# Design of a Real-Time Switch with Bounded Delays

Kyungtae Kang and Lui Sha

**Abstract**

Most network switches are designed for best-effort transmission of Internet traffic. Relevant studies have focused on maximizing throughput and minimizing delay in an average sense, but this is not suitable for hard real-time applications, in which guaranteeing delay bound is critical. We propose a real-time switch design, based on a crossbar switching fabric, which combines clearance-time-optimal switching with clock-based scheduling, and we show that any feasible traffic is guaranteed to be switched in two clock periods. The concept of *one-shot traffic* allows delay to be bounded without a requirement for traffic periodicity, at the cost of a fixed delay of one clock period. The proposed switch uses real-time virtual machine tasks to serve traffic, which simplifies analysis and provides isolation from other system operations. Simulations show that our real-time switch achieves a larger schedulability region and a bounded lower end-to-end delay with a shorter clearance time than $i$SLIP, which is one of the most widely used crossbar switch schedulers.

**Index Terms**

Real-time switch, bounded delay, schedulability, clock-driven scheduling.

## I. Introduction

By seamlessly integrating sensing, networking and computation with the control of physical devices and processes, cyber-physical systems (CPS) are expected to transform the way in which we interact with and manipulate the physical world. CPS will have a far-reaching impact on science and engineering, and are critical to a wide range of applications such as transportation systems and healthcare. One basic enabler of CPS is pervasive wireless networking, which provides the coordinating foundation for sensing, computing, and control [1], [2]. CPS wireless networks support mission-critical, real-time, closed-loop sensing and control, and represent a significant advance on existing wireless or wireless sensor networks. A critical aspect of CPS is controllable, predictable messaging, and so real-time networking plays a key role in control, communication, and computation.

Real-time networks rely on efficient real-time switches. Although there have been extensive studies of network switches, they have typically focused on improving average throughput and delay, rather than on the provision of the guaranteed delay bound that is critical to real-time applications. This is the problem that we address.

Our design of real-time switch has a bounded delay under any feasible traffic, and this guarantees a bound on the resulting end-to-end delay in a packet-switched network. The adoption of an optimal clearance-time policy, which offers arbitrarily low computational complexity, and clock-driven scheduling guarantees that *any feasible traffic* is switched in *two clock periods*. This is accomplished by introducing the concept of *one-shot traffic*, in which the traffic arriving during each clock period is buffered and then scheduled to be switched in the next clock period by a clearance-time-optimal scheduler. Thus clearance-time-optimality is achieved at the price of an additional delay of one clock period.

This switch design provides a building-block for obtaining guaranteed end-to-end delay bounds and for enabling hierarchical scheduling policies [3]–[6]. Additionally, our switch has a large schedulablity region, a high throughput, and a low bounded delay, which are shown through extensive simulations.

As we already said, there has been a lot of work on the design of network switches [7]–[11], but most of these studies have focused on reducing the average delay, rather than on real-time applications. Nevertheless, real-time switches have recently received more attention [12], [13]. In particular, Qixin et al. [13] created an efficient real-time switch for deterministic and periodic real-time traffic, by making minimal modifications to $i$SLIP, which is a popular scheduling mechanism for crossbar [14] switches. Our work differs significantly in that we introduce *no assumption about the periodicity of traffic*, but we do provide a *delay guarantee for any feasible traffic*. However, our focus is *not* on the development of switching theory, but on using it to design an efficient real-time packet switch.

The remainder of this paper is organized as follows: In Section II, we introduce the concept of a crossbar switch and the $i$SLIP crossbar switch scheduler. In Section III, we present a real-time switching algorithm, based on clock-driven scheduling, which can guarantee a bounded delay for any feasible traffic. We perform a numerical study on this algorithm in Section IV, and then assess its schedulability and delay performance. We review related work on switch design in Section V. Finally, we present our conclusions and suggest some future research avenues in Section VI.

The authors are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Thomas M. Siebel Center for Computer Science, 201 North Goodwin Avenue, Urbana, IL 61801-2302, e-mail: {ktkang,lrs}@illinois.edu.

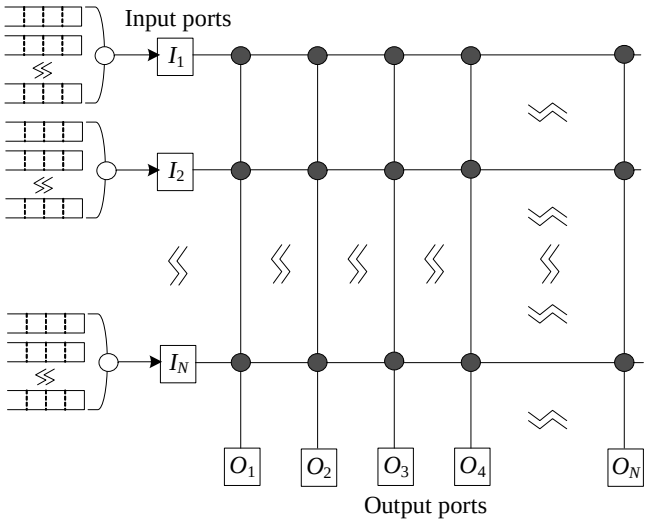



Fig. 1. $N \times N$ crossbar switch fabric.

## II. Crossbar Switches and $i$SLIP

### A. Crossbar switches

The underlying hardware fabric that we adopt is $N \times N$ crossbar, as shown in Fig. 1. At every time-slot, each input port picks up a packet from one of its queues and sends it to its destined output port over a data bus, which is one of the horizontal lines in Fig. 1. The data bus from each input intersects with that of each output, shown as a vertical lines segment in Fig. 1. The switch fabric makes and breaks the connections between inputs and outputs during runtime, following the switching logic. Connections must not be made at two or more crossing-points in the same column, or the corresponding packets will collide at the same output. Similarly, if connections are made at two or more crossing-points in the same row, one or more packets will be delivered as unexpected duplicates at the wrong output port. Consequently, crossbar connections are limited to a set of $N!$ permutation matrices, where a permutation matrix is defined as an $N \times N$ matrix containing 0s and 1s, with a maximum of a single 1 in each row and column. To facilitate the design of the switching logic, we will assume that the size of a packet is fixed, and that transmission of a packet takes one time-slot.

We will now analyze the logic of this type of switch in more detail. Let the input queue $(i, j)$ hold the packets traveling from input $i$ to output $j$. In addition, let $A_{ij}(t)$ denote the number of packets arriving at queue $(i, j)$ at time-slot $t$, and let $Q_{ij}(t)$ denote the number of packets in the queue $(i, j)$ during the same time-slot. Then the switching decision variable $S_{ij}(t)$ at time-slot $t$ can be defined as follows:

$$S_{ij}(t) = \begin{cases} 1, & \text{if crossing-point } (i,j) \text{ is turned on at } t; \\ 0, & \text{otherwise.} \end{cases}$$

Note that the switching matrix $[S_{ij}]$ must correspond to one of the $N!$ permutation matrices because of the crossbar constraint described above. The number of packets in queue $(i, j)$ at time-slot the next $t+1$ is then determined by the following equation:

$$Q_{ij}(t+1) = \max\{Q_{ij}(t) - S_{ij}(t), 0\} + A_{ij}(t). \tag{1}$$

The maximum operation on the right-hand side of this equation is present to cover the case in which $Q_{ij}(t) = 0$ and $S_{ij} = 1$. Informally, we can say that the purpose of a real-time switching algorithm is to select a switching matrix $[S_{ij}]$ at every time-slot, so as to satisfy the timing constraint.

Now we will look into the stability region of a crossbar switch. We can define the rate of arrival of packets at queue $(i, j)$ as follows:

$$\lambda_{ij} := \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t} A_{ij}(\tau).$$

It is well known [14] that the stability region of a switch can be expressed as the set of rate matrices that satisfy the following $2N$ inequalities:

$$\sum_{j=1}^{N} \lambda_{ij} \le 1, i = 1, \ldots, N, \tag{2}$$

and

$$\sum_{i=1}^{N} \lambda_{ij} \leq 1, j = 1, \ldots, N. \tag{3}$$

A consequence of this formulation is that every queue in the switch algorithm is certain to remain of finite length, so long as the arrival rates remain in the stable region. If these inequalities do not hold for any $i$ or $j$, the corresponding queue is equally certain to grow without limit. It is not a simple task to develop a switching algorithm that is efficient, able to serve any periodic or non-periodic traffic, and sure to stay in the stability region. Switching algorithms that can handle any traffic within the stability region are called *throughput-optimal*.

We will now introduce a graph-theoretic approach to the design of switching algorithms, by transforming the problem of packet switching with a crossbar fabric into a search for matches in bipartite graphs as follows: At each time slot, let $G$ denote a bipartite graph connecting the input ports to the output ports. Each port in the switch is considered as a node in $G$. (Thus *port* and *node* will be used interchangeably). There will be an edge connecting input port $i$ and output port $j$ in $G$ if there are packets waiting at input port $i$ that are destined for output port $j$. A switching algorithm finds a matching $M$ in $G$, possibly based on the current state of the queues, and that matching corresponds to the selection of a switching matrix $[S_{ij}]$ in (1), and determines which packets from the input ports will be switched at the current time-slot.

We will use $Q_i$ to denote the total number of packets waiting at the input port $i$; and $Q_j$ denotes the total number of packets (at all the inputs) destined for output port $j$. Thus $Q_i = \sum_{j=1}^{N} Q_{ij}$ and $Q_j = \sum_{i=1}^{N} Q_{ij}$. These queue lengths $Q_i$ and $Q_j$ will be used as weights, applied to the corresponding nodes in the bipartite graph $G$.

### B. *i*SLIP crossbar switch scheduler

*i*SLIP [9] is a popular scheduling mechanism for virtual-output queue crossbar switches, which has now been extended in several directions, with variations such as weighted *i*SLIP and prioritized *i*SLIP. Commercial *i*SLIP switches are typically based on a combination of ideas from several of these variations. According to McKeown [9], *i*SLIP can achieve 100% throughput for uniform traffic, meaning that every output reaches maximum capacity. In this situation, a complete matching in the bipartite graph between the inputs and outputs defined by the crossbar fabric is found at every time-slot. If the traffic is not uniform, McKeown's results suggest that *i*SLIP adapts to a fair scheduling policy which never discriminates against any input queue.

Although *i*SLIP is simple to implement and uses the switch hardware efficiently, it does not provide any guarantee of real-time performance; and the determination of usefully tight delay bounds for *i*SLIP remain an open problem [12]. The best delay bound currently available is still very pessimistic [12]: for example, if every input to an $N \times N$ *i*SLIP switch has periodic real-time traffic going to every output, the known single-hop delay bound $\delta_{i\text{SLIP}}$ for packets from input $I_i$ to output $O_j$ is

$$\delta_{i\text{SLIP}} = N^2 \sum_{k} C_{i,j,k}, \tag{4}$$

where $C_{i,j,k}$ is the transmission time for each packet of the $k$th real-time flow from $I_i$ to $O_j$. See how badly this works for some likely numbers: if $N$ is 32, $C_{i,j,k}$ is the same for all links and flows, and there are 100 real-time flows going from $I_i$ to $O_j$, then the single-hop delay bound is a factor of 102400 greater than the packet transmission time.

## III. Design of a Real-Time Switch

The key issue in the design of a real-time packet switch is to bound the buffering delay while also handling the contention. This can be accomplished so long as the switching delay can be bounded. The buffering delay can be further reduced if the network operates in a synchronous manner.

In this section, we design a real-time switching algorithm that can guarantee a bounded delay with any feasible traffic. We will use the widely-adopted virtual-output queue architecture; but note that we do not require traffic to be periodic, an assumption often made in previous work [13]. We will begin by introducing the concept of clock-driven scheduling [15] as a virtual-machine task [3]–[6], [15]–[17].

### A. Clock-driven scheduling as a virtual-machine task

A widely used approach to the scheduling of real-time virtual-machine tasks (VM-tasks) is clock-driven scheduling [15]. Suppose a VM-task $(L, C)$ is served $C$ times during each clock period of $L$ time-slots. Let $f_k^{ij}$ denote the $k$th real-time flow from input port $I_i$ to output port $O_j$, where $k = 1, 2, \ldots, K_{ij}$, and $K_{ij}$ is the number of flows from $I_i$ to $O_j$. Additionally, we will associate the $f_k^{ij}$ with a VM-task $(L, C_{ijk})$, which is written in this way because $f_k^{ij}$ has $C_{ijk}$ packets to be served. Using clock-driven scheduling, the delay to $f_k^{ij}$ will be bounded as long as the switching algorithm can guarantee a worst-case bound on the time required to forward all the $C_{ijk}$ packets of the flow $f_k^{ij}$.

Let $C_{ij}$ denote the total number of packets to be forwarded from $I_i$ to $O_j$, so that $C_{ij} = \sum_{k=1}^{K_{ij}} C_{ijk}$. The sets of VM-tasks $\{(L, C_{ijk})\}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$, $k = 1, \ldots, K_{ij}$ must meet the stability condition expressed by (2) and (3); in addition we quote the feasibility condition from [13] for the real-time traffic during each clock period of $L$ time-slots, as follows:

$$\sum_{j=1}^{N} C_{ij} \leq L, i = 1, 2, \ldots, N. \tag{5}$$

$$\sum_{i=1}^{N} C_{ij} \leq L, j = 1, 2, \ldots, N. \tag{6}$$

*We do not consider infeasible traffic which does not meet these conditions, and is naturally unschedulable*. However it is possible to extend our framework, to a limited extent, to traffic that is infeasible, as long as it satisfies the stability condition in (2) and (3).

In summary, our real-time switch serves each traffic flow as a VM-task, and each VM-task is served by a slot-by-slot switching algorithm that minimizes the clearance time for any feasible traffic: we will explain this in the next section. Note that our method of clock-driven scheduling naturally makes the network operate in a synchronous manner, which is critical in reducing the amount of buffering required by a switch.

### *B. Clearance-time-optimal switching policies*

In order to design a switching algorithm with a limited delay, we need a clearance-time-optimal switching policy. Suppose that the number of packets in every queue in the system is initially $Q_{ij}(0)$, and there are no further arrivals; then we have *one-shot traffic* and the clearance time is the time required to serve every packet in the system. A switching policy that minimizes the clearance time is called a clearance-time-optimal policy. In this study, we will express the clearance time as the number of time-slots required to clear all the packets in the system. The clearance time can easily be determined as a multiple of the slot duration.

As we explained in the previous section, the crossbar constraints mean that no more than one packet can be switched at any port. Consequently, an obvious lower bound on the clearance time $T_{\text{clear}}$ is the maximum number of packets waiting at any input port:

$$T_{\text{clear}} \geq \max \left( \max_i \sum_{j=1}^{N} Q_{ij}(0), \max_j \sum_{i=1}^{N} Q_{ij}(0) \right). \tag{7}$$

It is apparent that this bound is tight, and that the minimum clearance time $T^*_{\text{clear}}$ is equal to the right-hand side of this equation.

To design a switching algorithm that can achieve this minimum clearance time $T^*_{\text{clear}}$, we first introduce a *critical-port policy*, as follows: Given a bipartite graph $G$, node $i$ is *critical* if its weight, which is the length of its queue, is no smaller than that of any other node; and a critical-port matching $M$ matches every critical port. Consequently, a critical-port scheduling policy must generate a critical matching for every time-slot. It can be shown that a critical-port policy is also clearance-time-optimal, as follows:

**Proposition 1** *A switching policy is clearance-time-optimal if and only if it is a critical-port policy.*

**Proof** For any clearance-time-optimal policy, at any time-slot $s < T^*_{\text{clear}}$, the inequality $Q_i(s) \leq T^*_{\text{clear}} - s$ holds at every port $i$. If it did not, the crossbar constraint would mean that the corresponding port could not be cleared by $T^*_{\text{clear}}$. Similarly, it is apparent that any ports at which the initial length of the queue $Q_i(0)$ was $T^*_{\text{clear}}$ will now have a queue of length $Q_i(s)$ which equals $T^*_{\text{clear}} - s$. Consequently, it can be concluded that each of the critical ports for which $t = s$ has a queue of length $T^*_{\text{clear}} - s$. If any critical ports are not served during time-slot $s$, then these ports cannot be cleared by $T^*_{\text{clear}}$. Hence, every clearance-time-optimal policy is a critical-port policy. Now, suppose we have a critical-port policy. Then, since all the ports with the highest weight, meaning the longest queues, at any time-slot are critical ports, those ports will be served and the lengths of their queues will decrease by one. Hence, a critical-port policy must also be a clearance-time-optimal policy. ■

**Remark** By adopting a critical-port policy, we can guarantee a minimal clearance time for one-shot traffic. In the next section, we will present a design framework that can guarantee a bounded delay for any feasible traffic.

An illustrative example of a critical-port switching algorithm is shown in Fig. 2. In this figure the number next to each input port $I_i$ and output port $O_j$ denotes their current weights $Q_i$ and $Q_j$ respectively. While the meaning of $Q_i$ is obvious, we must explain that $Q_j$ denotes the total number of packets at any input port *destined* for output port $O_j$. In addition, the $j$th sub-queue (numbered from the top) at each input port $I_i$ is the current number of packets $Q_{ij}$ destined for output port $O_j$. Thus it should be apparent that each $Q_j$ is the sum of the lengths $Q_{ij}$ of the sub-queues at each input port $i$. For example,

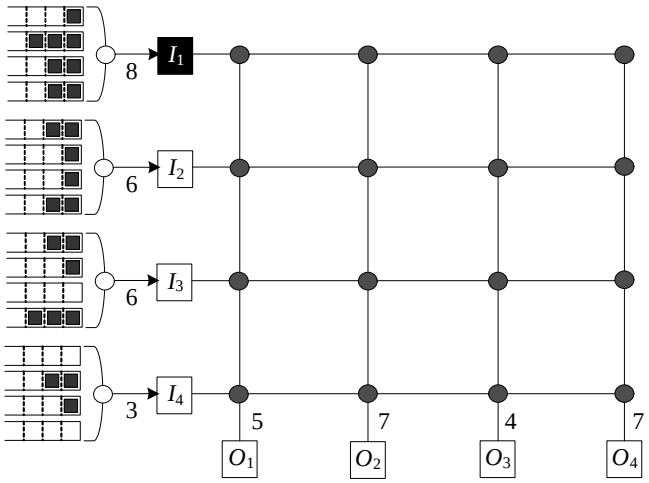



Fig. 2. Illustration of critical-port switching with a $4 \times 4$ switch fabric.

**Algorithm 1** Clock-based switching algorithm

1: **for** each clock period **do**
2: Switch the packets that arrived in the previous clock period using Algorithm 2
3: **end for**

the weight applied to output port $O_1$ is $Q_{O_1} = Q_{11} + Q_{21} + Q_{31} + Q_{41} = 1 + 2 + 2 + 0 = 5$. In the state of the queues shown in Fig. 2, input port $I_1$ has a larger weight than any other input or output port, making $I_1$ the unique critical port. Hence, a critical-port switching algorithm must now serve $I_1$ with highest priority.

### *C. Design of a real-time switch with clock-driven scheduling*

We can now use the concept of clearance time and the critical-port policies described in the previous section to design a switching algorithm that can guarantee a bounded delay for the feasible traffic defined in (5) and (6). Our particular goal is to produce a switch design framework that can achieve this bound by combining clearance-time-optimal scheduling with the VM-task architecture and the critical-port policy introduced in the previous section, in the following manner: The traffic arriving during each clock period is buffered and served in the next clock period. This achieves one-shot traffic, which is the basic requirement for clearance-time-optimal scheduling: clearance-time-optimality then guarantees that any feasible traffic is served in two clock periods. In effect, we accept an additional delay of one clock period, to ensure a deterministic delay bound of $2L$. This policy can be formalized as follows:

**Property 1 (Clock-based switching)** *Using the clock-based switching policy of Algorithm 1, any feasible traffic that satisfies* (5) *and* (6) *is guaranteed to be switched in two clock periods.*

Note that the clock period of $L$ time-slots is much shorter than the delay constraint of a typical real-time application. We will explore this issue by means of a numerical example in Section IV.

Among many possible realizations of critical-port policies, we adopt lazy heaviest-port first (LHPF) matching, which we will now explain: First, the threshold $th$ of a matching $M$ is defined as the lowest integral weight for which $M$ matches all the ports. For example, a perfect matching that switches every port with packets in its queue has $th = 1$. An LHPF matching has the lowest threshold of all possible matchings. Algorithm 2 is an implementation of LHPF matching.

The LHPF class of policies has optimal throughput [11], which means that the length of the queue at every switch is guaranteed to remain finite for any traffic that satisfies the stability condition of (2) and (3), even if that traffic does not satisfy the feasibility condition of (5) and (6). This extends the schedulablity region of the proposed switching algorithm, and this will be clarified in the numerical study in the next section.

It is necessary to explain several graph-theoretic notions that are incorporated in Algorithm 2. For a given bipartite graph, the length of a path is defined as the number of edges in that graph. For a given matching $M$ and any node $i$ not matched by $M$, an *augmenting* path from node $i$ to an unmatched node is an odd-length path $P$ whose every other edge is in $M$. An *absorbing* path from node $i$ is any path $P$ containing an even number of edges, whose every other edge is in $M$, and whose other end-point has a weight larger than that of node $i$. For any given matching $M$ and path $P$, we have $M \oplus P =: M - (M \cap P) + (M^c \cap P)$.

It has been shown [20] that the members of LHPF class of policies have a time complexity of less than $O(N^{2.5})$, compared to $O(N^3)$ [21] for MWM [21], [22]. The class of LHPF matching algorithms contains policies which are simple to implement,

**Algorithm 2** Lazy Heaviest-Port-First (LHPF) policy

INPUT: Any initial matching $M_0$
OUTPUT: LHPF matching $M^*$
// Initialization
$l \leftarrow 1$
// Iteration
**loop**
  **if** $M_{l-1}$ matches all nodes **then**
    $M^* \leftarrow M_{l-1}$
    BREAK
  **end if**
  Pick any of the highest unmatched nodes $i$ in $M_{l-1}$
  Find an augmented or absorbing path $P$ from $i$
  **if** $P$ exists **then**
    $M^* \leftarrow M_{l-1} \oplus P$
    $l \leftarrow l + 1$
  **else**
    $M^* \leftarrow M_{l-1}$
    BREAK
  **end if**
**end loop**

making them suitable for a low-complexity delay-efficient scheduler with theoretical guarantees on the throughput. Additionally, these policies are clearance-time-optimal because any critical-port policy is clearance-time-optimal as proved in Proposition 1.

## IV. Performance Evaluation

We will now compare the performance of the proposed switching scheme with that of the $i$SLIP algorithm, which we believe to be the most popular scheme at present, and one that is implemented in many commercial products to improve hardware utilization.

### A. Schedulability

First, we compare the schedulability of $i$SLIP and our scheme. In the simulations, the clock period $\mathcal{P}$ is fixed to 1 ms, the capacity of each port is 1, 10, or 100 Gb/s, the number of input ports $N$ is 4, 8, or 16, and the packet size is 10 kb. The curves in Fig. 3 show average results for 1000 simulation runs.

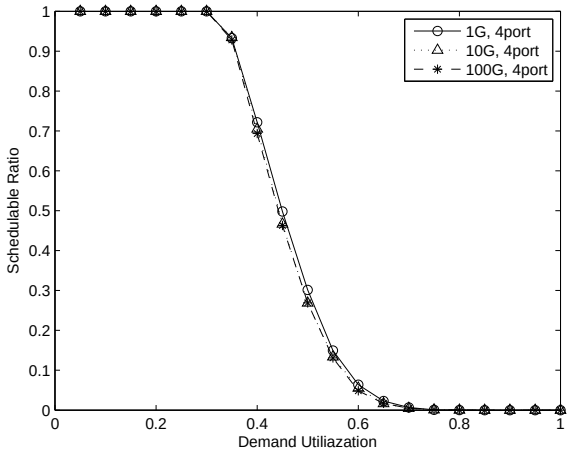
Demand utilization is the ratio between the average traffic at a port and the port capacity. Schedulablity depends on the distribution of traffic across the input ports, and is less than unity due to the cross-bar constraint explained in Section II: contention among ports stops every packet at an input port from being switched in a given time-slot. Fig. 3 shows how the schedulability drops as the demand utilization increases; eventually the traffic cannot be scheduled at all. In these simulations packets, corresponding in quantity to the demand utilization, were generated and assigned to an input port at random. Note that infeasible traffic is likely to be generated more often as the demand utilization increases, and the schedulability of the switch decreases accordingly.

Fig. 3 also indicates that our approach has better schedulability than $i$SLIP in all cases, although the difference becomes smaller as the number of input ports increases. This is of minor importance because our primary application scenario is an embedded system, in which the number of input ports is usually small. A more significant way in which our switch design is an improvement on $i$SLIP is that it guarantees all feasible traffic will be switched in two clock periods, as we have shown. As a result, within the range of demand utilization for which all the traffic arriving at the switch is expected to be feasible, our switching algorithm bounds the delay incurred at the switch. For example, we can infer from Fig. 3 that if the clock period is 1 ms and the demand utilization is less than 0.3, all the traffic arriving at the switch is expected to be pass through in 2 ms regardless of the port capacity of the switch and the number of ports.

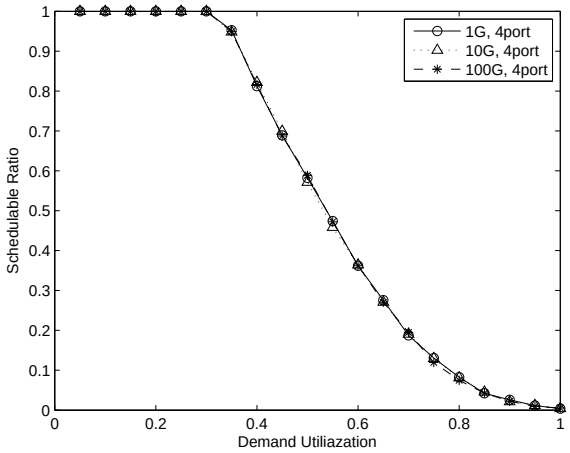
### B. Clearance time

We now compare the clearance time of the proposed scheme to that of the $i$SLIP scheme using the one-shot traffic model described in Section III-B, while focusing on the port capacity of 1 Gb/s. Fig. 4 shows the clearance time of each scheme against demand utilization. Again we have averaged 1000 simulation runs.

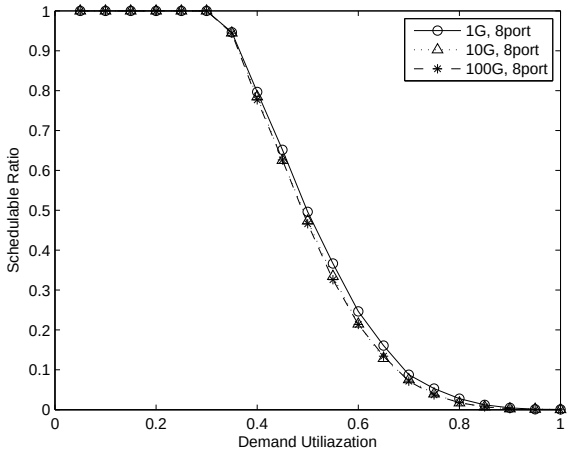
Fig. 4 shows that the clearance time increases as the number of input ports increases, because more contentions occur, as we would expect. We can also see that the clearance time increases almost linearly with the demand utilization for both schemes, but the gradient is steeper with $i$SLIP, which means that our switch algorithm provides better performance in terms of switching delay.



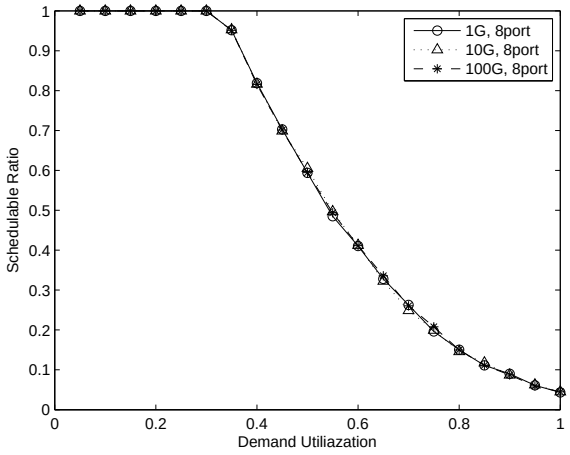

(a) $i$SLIP



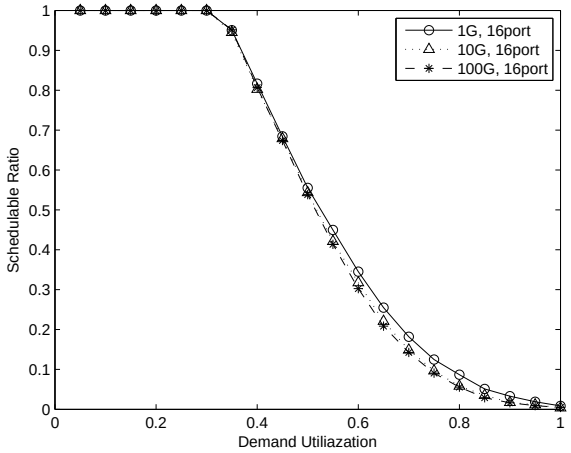

(b) Proposed scheme



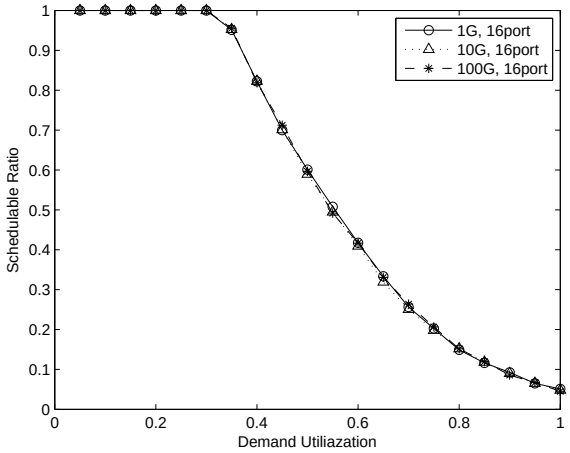

(c) $i$SLIP




(d) Proposed scheme

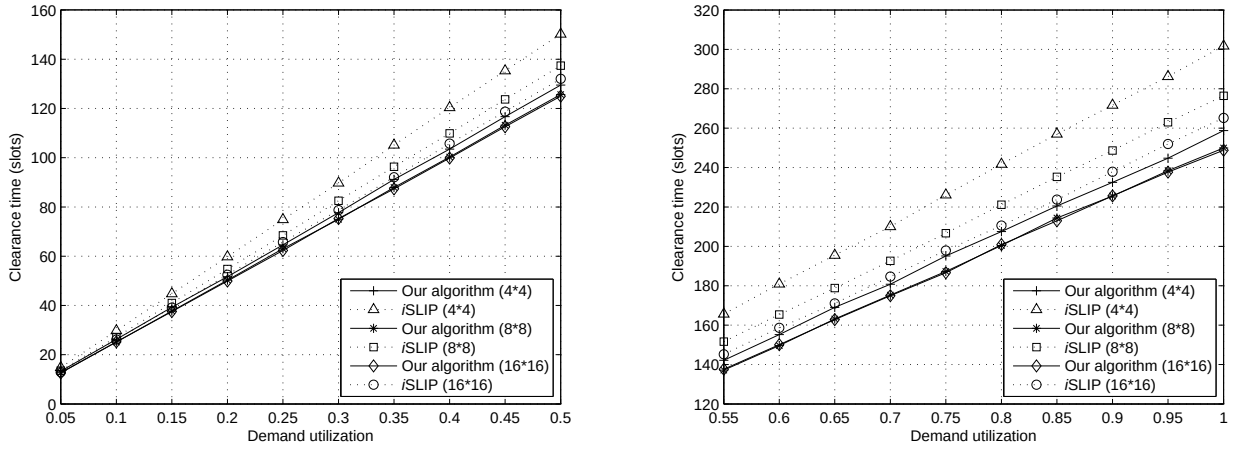



(e) $i$SLIP




(f) Proposed scheme

Fig. 3. Schedulability of $i$SLIP and the proposed scheme with port capacities of 1, 10, and 100 Gb/s: (a) and (b) 4 input ports; (c) and (d) 8 input ports; and (e) and (f) 16 input ports.




(a) Demand utilization below 0.5

(b) Demand utilization above 0.5

Fig. 4. Comparison between the clearance time of $i$SLIP and the proposed scheme when the number of input ports $N$ is 4, 8, and 16 and the port capacity is 1 Gb/s.

### C. End-to-end switching delay

Let $\delta_{SW,i}$ be the switching delay introduced by a switch $i$. Since the propagation delays between switches are negligible, the end-to-end delay $\delta_{\text{E2E}}$ using the proposed switches for any feasible traffic can be bounded as follows:

$$\delta_{\text{E2E}} = \sum_{i=1}^{H} \delta_{\text{SW},i} \leq 2H\mathcal{P}, \tag{8}$$

where $H$ is the hop count (a number of switches each packet traverse), and $\mathcal{P}$ is the clock period. This holds because any feasible traffic satisfying (5) and (6) is guaranteed to be switched in two clock periods.

Assuming the clock period is 1 ms and the maximum hop count is 15, any feasible traffic is guaranteed to be switched in 2 ms at each crossing-point in the switch, and the resulting end-to-end delay is also guaranteed to be less than 30 ms. This is not affected by the number of ports or their capacity. This period is significantly less than the end-to-end delay of 50ms allowable in typical industrial real-time control and automation applications, such as sensing and actuation [18] or video monitoring [19]. Conversely, even the bound on the *single-hop* delay in an $i$SLIP switch, obtained from (4), may frequently exceed 100ms in a similar situation.

## V. Related Work

There has been extensive research on the design of network switches [7], [9]–[11], but most of it has been focused on the development of efficient switching algorithms, within a graph-theoretic framework, in order to realize a stability region suitable for general traffic. In pursuit of this goal, an asymptotic logarithmic delay bound can be derived from the stability condition [10]. A more recent switching policy [11] has been shown to be clearance-time-optimal as well as throughput-optimal (and this policy is incorporated into our framework). However, very few of these studies have explicitly considered real-time applications.

Network infrastructures for real-time communication have typically met hard constraints on delay by prioritization within switches. Internet switches usually have 4 to 8 priority levels, but this is insufficient to provide hard real-time guarantees. Switches for real-time systems are typically of a very different design from the commercially-available routers used for Internet traffic. Nevertheless, the full or partial reuse of hardware is a strong driver in terms of purchasing and maintenance costs.

Research on network infrastructure for real-time applications has usually focused on specific, small-scale network architectures. For example, prioritized bus and ring networks have been used in small real-time systems [23], [24], but they are not designed for high-speed network backbones, such as those of WANs. Rexford et al. [8] proposed a router for real-time communication, but this was designed to support deadline-based scheduling, which requires significantly different hardware; nor is this router designed for the backbones of high-speed networks. The same remark applies to the real-time Ethernet switch proposed by Venkatramani et al. [25].

Considerable effort has been devoted to analyzing the performance of high-speed switches and routers, and obtaining delay bounds [26], [27]. The scheduling of crossbar switches can be reduced to the search for a matching in a graph, and fast matching algorithms have been developed [28]. These are based on stochastic models of traffic patterns, and the asymptotic performance bounds that they provide are not sufficient for industrial systems that require greater predictability.

There have been some recent studies on the design of real-time switches [12], [13]. In particular, an efficient switch design for real-time applications has been proposed by Qixin et al. [13], who provide a mechanism for guaranteeing the allocation of a certain number of communication slots to a task over a fixed time interval, under the assumption of deterministic and periodic real-time traffic. Realizing this mechanism involves only minimal modifications, or even simplifications, of the $i$SLIP scheme. This work provided us with some inspiration but our approach is significantly different because we are able to guarantee a delay bound for any feasible traffic, including non-periodic traffic. Another difference worth mentioning is that in our design, all flows from the same input port to the same output port share one queue; while the design in [13] carries out less scalable per-flow queueing.

## VI. Conclusion and Prospects

We have proposed a design framework for a real-time switch which is based on the concept of clock-driven scheduling. Our switch serves traffic using real-time virtual-machine tasks, which simplifies analysis, provides isolation from other system operations, and facilitate further hierarchical scheduling and flow aggregation. The flows in most industrial real-time networks rarely change, so a switch only needs to be configured initially to suit a particular real-time schedule, and a polynomial-time algorithm can then be used

Our real-time switch has several features that distinguish it from previous proposals: First, it can provide a guaranteed delay for any feasible traffic whether it is periodic or not. Second, we explicitly consider realistic system parameters, so as to facilitate practical hardware-software co-design. Our simulations confirm that our design gives a schedulablity and clearance time that compare favorably with the $i$SLIP crossbar scheduler, which is already widely implemented in commercial products.

There are several avenues for future research. In particular, we believe that our framework for switch algorithm design might be extended to the general problem of a switching network with numerous sensing and monitoring devices, of the sort seen in avionics systems. The pursuit of this goal could usefully draw on a number of recent research contributions to networking research.

## References


[1] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and Obligations for Physical Computing Systems," *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.

[2] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-Physical Systems: a New Frontier," *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'08)*, June 2008.

[3] Z. Deng and J. W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS'97)*, Dec. 1997, pp. 308–319.

[4] T.-W. Kuo and C.-H. Li, "A Fixed-Priority-Driven Open Environment for Real-Time Applications," *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS'99)*, Dec. 1999, pp. 256–267.

[5] G. Lipari and E. Bini, "Resource Partitioning among Real-Time Applications," *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'03)*, Jul. 2003, pp. 151–158.

[6] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'03)*, Dec. 2003, pp. 2–13.

[7] T. Weller and B. Hajek, "Scheduling Nonuniform Traffic in a Packet-Switching System with Small Propagation Delay," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 813–823, Dec. 1997.

[8] J. Rexford, J. Hall, and K. G. Shin, "A Router Architecture for Real-Time Communication in Multicomputer Networks," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1088–1101, Oct. 1998.

[9] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[10] M. J. Neely, E. Modiano, and Y.-S. Cheng, "Logarithmic Delay for $N \times N$ Packet Switches Under the Crossbar Constraint," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 657–668, June 2007.

[11] G. R. Gupta, S. Sanghavi, and N. B. Shroff, "Node Weighted Scheduling," *Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems (Sigmetrics'09)*, June 2009, pp. 97-108.

[12] S. Gopalakrishnan, M. Caccamo, and L. Sha, "Switch Scheduling and Network Design for Real-Time Systems," *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, Apr. 2006, pp. 289–300.

[13] Q. Wang, S. Gopalakrishnan, X. Liu, and L. Sha, "A Switch Design for Real-Time Industrial Networks," *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, Apr. 2008, pp. 367–376.

[14] L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*. Morgan Kaufmann, 2000.

[15] J. W.-S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[16] R. Davis and A. Burns, "Hierarchical Fixed Priority Preemptive Scheduling," *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'05)*, Dec. 2005, pp. 389–398.

[17] R. Davis and A. Burns, "Resource Sharing in Hierarchical Fixed Priority Pre-Emptive Systems," *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS'06)*, Dec. 2006, pp. 257–270.

[18] B. Fisher, S. Fels, K. MacLean, and T. Munzner, R. Rensink, "Seeing, Hearing, and Touching: Putting It All Together," *ACM Siggraph'04 Course*, Aug. 2004.

[19] B. Fisher, S. Fels, K. MacLean, T. Munzner, and R. Rensink, "Exploiting Perception in High-Fidelity Virtual Environments," *ACM Siggraph'06 Course*, Aug. 2006.

[20] A. Mekkittikul and N. McKeown, "Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," *Proceedings of the IEEE Infocom'98*, 1998, pp. 792–799.

[21] R. Karp and J. Hopcroft, "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[22] D. Shah and D. Wischik, "Optimal Scheduling Algorithms for Input-Queued Switches," *Proceedings of the IEEE Infocom'06*, Apr. 2006.

[23] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Real-Time Scheduling Support in Futurebus+," *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'90)*, Dec. 1990, pp. 331–340 .

[24] S. Gopalakrishnan, L. Sha, and M. Caccamo, "Hard Real-Time Communication in Bus-Based Networks," *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'04)*, Dec. 2004, pp. 405–414.
[25] C. Venkatramani and T. Chiueh, "Design and Implementation of a Real-Time Switch for Segmented Ethernets," *Proceedings of IEEE International Conference on Network Protocol (ICNP'97)*, Oct. 1997, pp. 152–161.
[26] D. Shah, P. Giaccone, E. Leonardi, and B. Prabhakar, "Delay Bounds for Combined Input and Output Switches with Low Speedups," *Performance Evaluation*, vol. 55, no. 1/2, pp. 113–128, Jan. 2004.
[27] D. Shah, P. Giaccone, and E. Leonardi, "Throughput Region of Finite-Buffered Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol 18, no. 2, pp. 251–263, Feb. 2007.
[28] S. Deb, D. Shah, and S. Shakkottai, "Fast Matching Algorithms for Repetitive Optimization: an Application to Switch Scheduling," *Proceedings of the Conference on Information, Sciences and Systems (CISS'06)*, Mar. 2006, pp. 1266–1271.