all! First we have a competition ($C(2,0)$) in which one packet gets its prefered port for sure, but the other one may be deflected. This is followed by $C(1,1)$—a no-choice relay—in which packet 3 may also suffer a deflection.

We can see that the best routing decisions in the above scenario are made when W is greater than 0 but smaller than L. Analysis of of frequencies of particular types of routing decisions under heavy load indicates that indeed, when $W = 0$, the percentage of potentially bad routing decisions $C(1,1)$ is higher than for $W = 1024$. Then as W increases, the frequency of $C(1,1)$ increases as well. At the same time, the percentage of good decisions $C(1,0)$ decreases.

When the input buffer is longer than L, it is more likely that a packet P_2 following packet P_1 that initiates a routing decision will be ahead of another packet P_3 competing with P_1. In this scenario, P_2 will have only one link to be routed over. If buffers are smaller, this situation is less likely to happen.

The same phenomenon is observed for connectivities higher than 2. Note that the throughput for $k = 4$ and $k = 8$ starts to drop before W reaches L (Figure 4). The reason for this is given below.

One observation that one can make is that a smaller W causes splitting a large routing decision into several smaller decisions involving fewer packets. In the extreme case of $W = 0$, every routing decision concerns only one packet (unless two packets arrive at the switch at exactly the same time). Then the sustained deflection scenario mentioned in the introduction is likely to occur, especially if $k = 2$. But even in a 4-connected network, if four packets are routed to the output ports and then another packet arrives and reaches the end of the input buffer, it may not have too many output ports to select from. In particular, if it follows the packet that initiated the routing decision, there will be only one port available. If the buffers are not very long, fewer packets (e.g., 2) may be routed at a time, so packets following them will be more likely to find more output ports available. But this can only happen if consecutive packets arriving at the switch over the same link are separated by some gaps. If every packet is immediately followed by another, W becomes irrelevant because then (ignoring the extremely rare cases of two or more packets arriving at exactly the same time) every packet is routed individually. This also explains why the throughput in a saturated network tends to drop: the benefits of a nonzero W tend to disappear when packets become more closely spaced.

To understand the reason why the maximum throughput for $k = 4$ and $W = L$ is even worse than for $W = 0$, consider the scenario shown in Figure 6.

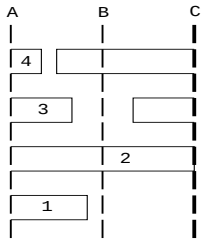

Figure 6.

In case A, W is large but still somewhat smaller than L. First, packet 2 competes with packet 1 in $C(2,2)$. Note that it cannot compete with with 3 or 4 because these packets are preceded by packets that are being routed. Then, the routing decisions for packets 3 and 4 are both $C(1,3)$. The history is $C(2,2)$, $C(1,3)$, $C(1,3)$.

In case B, packet 2 is routed alone in $C(1,2)$. Then, 1 competes with 3 in $C(2,2)$. Finally, packet 4 finds three outgoing ports busy, so the routing decision is $C(1,3)$. The history is $C(1,2)$, $C(2,2)$, $C(1,3)$.

With $W = 0$, packet 2 is routed as before ($C(1,2)$). Then, packet 1 is routed alone in $C(1,2)$ (the predecessor of packet 3 has left the buffer, so again two output ports are available). Finally, packets 3 and 4 reach C and each of them finds three ports busy. The history is $C(1,2)$, $C(1,2)$, $C(1,3)$, $C(1,3)$.

In the last case, we have two no-choice relays $C(1,3)$. In the second case, the situation is better: there is only one no-choice routing, one "reasonable" routing ($C(1,2)$) and one "passable" decision ($C(2,2)$). In case A, we have two no-choice relays $C(1,3)$ and one "passable" decision $C(2,2)$. The routing history in this case (with the largest buffer) is surprisingly the worst!

3.2. Standard algorithm with longer delay lines

In the previous section, we noticed that small gaps between consecutive packets arriving on the same link tend to worsen the quality of routing decisions. One natural way to increase these gaps is to increase d—the length of the delay lines—beyond L. As we increase packet spacing, the maximum throughput achievable by the network will tend to decrease because of the incurred bandwidth wastage. However, the improved quality of routing decisions may compensate for this loss and outweigh it. Clearly, there must be a middle ground somewhere.

Let F be the fraction by which the length of the delay line d is increased above L. For example, if $F = 0.5$, $d = L + 0.5 * L$. Figure 7 shows the saturated throughput for different connectivities k and for the buffer size

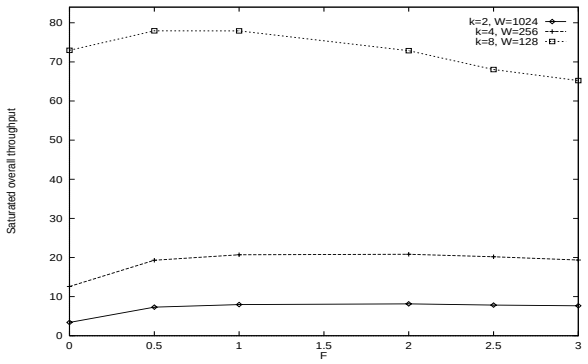

Figure 7. Saturated throughput vs F for different connectivities

W for which the observed throughput (see Figure 4) was maximum[2]. This figure suggests that increasing d up to $3L$ ($F = 2$) in a network with $k < 8$, and $2L$ ($F = 1$) for $k = 8$ results in some improvement in the achievable throughput, with the most significant improvement observed for $k = 8$ and $F = 1$. Again, analysis of frequencies of routing decisions for different F confirms the validity of the above results.

4. The "quick" algorithm

One possible way to improve the performance of the "standard" algorithm is to speed up routing in those cases when a packet's fate becomes known before the packet has reached the end of the input buffer. Such a packet will be immediately forwarded to the selected output port, without having to go through the entire buffer. Besides decreasing the amount of time spent by the packet in the buffer, this approach will also improve the chance that the next packet will find more output ports free. One drawback of this solution is a more complicated organization of the buffer space and, consequently, more expensive switch hardware.

4.1. Comparison with the standard algorithm

We have observed that in the case in which the delay lines were L bits long, the throughput achieved by the quick algorithm is significantly higher than that of the standard algorithm. For $k = 2$ and 4, the improvement factor exceeds 2, and it comes close to 2 for $k = 8$.

It is interesting to note that the throughput increases in a wider range of the buffer size W. For example, if $k = 4$, the throughput for $W = 1024$ is higher than the throughput for $W = 256$ (which is the optimum buffer size for the standard algorithm). This is because many packets do not have to go through the entire buffer and, from their perspective, the buffer appears shorter than it is.

[2] Saturated throughput in an asynchronous deflection network tends to be lower than maximum — the reason of this is explained in [2].

4.2. Longer delay lines

A natural next step is to see how the quick algorithm will perform when we increase d—the length of the delay lines—which enforces packet spacing. This idea proved beneficial for the standard algorithm (Section 3.2).

The results of this experiment (not shown here) indicate that surprisingly, increasing d does not help at all. The best results were consistently observed for $F = 0.0$, i.e., $d = L$. To understand why, consider the following example illustrated in Figure 8.

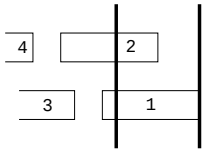

Figure 8.

The input buffer is visualized as the space between the two vertical lines. The presented scenario is quite frequent in a heavily loaded network. Packet 1 initiates a routing decision and competes with packet 2 ($C(2,0)$). When packet 3 reaches the end of the buffer, it will find both outgoing ports free—packet 2 will have been removed by then because it does not have to pass the entire buffer. The routing decision initiated by packet 3 will be again $C(2,0)$.

This is exactly what we wanted to achieve in the standard algorithm by increasing d. We conclude that this mechanism is already present in the quick algorithm by the virtue of the fast removal of packets whose fate is known. Consequently, we cannot gain much by increasing d, but we lose some throughput due to the bandwidth wastage.

Analysis of the frequencies of the routing decision types for connectivities 2, and 4 shows that for $k = 2$, increasing F (and thus d) has almost no effect on the number of worst routing scenarios $C(1,2)$. The routing decisions are not improved and the number of packets in the network decreases.

Throughput differences for $k = 4$ are smaller. This fact is also reflected in the number of the worst routing scenarios ($C(1,3)$) which drops much faster than for $k = 2$.This improvement counteracts the throughput deterioration caused by the smaller number of packets that may be inserted into the network.

5. The complete algorithm

This time we will try to improve the quality of a routing decision by postponing it until the last possible moment. With this approach, every packet is forced to go through the entire input buffer (as with the standard algorithm). Whenever a packet reaches the end of the input buffer and must be routed, the routing decision will take into account all packets currently visible to the switch, even if some of them have been already assigned to output ports by a previous routing decision.

This algorithm is complex. When the connectivity is high and input buffers are long, many packets may compete at the switch at the same time. Since solving the routing problem practically boils down to examining every possible assignment of packets to output ports, the algorithm may turn out to be infeasible for $k = 8$ or higher. This is why we confine our discussion to connectivities 2 and 4.

One might expect that the throughput achievable by this algorithm will never decrease with increasing W. The more packets are considered by the routing algorithm, the better its outcome should be. However, as shown in Section 3, a packet initiating a routing decision often finds all $k-1$ ports busy. It has only one outgoing link to be routed over, which means that when the load is high (and gaps between packets are small), the routing algorithm is doomed to perform poorly regardless of the number of considered packets.

k=2			
W	algorithm	max.	saturated
1024	C	5.501	3.756
1024	S	7.212	5.543
1536	C	4.528	3.948
1536	S	6.471	5.349
2048	C	5.416	4.220
2048	S	5.687	4.991
k=4			
768	C	18.911	15.413
768	S	14.981	12.897
1024	C	18.410	16.151
1024	S	15.156	11.825
2048	C	16.249	13.341
2048	S	12.004	9.682

Table 1. Maximum and saturated throughput of the standard (S) and complete (C) algorithms for different W and k.

As we can see in Table 1, the throughput in the network with $k = 2$ is even worse with the complete algorithm than with the standard one. To understand why, let us consider a scenario (in a connectivity-4 network) that is favorable for the complete algorithm (see Figure 9).

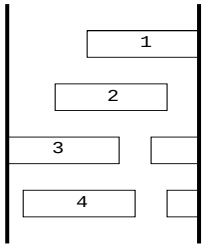

Figure 9.

In the standard algorithm, packet 1 would compete with packet 2 for two free ports. The routing decision would assign these two packets to the ports such that the cost C (combined deflection penalty) of this decision would be minimal. In the complete algorithm, packets 3 and 4 also take part in the routing decision. They may influence the assignment of packets 1 and 2 such that the cost of routing these two packets is higher than C but the cost of routing all four packets is reduced. This way, as more packets are considered, a better routing decision can be made.

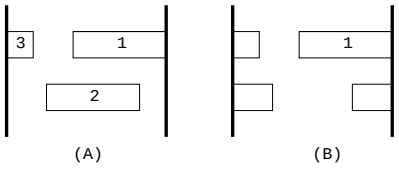

Figure 10.

Note, however, that this kind of improvement is only possible in a network with a connectivity higher than 2. If $k = 2$, only two scenarios can occur (Figure 10). In scenario (A), two packets compete for two free output ports. With the standard algorithm, the port assignment would be optimal from the viewpoint of packets 1 and 2. With the complete algorithm, including packet 3 in the routing decision might theoretically decrease the routing cost for all three packets by changing the assignment for 1 and 2. However, since packet 3 follows 1, 3 will only be routed after packet 1 leaves the buffer. This means that packet 1 can be routed over its best link without considering packet 3. Of course, the same situation occurs when 3 follows 2.

By the same token, in scenario (B), the number of packets considered in the routing decision does not matter. Packet 1 has only one port over which it can be routed.

Similar scenarios may occur in a network with connec-

tivity 4. However, in such a network, the complete algorithm is at least given an opportunity to exhibit its feature. In some range of W, this opportunity outweighs the negative impact of the malicious scenarios.

The effect of increasing W is similar to what we have seen for the standard algorithm (Section 3), i.e., the throughput increases and then starts to decrease. It is noticed that the range of W in which the throughput increases is larger than in the standard algorithm.

When we compared the frequencies of routing decisions for particular W in this algorithm with their counterparts in Section 3.1, we noticed that for $k = 2$, the percentage of potentially bad decisions was higher in the complete algorithm than in the standard one. This was reversed for $k = 4$, in line with our earlier reasoning.

6. Comparison of asynchronous and synchronous routing algorithms

k=2			
W	algorithm	max.	saturated
1024	Q	16.901	16.825
2048	C	5.416	4.220
1024	S,F=2.0	8.157	8.049
1024	S	7.212	5.543
k=4			
1024	Q	42.797	42.193
768	C	18.911	15.413
256	S,F=2.0	21.286	20.721
256	S	18.605	15.456
k=8			
512	Q	132.436	132.124
128	S,F=1.0	77.935	77.933
128	S	72.957	72.757

Table 2. Maximum and saturated throughput for different W and k—comparison of asynchronous schemes (S-standard, Q-Quick, C-Complete).

Table 2 compares all asynchronous routing algorithms presented in the preceeding sections. The waiting time W was chosen to maximize the throughput in each case.

As expected, the lowest throughput in the network with $k = 2$ is obtained when the complete algorithm is used. We showed in Section 5 that for connectivity 2, routing decisions can only be worsened by considering all packets competing at the switch.

A higher throughput is achieved by the standard algorithm, particularly if the delay line is longer than L. We showed in Section 3.1 that increasing the gaps between the packets improves the routing decisions made by this algorithm.

k=2			
W	algorithm	max.	saturated
	synch.	18.306	
1024	Q	16.901	16.825
2048	C	5.416	4.220
1024	S,F=2.0	8.157	8.049
k=4			
	synch.	52.386	
1024	Q	42.797	42.193
768	C	18.911	15.413
256	S,F=2.0	21.286	20.721
k=8			
	synch.	165.838	
512	Q	132.436	132.124
128	S,F=1.0	77.935	77.933

Table 3. Maximum and saturated throughput for different W and k—comparison with synchronous routing.

Table 3 compares the maximum throughput in asynchronous and synchronous torus networks. The amount of buffer space in the synchronous networks was equal to the minimum required for slot alignment, i.e., one slot per input port.

We may see that the throughput in the synchronous networks exceeds the throughput in their asynchronous counterparts, even if the best (quick) algorithm is used. When the load is sufficiently high, some packets will always be "stuck" behind the packets whose fate has been already determined. Their chances for being routed via their prefered links are smaller because there are fewer free output ports to choose from.

Visible as it is, the difference between the quick algorithm and synchronous routing is much smaller than between the standard algorithm and the quick one. This difference amounts to about 7% for $k = 2$, 19% for $k = 4$, and 25% for $k = 8$.

7. Conclusions

Our results indicate that among the asynchronous routing algorithms, the best are those that assure that a packet competing at a switch will perceive a large number of free ports. That is why, the standard algorithm with long delay lines and the quick algorithm give the best results.

We have also shown that some solutions may produce counterintuitive results, e.g., increasing the amount of buffer space may deteriorate the network's performance. For this reason, the selection of the buffer size in an asynchronous network should be made with caution. Generally, the higher the connectivity, the smaller the recommended length of the buffer.

Although our experiments have been carried out for Poisson traffic, one should expect that the performance of the routing algorithms for other traffic patterns will be similar. As we have noticed in [11], deflection networks are not very sensitive to changing traffic patterns.

Among the routing algorithms discussed in this paper, the standard algorithm is the simplest, and the quick algorithm (which is only slightly more complicated) offers the highest throughput. This throughput approaches that achievable with synchronous routing; thus, asynchronous deflection appears to be a feasible alternative, especially in high-speed applications, where synchronous deflection may be difficult to implement.

References

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

[2] P. Gburzynski and J. Maitan. Deflection routing in regular MNA topologies. *Journal of High Speed Networks*, 2(2):99–131, 1993.

[3] J. Maitan, L. Walichewicz, and B. Wealand. A new low cost communication scheme for military applications. In *Proceedings of Milcom'90*, Monterey, CA, 1990.

[4] J. Maitan, L. Walichiewicz, and B. Wealand. Integrated communication and information fabric for space applications. In *AIAA/NASA Second International Symposium on Space Information Systems*, pages 1175–1184, Sept. 1990.

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

[8] 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.

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

[10] 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.

[11] 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.