University of Castilla-La Mancha

A publication of the

Department of Computer Science

Switch Scheduling in the Multimedia Router (MMR)

by

M. B. Caminero, F. J. Quiles, J. Duato, S. Yalamanchili, D. Love

Technical Report **#DIAB-00-02-10** February 2000

This work is the extended version of the paper accepted for presentation in the IPDPS'2000 Symposium, held in Cancun (Mexico) in May, 2000

J. Duato is with the Dept. of Information Systems and Computer Architecture, Universidad Politécnica de Valencia, P.O.B. 22012, 46071 - Valencia

S. Yalamanchili and D. Love are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0250, U.S.A.

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Switch Scheduling in the Multimedia Router (MMR)

M. B. Caminero, F. J. Quiles, J. Duato, S. Yalamanchili, D. Love

Abstract

The primary objective of the Multimedia Router (MMR) project is the design and implementation of a single-chip router optimized for multimedia applications. The router is targeted for use in cluster and LAN interconnection networks which offer different constraints and therefore differing router solutions than WANs. This paper describes and evaluates a switch scheduling algorithm based on a priority biasing scheme for dynamically updating the priorities of the connections established through the router. Unlike existing schemes that simply use the age of a flit as its priority, the novel feature of the proposed approach is that the priority is biased using the measured quality of service (QoS) values for the connection. Furthermore, the structure of the switch scheduling algorithm is motivated by opportunities for pipelined and concurrent operation so that scheduling decisions could be made at switching speeds. The performance of two of the many possible biasing functions is evaluated.

1 Introduction

The primary objective of the Multimedia Router (MMR) project is the design and implementation of a single-chip router optimized for multimedia applications and targeted for use in clusters and LANs. The goal is to provide architectural support to enable a range of quality of service (QoS) guarantees at latencies comparable to state-of-the-art multiprocessor cut-through routers. To achieve this goal we must provide solutions to many difficult hardware resource management and scheduling problems while constraining required resources to permit effective single-chip implementations. This paper focuses on one specific problem, namely, the ability to make effective switch scheduling decisions at router switching speeds.

2 Background and Motivation

The continuing rapid decrease in the cost of both processor and network components has led to a tighter integration of computation and communication. The result has been an explosion of network-based applications characterized by the processing and delivery of continuous data streams and dynamic media [6, 12] in addition to servicing traditional static data. Numerous examples can be drawn from web-based applications, interactive simulations, gaming, visualization, virtual meetings, and collaborative design environments. The changing nature of the workload and cost/performance trade-offs have prompted the development of a new generation of scalable media servers structured around networks of workstations (NOWs) interconnected by high-speed system/local area network (SAN/LAN) fabrics. Systems such as real-time media servers need to service hundreds and, possibly thousands, of clients typically each with their own distinct quality of service (QoS) requirements, such as packet dropping vs. reliable message transmission, low latency vs. jitter, or throughput vs. latency. We must concurrently meet diverse service requirements with the same set of hardware and software resources.

To meet the requirements of SAN/LAN environments we have proposed the Multimedia Router (MMR), whose architecture is illustrated in Figure 1. The router is organized around a synchronously controlled crossbar switch. Messages are partitioned into flow control digits or flits which are quite large: 128-1024 bits. The switch synchronously transmits one flit from each input port containing ready flits. During the transmission the next set of flits is scheduled. Synchronous operation simplifies switch control and increases throughput. The large flit size is a trade-off between hiding flow control/scheduling delays, and jitter and delay. In order to guarantee QoS, connections are established before starting data transmission, thus reserving the required resources.

Connections that are established through the router may deliver constant bit rate (CBR) or variable bit rate (VBR) traffic. Each physical link will support on the order of 256 connections, each one having a dedicated virtual channel. Virtual channels are implemented in interleaved memory banks that support pipelined access to large flits eliminating the traditional virtual channel multiplexors that have been the source of high overhead in previous generation routers.

The multiplexed crossbar allows a compact implementation, since the silicon area occupied by it remains small. It also facilitates scalability, since the complexity of the crossbar increases quadratically with the number of physical links, instead of with the number of virtual channels. Also, scalability is feasible because the buffer requirements increase linearly with the number of virtual channels. Up to an 8×8 MMR implementation is feasible by using current VLSI technology. The interested reader is referred to [7] for a more detailed description of the MMR architecture and design trade-offs.

Link bandwidth is measured in flit cycles/sec. A flit cycle or slot is the time required to transmit a flit through the internal crossbar or through a link. Slots are grouped into frames. The number of slots in a frame is an integer multiple of the number of virtual channels per link.

A network-wide admission control protocol establishes connections through a router if the bandwidth required for the connection (expressed as a number of slots/frame) is available on the requested output link. Such a request corresponds to a CBR connection. VBR connections must specify both a nominal and peak burst bandwidth, and the admission control protocol differs slightly. A scheduling decision is made every flit cycle: an input virtual channel must be selected from each input port and the crossbar synchronously set to connect input ports to output ports. Some connections may be infeasible if multiple input ports request a connection to the same output port. Once a set of conflict-free connections have been established through the switch, flits are synchronously transmitted to the output ports. A point of terminology: we use the term *link scheduling* to refer to the selection of a virtual channel on a physical input link. *Switch scheduling* refers to the problem of matching input ports with requested output ports in a conflict-free manner. Note that we refer to input virtual channels rather than virtual connections. This is because while some virtual channels may be allocated to virtual connections, others may be serving best-effort or control messages. However, from the switch scheduling perspective, they are all treated equivalently.

The main distinguishing features of the MMR can be found in [7]. This paper describes the switch scheduling algorithm proposed for the MMR. The switch scheduling challenge is to compute switch settings to establish connections between input and output ports at speeds comparable to the time to transfer a flit through the switch. Targeting 1.24 Gbps links and 1024-bit flit sizes, the crossbar must be capable of computing switch settings at a rate of one every 825.6 ns. In this paper we propose a partitioning of switch scheduling algorithms into a basic set of decisions and define policies for each of those decisions. The partitioning is motivated by the need for concurrency and pipelining in the scheduler implementation. The design priority is to develop switch scheduling algorithms that are very fast and amenable for single-chip implementation. The hypothesis is that simple, fast algorithms are preferable to optimal algorithms or algorithms that can find maximal input/output matchings but which cannot be implemented in a single flit cycle nor efficiently pipelined, since any improvements in switch throughput would be likely overshadowed by the lengthening of the flit cycle time.

At the heart of the proposed scheduling algorithm is a dynamic priority update scheme which is referred to as *priority biasing* [5, 9, 11]. The novel feature of the proposed approach is that the biasing functions modify the flit priority as a function of the measured QoS on a connection. Thus, the rate at which the priority of a flit increases at any point in time is determined by the relationship between the requested QoS and measured QoS. The performance of two new biasing functions based on this approach are presented at the end of this paper.

3 Priority Biasing

The key idea within the scheduler is the algorithm used to update the priorities of the flits at the head of the input queues. The flit priority is updated as a function of the QoS that a flit has experienced up to that point in time *relative* to the QoS requested by the corresponding connection. Such an update operation has been referred to as *priority biasing* [5, 9, 11] since the priority value is biased by the relative degradation of its service. The biasing operation couples the effect of the scheduler (e.g., queuing delay) with the QoS demand (e.g., jitter bound). This distinguishes this approach from priority update mechanisms such as age counters that do not distinguish connections based on their QoS requirements.

For example, consider only constant bit rate (CBR) connections where QoS is directly related to the bandwidth allocated to a connection. The priority of a flit can be computed as the ratio of the queuing delay to the connection's inter-arrival time. Increasing queuing delay increases its priority in successive scheduling cycles. However, the rate at which the priority increases depends on the bandwidth of the connection. Such a priority biasing mechanism couples the ongoing effect of the switch scheduler (queuing delay) with a measure of the demands made by the application (connection bandwidth). As the negative impact of the switch scheduler grows so does the priority, effectively "biasing it" with time. Thus, different connections are biased at different rates, i.e., higher speed connections are biased faster. Generalizing this approach we hope to be able to effectively accommodate QoS guarantees across heterogeneous streams, i.e., distinct QoS metrics across multiple streams through a switch.

The first biasing function we propose is referred to as *Inter-Arrival Based Priority* (IABP).

The priority of the head flit in an input queue is computed as the ratio of the queuing delay and the connection's inter-arrival time. The rate at which the priority of a flit increases depends on the bandwidth of the connection. IABP couples the ongoing effect of the switch scheduler (queuing delay) with a measure of the demands made by the application (connection bandwidth). For example, consider two flits, one each from a low speed (64 Kbps) and high speed (55 Mbps) connection, respectively. If they both experience the same delay through the switch, say, 1 microsec, this delay is much greater percentage of the inter-arrival time on the high-speed connection and as a result has a much higher impact on the connection's QoS statistics. Biasing based on IABP increases the priority of a flit on the higher speed connection at a much faster rate.

A second biasing function is *Jitter-Based Priority* (JBP) where the priority of a flit is computed as the ratio of the flit jitter[1] plus the accumulated jitter, and the connection's inter-arrival time. We compute the accumulated jitter as the sum of all the successive flit delay differences. The idea is to have successive flits on a connection experience the same service time and thereby minimize jitter. This approach reduces the priority of a flit if it is "early". A flit is early if it has experienced a smaller queuing delay than the average queueing delay from its parent connection. An early flit will have a low priority, thus delaying its scheduling so that it can "age" longer. If the flit has experienced an equal or longer delay than its parent connection's average, it will increase its priority at a rate that depends on the bandwidth requirements of the connection. In both cases, the priority is biased. When a flit is transferred over the crossbar, its delay through the switch is recorded and the accumulated jitter for its parent connection is updated.

Finally, in addition to how the priority of a flit is computed, we have the question of when the priority is computed. For example, the priorities of the head flits in all input queues can be recomputed every flit cycle. Alternatively, they may be updated periodically or in an event-driven manner, for example, when a flit arrives at a queue or when a flit leaves the queue. These alternative approaches may significantly reduce the amount of hardware required to update priorities for all the virtual channels in real time. We do not address this problem in depth in this paper but assume that flit priorities are recomputed every flit cycle. This will enable us to assess the performance limits of priority biasing.

With priorities updated as described in this section, the following section discusses the

[1]Flit jitter is equal to the queuing delay for the current flit minus the queuing delay for the previous flit

structure of the switch scheduler. This structure is motivated by the need for concurrent and pipelined operation to keep the switch scheduler operating at link speeds.

4 Scheduler Implementation

The MMR has a link scheduler associated with each input link and a single switch scheduler (see Figure 1) Each link scheduler provides a set of requests for output ports to the switch scheduler. The switch scheduling problem is one of matching input ports to output ports to maximize switch utilization and the QoS across connections. A common approach towards generating a switch setting is the use of iterative matching algorithms [1]. However, such solutions with iterative flow of control between input and output ports are not amenable to high-speed hardware implementations. We are interested in implementations that are amenable to pipelining and parallelism and with a feed forward flow of control information. To do so, the switch scheduling functionality is partitioned into a sequence of basic decisions: *candidate selection* (performed by link schedulers), *ordering*, and *arbitration*. The scheduler implementation can be pipelined at the level of these functions.

4.1 Candidate Selection

A *candidate* on an input port is defined as the next flit in a virtual channel that is ready to be scheduled to an output port. Each link scheduler will produce a short list of candidates from the virtual channels on that link and record them in the input port *candidate vector*. Each entry in the vector records the scheduling data for the corresponding flit. Typically this is at least the output port and the priority value. Figure 2 shows a typical input candidate vector. The priority value shown here is the bandwidth of the connection expressed in Mbps. This does not correspond to the way priorities are actually encoded. The destination is the crossbar output port.

Candidate selection is based on the priority value and can be done in parallel by link schedulers. Based on this criterion anywhere from 1 to N candidates are selected from each input port for an $N \times N$ switch (assuming that there exist active connections in all the ports). The candidates are ordered according to priority value and the position of a candidate on this order is the *level* of the candidate. Thus, the highest priority flit at an input port is a level 1 candidate at that port, the next highest priority flit a level 2 candidate, and so on. Candidate vectors are then forwarded to the switch scheduler.

Following candidate selection, the results are recorded in a *selection matrix* which is a matrix that is used to record the output port requests from all input ports. An example of a selection matrix is shown in Figure 3. The matrix has column dimension equal to the number of input ports, and row dimension equal to the product of the number of output ports and the number of candidates at each input port (each input port is assumed to provide the same number of candidates, if available). The columns are labeled with the corresponding input port, and the rows are labeled with output ports. The first N rows record the highest priority candidates from each input. For example, entry (i, j) is set if the highest priority flit from input j requests output i. The next N rows in the matrix record the level two candidates from each input port, and so on. For example, in Figure 3 we see that there are two candidate flits in row 0 - in column 1 and in column 3. This corresponds to the highest priority connections from input ports 1 and 3 having flits destined for output port 0. In row 7, we see that input ports 0 and 2 have flits destined for output port 3 denoting that the second level candidates from input ports 0 and 2 are destined for output port 3. Input ports 1 and 3 provided a single candidate. This indicates that there were no more flits ready for scheduling at those ports.

From the selection matrix the switch scheduler creates a *conflict vector*. This is the number of non-null entries in each row. The conflict vector identifies the number of conflicts for an output port. Figure 4 shows the conflict vector corresponding to the matrix in Figure 3. Note that the conflicts are naturally grouped by priority level.

4.2 Port Ordering

Two critical decisions that must now be made by the switch scheduler are the order in which output ports must be examined to resolve port conflicts and the arbitration policy. The former is important because when an input port is matched to an output port, it drops requests for all other candidates from this input port. This may change the conflicts at other ports. As a result, the order in which ports are considered can affect the number of input port requests that can be satisfied, and thus, switch utilization. This decision is captured in the *ordering function*. The ordering function proposed and evaluated here selects output ports first by level and then in increasing order of conflict within a level. Ties are broken by randomly selecting one of the ports. The rationale is that ports with the most conflicts should be matched last since those ports have the most opportunities to be matched to an input port. If this ordering function was applied to Figure 4, we would first consider conflicts among level 1 candidates (highest priority candidates) from all inputs. Output ports 1 and 2 would be selected first and assigned to input ports 2 and 0, respectively. Then output port 0 would be considered and it would be necessary to arbitrate between competing requests from input ports 1 and 3. Thus, we cannot fully utilize the switch bandwidth using only level one candidates. The algorithm now moves to level 2 candidates to try to fill in any remaining “holes” in the next flit cycle. This process is repeated for all candidates. A larger number of candidates clearly increases the probability of fully utilizing the switch but with increased implementation complexity.

4.3 Arbitration

Arbitration at an output port is captured in the *arbitration function.* Arbitration is simply a matter of picking the highest priority flit where priorities are computed and updated as described in Section 3.

5 Example

Figure 5 shows an example selection matrix derived from a 4×4 switch with two levels of candidates per input port. Each matrix entry in the figure contains the bandwidth requirements for the corresponding connection. Fixed priority will be used in this example, where the priority is determined by the inter-arrival times of the connections. The ordering function selects output ports in increasing order of conflict. Since both ports 0 and 2 have the same degree of conflict, one is randomly selected. This selection is implemented using a free running counter to implement a random number. Let us assume that port 0 is considered first. The first invocation of the ordering function returns 0 as the port for which arbitration is required. Since there is a conflict at port 0, the arbitration function will be invoked and it will return input port 1 ($55 > 24$). Therefore, input port 1 will be assigned to output 0 for the next flit cycle. Now that output port 0 is assigned, the selection matrix must be updated. The column corresponding to input port 1 must be re-initialized to 0. Any candidate from another input port with output port 0 as a destination must also be removed from further consideration by updating the matrix, including candidates with a different level. Once the matrix is updated, the conflict vector must be recomputed. The updated data structures are shown on the left side of Figure 6 (assigned output ports are marked with an X in the figure). The next invocation of the ordering function returns output port 2. The arbitration function returns input port 0 ($55 > 12$). Input port 0 is assigned to output port 2 and the data structures are updated. The state of the matrix and conflict vector after this iteration is shown on the right side of Figure 6. If a single level of candidates was considered, switch scheduling would finish at this point and two output ports would remain unassigned. Now, output port 1 is arbitrated for next followed by output port 3. The results of these arbitrations are shown in Figure 7.

6 Performance Evaluation

In this section some results showing the performance obtained by the IABP and JBP scheduling algorithms are presented. Simulations of a 4×4 router have been carried out. There are 256 virtual channels per physical link. Flit size is 1024 bits. Physical link bandwidth is 1.24 Gbps, and links are 16-bit wide. Thus, the router takes 825.6 nanoseconds to transmit a flit. This is also the time available to compute the link and switch scheduling. Simulations have been run for all the possible number of candidate levels, ranging from 1 to 4, to check their effect on performance.

Workload is composed of combinations of three different types of CBR connections, which represent different amounts of bandwidth requirements: 0.064 Mbps (IAT[2] = 16250 microsecs), 1.54 Mbps (IAT = 675.33 microsecs) and 55 Mbps (IAT = 18.92 microsecs). In this way, we can check the behavior of the proposed algorithms on low, medium, and high bandwidth consuming connections. Workload level is computed as the percentage of the physical link bandwidth filled with connections.

First of all, the average crossbar utilization was measured. The results are presented in Figure 8. For the IABP algorithm, it can be seen that if the number of candidate levels is increased, crossbar utilization improves. However, as the number of candidate levels increases, diminishing returns are obtained. For example, the maximum utilization reached by the crossbar with two candidate levels at saturation is around 77% of link bandwidth. However, the router is only able to reach almost 90% utilization if four candidate levels are considered. If only one candidate level is considered, the router can only reach an utilization of 70%. On the other hand, the JBP algorithm enters saturation at around 70 % of workload, regardless the number of candidate levels (except for a single candidate). For

[2]Inter-Arrival Time

workloads greater than 72 %, the average crossbar utilization drops heavily, indicating that the JBP algorithm cannot effectively solve output contention.

But in order to provide QoS guarantees to the applications, we need to know not only the amount of data the router is able to forward, but also the QoS provided to each connection. For this reason, flit delay, distribution of flit delay, and jitter were measured. In Figure 9, average flit delays for the three different types of connections and the two scheduling algorithms are presented for those levels of workload which are close to saturation. Note that the workload consists of a mix of connections with different bandwidth requirements but we present results for each set of connections separately. For the sake of clarity, the delay results obtained for one level of candidates and IABP are not presented, since they are rather high even for workload levels of around 50%. It becomes clear that the IABP algorithm needs more than one level of candidates to work properly.

First, let us consider the IABP algorithm. The shape of the three plots is quite similar. One can notice that delay improves when the number of candidate levels is increased, as well as throughput. Below saturation, delay improvement is more noticeable for the connections with low bandwidth requirements. This behavior is due to the fact that, as the number of candidate levels is increased, more matchings between input and output ports are obtained, and because of the way priorities work, the additional matchings from every additional level of candidates will likely schedule flits from the lowest bandwidth connections. The improvement in terms of throughput is specially noticeable for the 55 Mbps connections and 4 levels of candidates. By comparing the plots for connections with different bandwidth requirements, it can be clearly seen that the improvement in throughput for high bandwidth connections is achieved at the expense of not servicing the other connections. This explains the higher crossbar utilization achieved by IABP with respect to JBP (see Figure 8). Clearly, the router should not work beyond saturation because the lowest bandwidth consuming connections are not serviced at all. The different time scales in the plots are also remarkable. Delays for the 55 Mbps connections (IAT = 18.92 microsecs) range between 2 and 40 microseconds, while the 0.064 Mbps connections (IAT = 16250 microsecs) experience delays on the order of a couple of hundreds of microseconds. This reflects the different QoS received by the different types of connections, according to their requirements. Finally, these plots show that the IABP algorithm is able to deliver QoS to the less bandwidth consuming connections only for workloads not greater than 70%-80%, depending on the number of candidate levels used.

Regarding the JBP algorithm, the plots show that the increase in the number of candidates has little effect on the delays experienced by the flits, for all the types of connections. This is very interesting, since the implementation of the algorithm can be much simpler by considering less levels of candidates, without losing QoS. Notice also that, in this case, all the connections receive similar QoS (the time scales for all the plots are the same). This might not be desirable, because the less bandwidth consuming connections have less strict delay requirements, and the algorithm is delivering them more QoS than required. Despite this fact, the higher bandwidth consuming connections still receive the requested QoS and forward all their flits in time, experiencing a delay similar to the one for the IABP algorithm. However, JBP saturates at a slightly lower workload than IABP when using more than two candidate levels.

As a conclusion, both the IABP and the JBP algorithms are able to satisfy the QoS requirements of all the types of connections within similar workload levels. However, the IABP algorithm favors the higher bandwidth consuming connections, while the JBP algorithm favors the less bandwidth consuming connections. Note that although the IABP algorithm is able to reach an utilization of 80% for two levels of candidates, and even nearly 90% if more levels of candidates are considered, the delays obtained by the lower bandwidth connections at those workloads are extremely high.

In order to get more information about the distribution of flit delay, the percentage of flits whose delays are lower than a set of thresholds have been computed, for some workload levels below and close to saturation. This gives an idea of the percentage of flits that will meet a given deadline. Data were obtained for all the possible levels of candidates, from one to four. The thresholds were different for each type of connection, and were related to their inter-arrival time (IAT). They were set to $2 \times IAT$, IAT, $IAT/2$, $IAT/4$, $IAT/8$, and $IAT/16$. Recall the different IAT for every type of connection: 16250 microsecs for the 0.064 Mbps connections, 675.33 microsecs for the 1.54 Mbps connections, and 18.92 microsecs for the 55 Mbps connections. Therefore, thresholds become tighter as bandwidth requirements increase. Results are presented in Figure 10 for the IABP algorithm, and in Figure 11 for the JBP algorithm.

The plots for the IABP algorithm confirm the influence of the number of candidate levels used on the percentage of flits that meet their deadline. This is more noticeable as the connection requirements increase, because their deadlines are also tighter. We will need at least two levels of candidates in order to be able to guarantee the required QoS, and even three levels of candidates are needed when the workload is 75%. The plots for the JBP algorithm also show that two levels of candidates are required to guarantee QoS. Furthermore, an increase in the number of levels of candidates does not have a significant impact on flit delay distribution.

In order to summarize, the maximum workload for which the number of flits whose delay is lower than the IAT is 99 % of the emitted flits is shown in Table 1 for both scheduling algorithms. It can be clearly seen that the throughput reached by the IABP algorithm while still providing QoS to all the connections is higher than the one achieved by the JBP algorithm.

Most important of all, both scheduling algorithms are able to guarantee QoS to all the flits when the router is working below saturation, flit deadlines are equal to or higher than IAT, and two or more levels of candidates are used.

Finally, average flit jitter has been measured. Results are plotted in Figure 12. As can be seen, the magnitude of jitter is the same as the delay corresponding to the same type of connection and scheduling algorithm. Thus, the JBP algorithm provides far less variation in latency than IABP for the low bandwidth consuming connections, and values in the same order of magnitude for the most bandwidth consuming ones.

7 Conclusions and Future Work

The Multimedia Router project arises as a response to the increasing need for QoS guarantees within local environments. The purpose of the MMR project is not only satisfying this need for QoS, but also to achieve it with a single-chip switching element that provides latencies similar to those obtained by the cut-through routers used in multicomputers and high-performance LAN/SAN.

The link and switch scheduling algorithms are key elements within the design of the router. In this paper, a new algorithm suitable for high-speed hardware implementation has been presented. This new algorithm introduces the concept of “biased priority”, which relates the QoS received by a flit with the QoS required by the connection in order to prioritize flit transmission.

Two different biasing functions are proposed and evaluated: Inter-Arrival Based Priority and Jitter Based Priority. Both of them are able to achieve high link utilizations, while still guaranteeing QoS to the connections. IABP makes more differences in the QoS delivered to the different types of connections, while JBP treats all the types of connections equally. Also IABP is more heavily influenced by the number of candidate levels than JBP.

As future work, the adaptation of the proposed scheduling algorithm in order to be used with VBR traffic has to be carried out, as well as its evaluation with mixed (CBR+VBR) traffic.

Finally, best-effort traffic has to be included in the priority scheme in such a way that it does not interfere with the QoS provided to the multimedia flows, while avoiding its starvation.

Acknowledgement

We would like to thank Dr. Luis Orozco-Barbosa, from the University of Ottawa, his useful comments which have helped to considerably improve the quality of this paper.

References

[1] T. E. Anderson et al, “High speed switch scheduling for local area networks,” Technical Report SRC research report 99, DEC. Also as *ACM Transactions on Computer Systems*, vol. 11, no. 4, November, 1993.

[2] S. Balakrishnan and F. Özgüner, “A priority-based flow control mechanism to support real-time traffic in pipelined direct networks,” *Proceedings of the 1996 International Conference on Parallel Processing*, vol. I, pp. 120–127, August 1996.

[3] N. J. Boden, et al., “Myrinet - A gigabit per second local area network,” *IEEE Micro*, pp. 29–36, February 1995.

[4] J. Carbonaro and F. Verhoorn, “Cavallino: The teraflops router and NIC,” *Proceedings of Hot Interconnects Symposium IV*, August 1996.

[5] A. Chien, J.H. Kim, “Approaches to Quality of Service in High Performance Networks,” *Proceedings of the Workshop on Parallel Computer Routing and Communication*, Lecture Notes in Computer Science, Springer-Verlag, pp.1-19, June 1997.

[6] K. Dienfendorff, P. Dubey, “How multimedia workloads will change processor design,” *IEEE Computer*, vol. 30, no. 9, pp.43-45, September 1997.

[7] J. Duato, S. Yalamanchili, M.B. Caminero, D. Love, and F.J. Quiles, “MMR: A high-performance multimedia router. Architecture and design trade-offs,” *Proceedings of the 5th Symposium on High Performance Computer Architecture (HPCA-5)*, pp. 300-309, January 1999.

[8] M. Galles, “Scalable pipelined interconnect for distributed endpoint routing: The SPIDER chip,” *Proceedings of Hot Interconnects Symposium*, August 1996.

[9] D. Garcia, D. Watson, “ServerNet II,” *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pp. 119-136, June 1996.

[10] M. G. H. Katevenis, et al., “ATLAS I: A single-chip ATM switch for NOWs,” *Proceedings of the Workshop on Communications and Architectural Support for Network-based Parallel Computing*, February 1997.

[11] J.H. Kim, "Bandwidth and latency guarantees in low-cost, high-performance networks," Ph. D. Thesis, Department of Computer Sciences, University of Illinois at Urbana-Champaign, 1997.

[12] C.E. Kozyrakis, D. Patterson, "A new direction for computer architecture research," *IEEE Computer*, vol.31, no. 11, pp. 24-32, November 1998.

[13] M. Prycker, *Asynchronous transfer mode: solution for broadband ISDN*, Ellis Horwood Limited, Chichester, West Susex, PO191EB, England, 1991.

[14] J. Rexford, J. Hall and K. G. Shin, "A Router Architecture for Real-Time Point-to-Point Networks," *Proceedings of the International Symposium on Computer Architecture*, May 1996.

[15] D. Stiliadis, A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," *Proceedings IEEE INFOCOM*, 1995.

Appendix: Figures and Tables

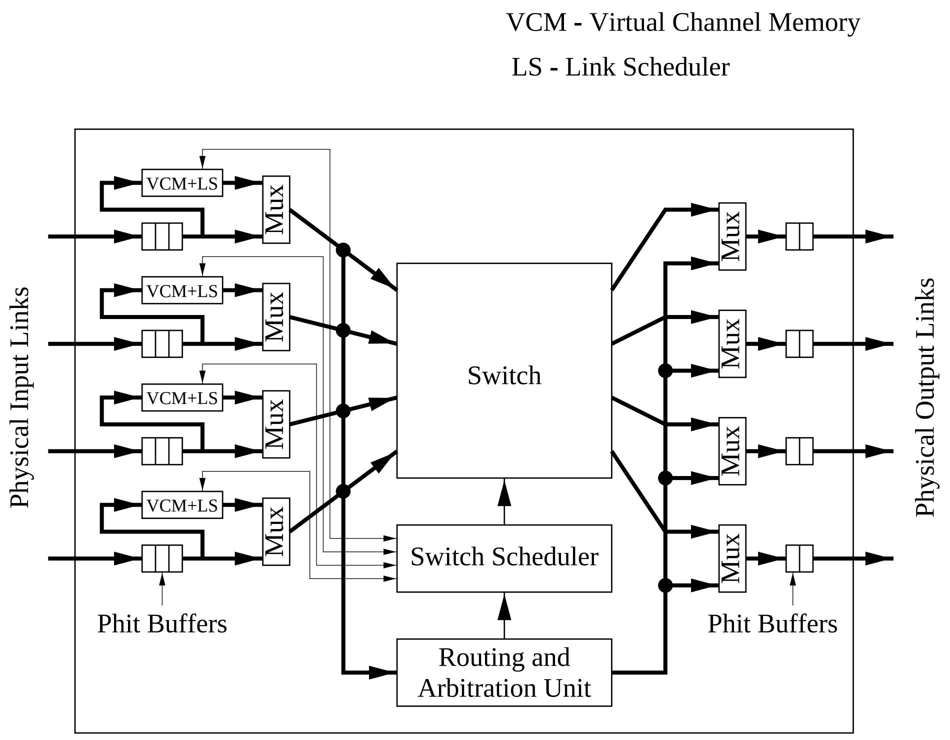


Figure 1. Multimedia Router (MMR) architecture

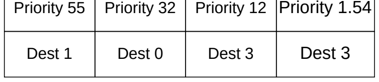


Figure 2. An example candidate vector

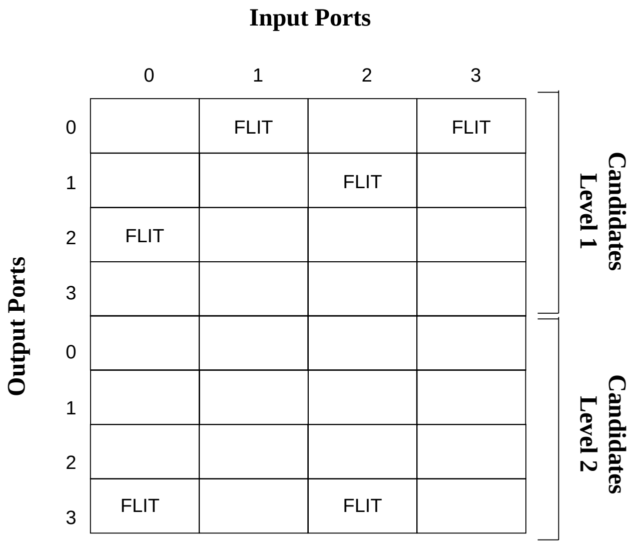


Figure 3. An example selection matrix

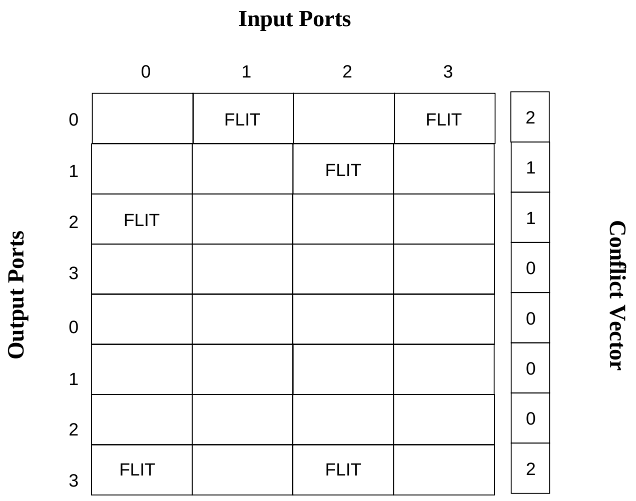


Figure 4. An example conflict vector

Output Ports \ Input Ports	0	1	2	3	Conflict Vector
0		55 Mbps		24 Mbps	2
1					0
2	55 Mbps		12 Mbps		2
3					0
0	64 Mbps				1
1			5 Mbps		1
2					0
3		20 Mbps		10 Mbps	2

Figure 5. Selection matrix for the example

Output Ports \ Input Ports	0	1	2	3	Conflict Vector
0		X			0
1					0
2	55 Mbps		12 Mbps		2
3					0
0					0
1			5 Mbps		1
2					0
3				10 Mbps	1

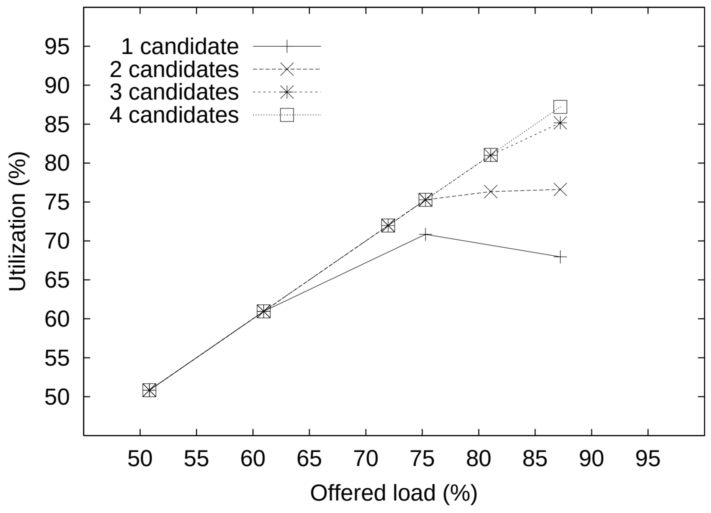
Output Ports \ Input Ports	0	1	2	3	Conflict Vector
0		X			0
1					0
2	X				0
3					0
0					0
1			5 Mbps		1
2					0
3				10 Mbps	1

Figure 6. Selection matrix after arbitration for output ports 0 and 2

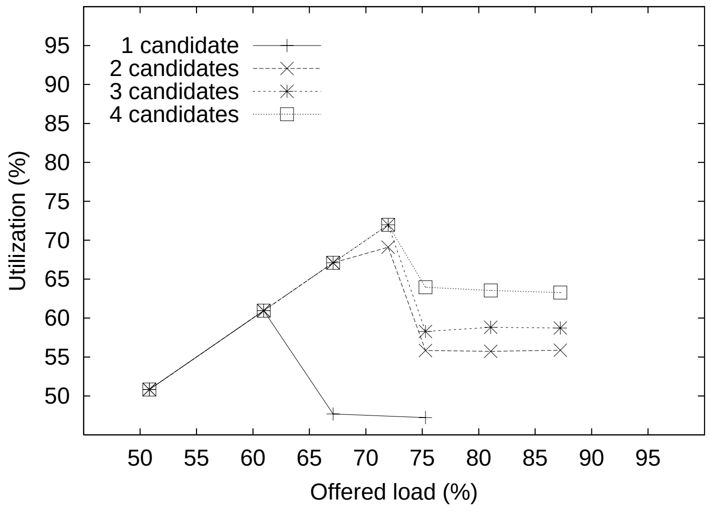
Output Ports \ Input Ports	0	1	2	3	Conflict Vector
0		X			0
1					0
2	X				0
3					0
0					0
1			X		0
2					0
3				10 Mbps	1

Output Ports \ Input Ports	0	1	2	3	Conflict Vector
0		X			0
1					0
2	X				0
3					0
0					0
1			X		0
2					0
3				X	0

Figure 7. Selection matrix after arbitration for output ports 1 and 3

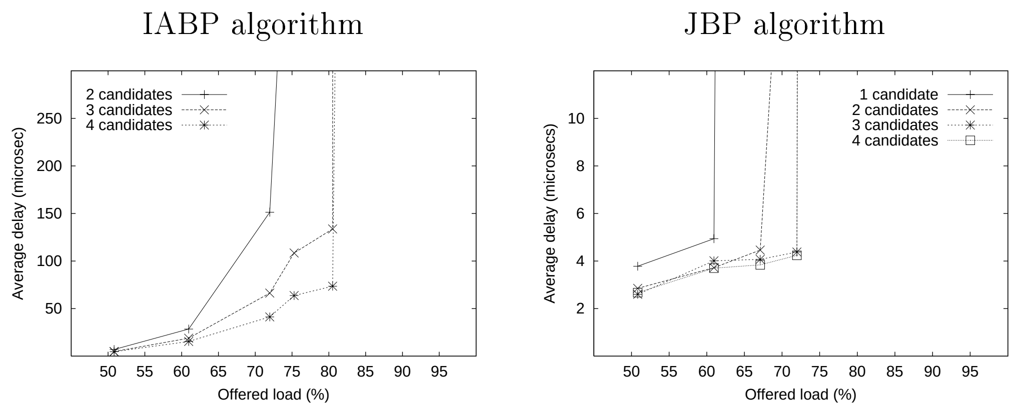


(a) IABP algorithm

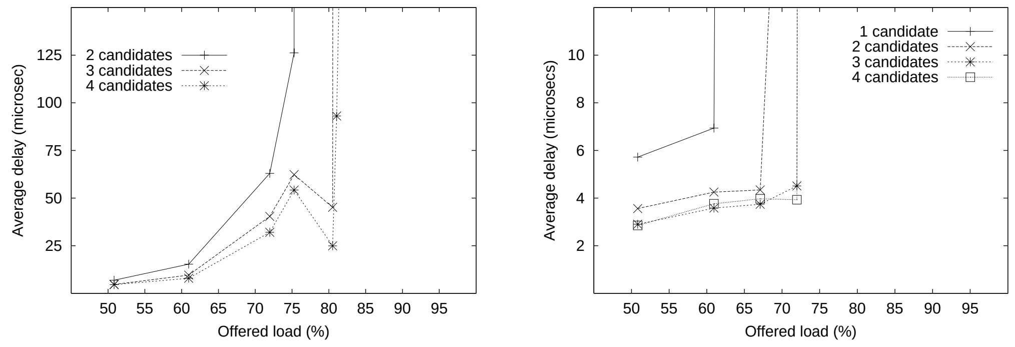


(b) JBP algorithm

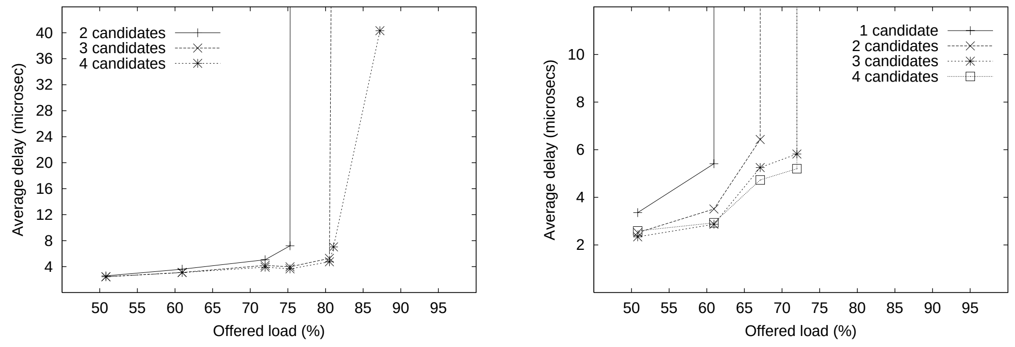
Figure 8. Average crossbar utilization



(a) 0.064 Mbps connections (IAT = 16250 microsecs)



(b) 1.54 Mbps connections (IAT = 675.33 microsecs)

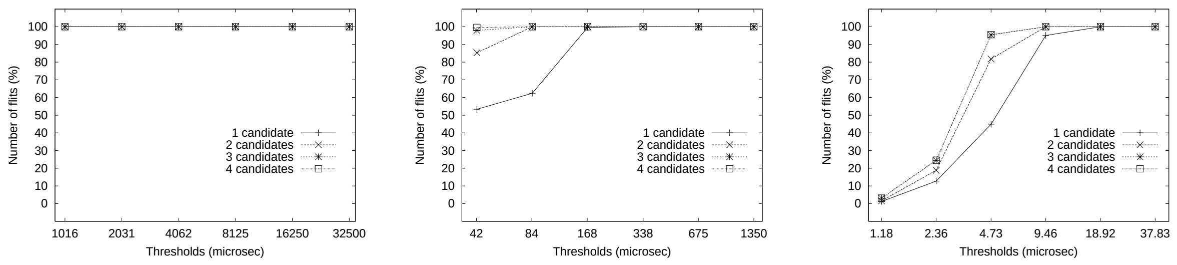


(c) 55 Mbps connections (IAT = 18.92 microsecs)

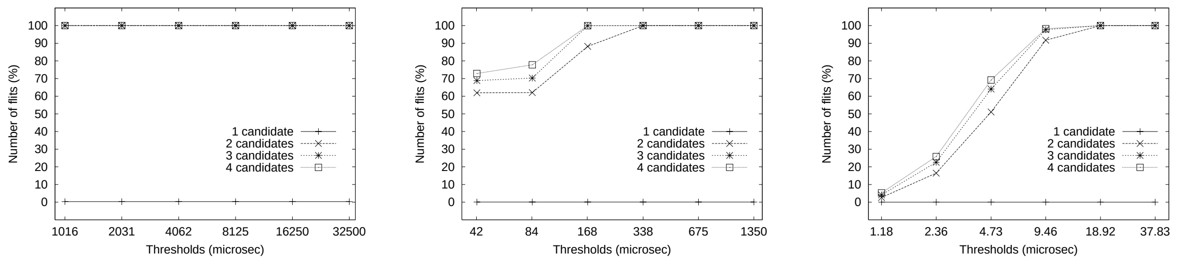
Figure 9. Average flit delay

0.064 Mbps connections   1.54 Mbps connections   55 Mbps connections

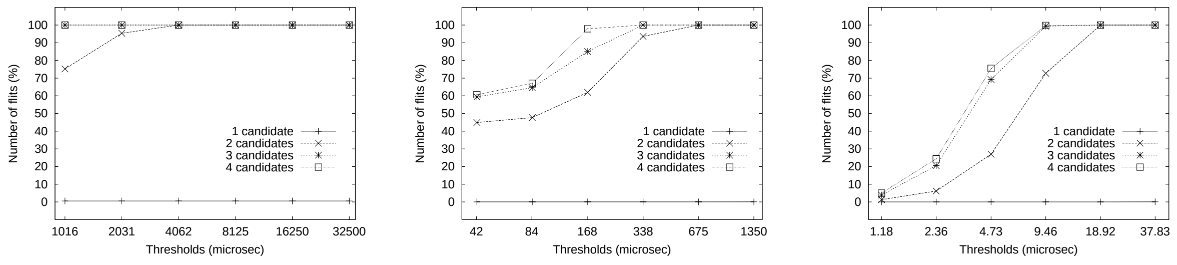
IAT = 16250 microsecs   IAT = 675.33 microsecs   IAT = 18.92 microsecs



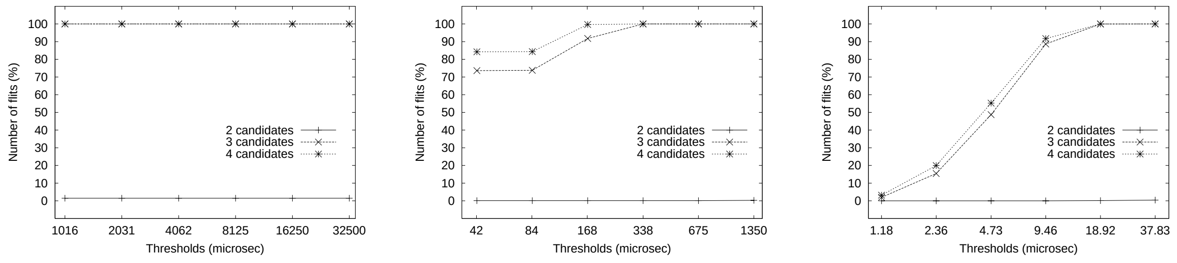
(a) Workload = 60.953 %



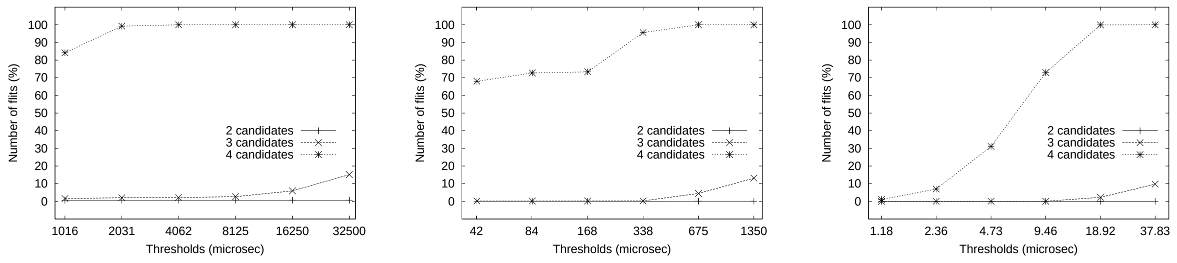
(b) Workload = 71.974 %



(c) Workload = 75.281 %



(d) Workload = 80.515 %

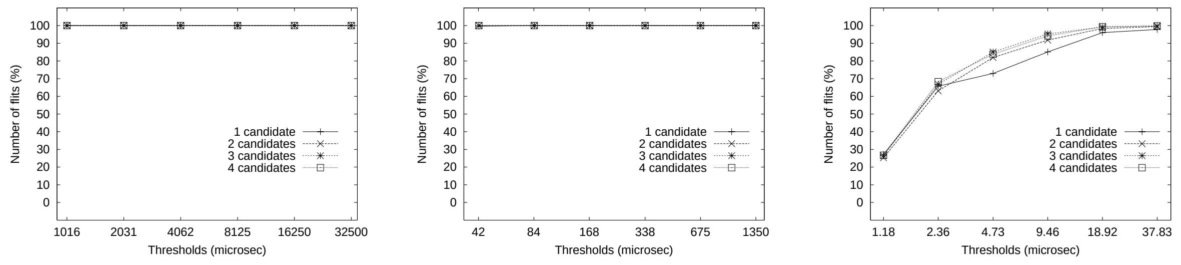


(e) Workload = 81.057 %

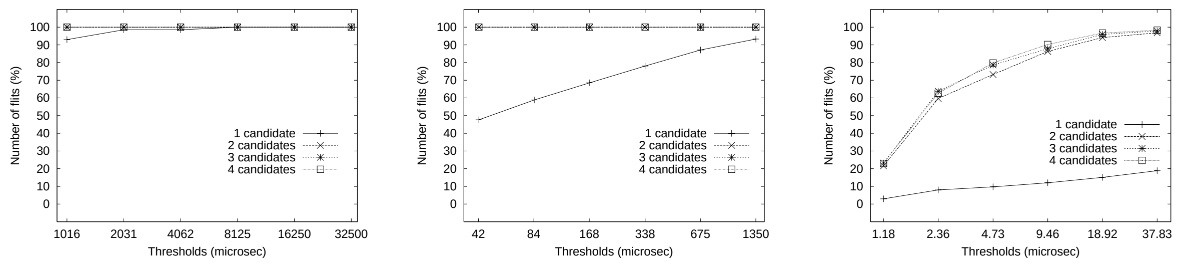
Figure 10. Distribution of flit delay for different workload levels and different types of connections, for the IABP algorithm

0.064 Mbps connections 1.54 Mbps connections 55 Mbps connections

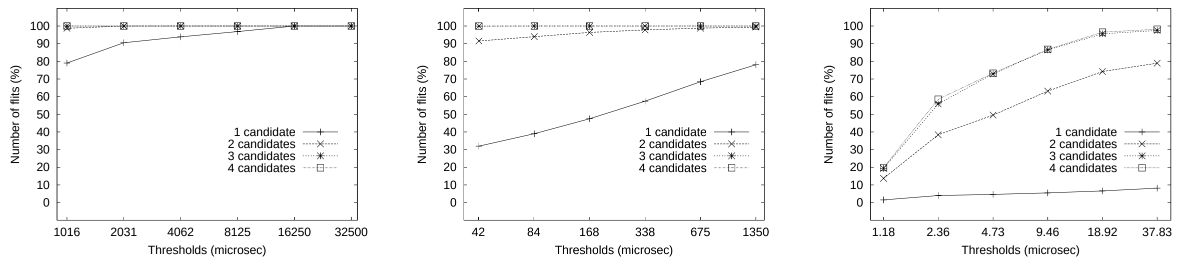
IAT = 16250 microsecs IAT = 675.33 microsecs IAT = 18.92 microsecs



(a) Workload = 60.953 %

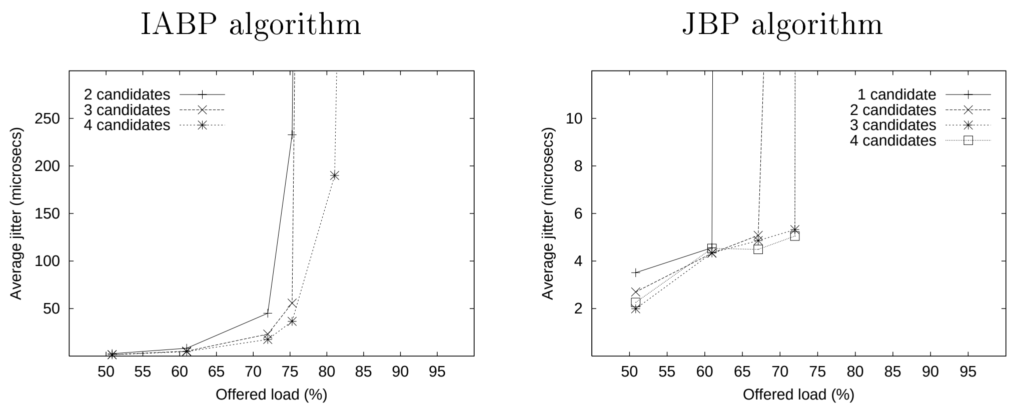


(b) Workload = 67.100 %

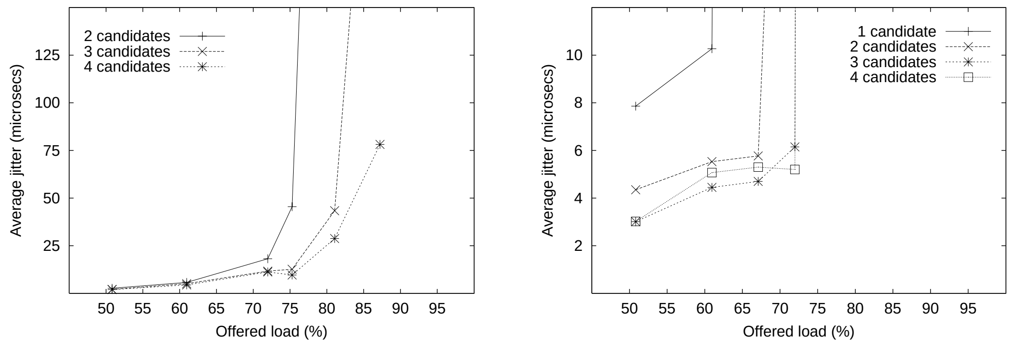


(c) Workload = 71.974 %

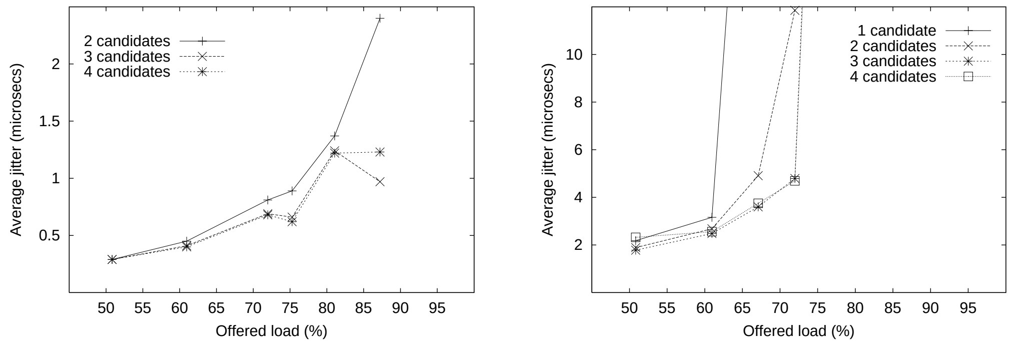
Figure 11. Distribution of flit delay for different workload levels and different types of connections, for the JBP algorithm



(a) 0.064 Mbps connections (IAT = 16250 microsecs)



(b) 1.54 Mbps connections (IAT = 675.33 microsecs)



(c) 55 Mbps connections (IAT = 18.92 microsecs)

Figure 12. Average flit jitter

	1 cand	2 cand	3 cand	4 cand
IABP	50.828 %	75.281 %	80.515 %	81.057 %
JBP	50.828 %	60.953 %	60.953 %	60.953 %

Table 1. Maximum wokloads with QoS guarantees: the 99% of the transmitted flits experience delays lower than the IAT of their connection