A Switch Design for Real-Time Industrial Networks

Qixin Wang*, Sathish Gopalakrishnan†, Xue Liu‡, and Lui Sha*
* Department of Computer Science, University of Illinois at Urbana-Champaign
† Department of Electrical and Computer Engineering, University of British Columbia
‡ Department of Computer Science, McGill University

Abstract

The convergence of computational activities and physical work is the theme for next generation networking research. This trend calls for real-time network infrastructure, which requires a high-speed real-time WAN to serve as its backbone. However, commercially available high-speed WAN switches (routers) are designed for best-effort Internet traffic. A real-time switch design for the aforementioned networks is missing. We propose a real-time switch design using a crossbar switching fabric. The proposed switch can be implemented by making minimal modification, or even simplification, to the widely implemented iSLIP crossbar switch scheduler. Our real-time switch serves periodic and aperiodic traffic with real-time virtual machine tasks, which simplifies analysis, provides isolation, and facilitates future hierarchical scheduling and flow aggregation. Taking advantage of the fact that most industrial real-time network flows rarely change, our switch is better adapted to providing high bandwidths and low latencies.

1 Introduction

A theme for next generation networking research, such as the Real-Time and Embedded GENI [27] initiative and the Cyber-Physical Systems [1, 15, 2, 33] initiative, is to enable the convergence of computers with the physical world. A target application of this convergence is industrial real-time control and automation.

As pointed out in Wang et al. [35], industrial real-time control/automation needs wired real-time *wide area networks* (WAN) as the communication backbone. To build such WANs, we need real-time switches (routers), whereas commercially available switches are tailored towards best-effort Internet traffic rather than real-time systems.

There are three approaches to building a switch: output queueing, input queueing, and *virtual output queueing* (VOQ).

In output queueing, queueing only takes place at the output ports (simplified as *outputs* in the following). When a packet arrives at an input port (simplified as *inputs* in the following), it immediately goes to the queue at its destined output. Due to its simplicity, most QoS scheduling algorithms, such as WFQ [23], WF^{2}Q [3], Deficit Round-Robin [32] etc., assumes output queueing [20].

Output queueing, however, creates a data bus bottleneck. Since there is no queue at the inputs, the data bus must deliver every arriving packet to output queue immediately. In the worst case, every input may reach its maximum capacity, and all incoming packets may go to a same output. Therefore, the data bus connected to each output must provide a capacity no less than total capacity of all inputs. Suppose a switch has N inputs, each with a data line rate of C, then the data bus connected to each output must provide a capacity of $N \times C$. We call this N *speed-up*. Such a speed-up makes output queueing undesirable for high-speed switches or switches with large number of ports (N) because of the challenges of developing high-speed memory banks.

In contrast to output queueing, input queueing buffers packets in queues at the inputs. This avoids the need for speedup in the switch, but suffers from *head of line* (HOL) blocking: if packets going to other outputs are blocked at the head of the input queue, a packet to output j must wait for the depletion of this backlog before it is transferred to output j, even though output j is idle. It is well known that if each input queue is first-in-first-out (FIFO), HOL blocking can limit the throughput to just 58.6% [16].

The solution to the HOL problem is to deploy *virtual output queueing*, where each input maintains a virtual output queue for each output. VOQs eliminate HOL blocking, but packets from different inputs' VOQs still contend for the same output. Various schemes are proposed to reduce this contention, so as to improve the hardware utilization. To our best knowledge, the most popular scheme is iSLIP [22, 21, 9], which is elaborated upon in Section 2. Although iSLIP efficiently utilizes the switch hardware and is sim-

ple to implement, it does not provide real-time guarantees. In fact, real-time high-performance switch design is still an open problem [13].

In this article we describe a design of a real-time switch by making minimal modifications to iSLIP, or even by simplifying iSLIP. This design benefits switch manufacturers since iSLIP is already widely implemented in commercial products, and the minor modifications can be easily incorporated into the manufacturing process. Our approach is to define operations that allow a switch to serve each link l for C_l units of time every M units of time. This enables hierarchical scheduling, which can then be analyzed using one of several known techniques [8, 31, 19].

In the following, Section 2 describes the iSLIP scheme; Section 3 proposes our switch design for industrial real-time communications; Section 4 evaluates our design; Section 5 discusses related work; and Section 6 concludes the paper.

2 Crossbar Switches and iSLIP

To support input queueing or VOQ, most high-performance switches use a crossbar hardware fabric [24] (Fig. 1). The data bus from each input (the horizontal line segments in the figure) intersects with the data bus of each output (the vertical line segments). The intersections can be turned on or off during runtime by the switch scheduling logic. To facilitate the scheduling logic, crossbar switches transfer packets in fixed-size fragments called *cells*; and the time to transfer one cell across the crossbar fabric is called a *cell-time*. Therefore, the scheduling logic works periodically: it determines a matching between inputs and outputs at the beginning of each cell-time; then all scheduled cells are transferred synchronously across the crossbar fabric, taking one cell-time; and then the next period starts, so on and so forth.

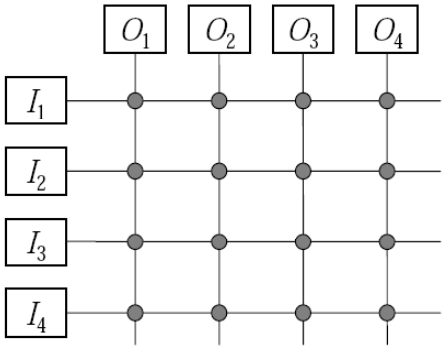


Figure 1. Crossbar Switch Hardware Fabric

iSLIP [22, 21] is a popular scheduling mechanism for VOQ crossbar switches. Without loss of generality, suppose a switch consists of N inputs $I_1 \sim I_N$ and N outputs $O_1 \sim O_N$ (denoted as an "$N \times N$ switch" in the following). Under iSLIP, every input I_i maintains a circular list of outputs $O_1 \sim O_N$, with pointer a_i pointing to O_1 initially. This circular list is called the input's *round-robin schedule*. The output pointed to by a_i has the highest priority, the next output (modulo N) has the next highest priority, and so on. In the same way, every output O_j also maintains a round-robin schedule of inputs, with pointer g_j pointing to the highest priority input, the next input (modulo N) has the next highest priority, and so on.

With the above data structures, the basic iSLIP runs following steps [22]:

Step 1 *Request*. Each unmatched input sends a request to every output for which it has a queued cell.

Step 2 *Grant*. If an unmatched output receives any requests, it grants the requesting input with the highest priority in the output's round-robin schedule. The output notifies each input whether or not its request was granted. The pointer g_i to the round-robin schedule is incremented (modulo N) to one location beyond the granted input *if, and only if, the grant is accepted in Step 3*.

Step 3 *Accept*. If an input receives any grants, it accepts the granting output with the highest priority in the input's round-robin schedule. The pointer a_i to the round-robin schedule is incremented (modulo N) to one location beyond the accepted output.

Since some granting outputs may not be accepted, iSLIP may carry out up to N iterations of Request-Grant-Accept at the beginning of each cell-time to increase the number of matching.

The original iSLIP mechanism [22, 21] also accommodates several variations such as weighted iSLIP and prioritized iSLIP. Different commercial iSLIP switches may implement certain subsets of these variations. According to McKeown [21], iSLIP can achieve 100% throughput (i.e., every output reaches maximum capacity; in other words, the bipartite graph between inputs and outputs defined by the crossbar fabric reaches full match for every cell-time) for uniform traffic, and quickly adapts to a fair scheduling policy that never starve any input queue for non-uniform traffic.

However, obtaining accurate delay bounds for iSLIP is still an open problem. The best known iSLIP delay bound is still "very pessimistic" [13]. For example, if in an $N \times N$ iSLIP switch, every input has periodic real-time traffic going to every output, the known single hop delay bound for packets from input I_i to output O_j is

$$d = N^2 \sum_k C_{ijk}, \tag{1}$$

where C_{ijk} is the per packet transmission time of the kth real-time flow going from I_i to O_j. Suppose $N = 32$, C_{ijk} is the same for all links and flows, and if there are 100 real-time flows going from I_i to O_i, then the single hop delay bound is at least 102400 times that of a packet transmission time.

3 A Real-Time Switch Design

To support real-time, we propose a real-time switch design by making minimum modifications to iSLIP. Interestingly, our design simplifies iSLIP rather than complicates it.

Firstly, we observe a large body of research on serving a real-time task or task-set with a real-time *virtual machine task* (VM-task) [20, 8, 17, 31, 19, 5, 6]. One simple and widely implemented form is clock-driven scheduling [20], where a VM-task (M, C) indicates that a real-time task or task-set is served C time units during each clock-period of M time units.

Using clock-driven scheduling, we may serve the kth real-time flow f_{ijk} from input I_i to output O_j in a crossbar switch with a VM-task (M, C_{ijk}) (*unless explicitly noted, the default time unit is "cell-time"*), where $k = 1, 2, \ldots, K_{ij}$, and K_{ij} is the total number of real-time flows going from I_i to O_j. That is, as long as the switch forwards C_{ijk} cells from I_i to O_j for f_{ijk} in each M cell-time clock-period, packets of f_{ijk} shall meet their local deadlines.

Secondly, we observe that the iSLIP request-grant-accept negotiation between inputs and outputs is for Internet random traffic, which changes frequently. *If the traffic rarely changes and is periodic, as that of real-time flows in industrial networks, there is no need for a request-grant-accept negotiation.* Instead, deterministic grants (or accepts) alone suffice. We only need to work out a conflict-free grant (or accept) schedule during configuration-time.

In summary, our real-time switch shall serve each real-time flow with a real-time VM-task, and the VM-task is served with deterministic grant (or accept). We elaborate such design in the following.

3.1 Per-flow VOQ

Our proposed real-time switch is an $N \times N$ crossbar VOQ switch. However, to control jitter for simple *end-to-end* (E2E) delay guarantee, we deploy *per-flow virtual output queueing* (per-flow VOQ), instead of combining all cells at input I_i destined for output O_j in one virtual output queue. In other words, if there are K_{ij} flows going from I_i to O_j, then for O_j, we maintain K_{ij} queues at I_i for each flow respectively.

The overall buffer requirements at the switch do not change (much) because of the per-flow VOQs; the same packets that would have been buffered at one VOQ are held in different buffers depending on their flow id. Flow differentiation can be performed in conjunction with IP lookup and output port identification, therefore the hardware complexity and the per-cell processing time overhead increase only marginally. It is also worth mentioning that per-flow VOQs are simple FIFO queues. We do not need to maintain per-flow state information, or perform sorting (as most timestamp based QoS schemes, such as WFQ [23] and WF^{2}Q [3], do), which may affect performance.

3.2 Traffic demand

All traffic demand in our real-time switch is abstracted by the clock-driven scheduling of VM-tasks (see Section 3.5 Equation (4)). According to clock-driven scheduling, the kth real-time flow f_{ijk} from I_i to O_j can be served by VM-task $\tau_{ijk} = (M, C_{ijk})$. That is, during each clock-period of M cell-time, C_{ijk} cells are forwarded from I_i to O_j for flow f_{ijk}.

Denote $C_{ij} \stackrel{def}{=} \sum_{k=1}^{K_{ij}} C_{ijk}$. That is, I_i needs to forward C_{ij} cells to O_j during each clock-period. Then the entire VM-task set $\{(M, C_{ijk})\}$ ($i = 1 \sim N, j = 1 \sim N, k = 1 \sim K_{ij}$) must meet the following constraints to be *feasible*:

Constraint 1 Feasible input utilization:

$$\sum_{j=1}^{N} C_{ij} \leq M, i = 1, 2, \ldots, N. \quad (2)$$

Constraint 2 Feasible output utilization:

$$\sum_{i=1}^{N} C_{ij} \leq M, j = 1, 2, \ldots, N. \quad (3)$$

Infeasible VM-task sets are unschedulable, and we do not consider them.

3.3 Runtime scheduling

Corresponding to the M cell-time clock-period, each output O_j maintains a round-robin schedule S_j^{out} of M elements. The gth ($1 \leq g \leq M$) element dictates the input from which O_j fetches a cell at the gth cell-time of a M cell-time clock-period. $S_1^{out} \sim S_N^{out}$ are *conflict-free*, meaning at any cell-time of the M cell-time clock-period, no two outputs fetch cells from the same input; and S_j^{out} ($j = 1 \sim N$) has exactly C_{ij} ($i = 1 \sim N$) elements for input I_i, meaning O_j fetches C_{ij} cells from I_i in each M cell-time clock-period. We will describe how to derive $S_1^{out} \sim S_N^{out}$ in a later subsection (Section 3.4).

Correspondingly, each input I_i maintains a round-robin schedule S_{ij}^{in} of C_{ij} elements for each output O_j. The ath

($a = 1, 2, \ldots, C_{ij}$) element of S_{ij}^{in} indicates the per-flow VOQ to send a cell from, when I_i is to connect O_j for the ath time during the M cell-time clock-period. That is, S_{ij}^{in} has C_{ijk} elements for f_{ijk} ($k = 1 \sim K_{ij}$) respectively; and *these elements are arbitrarily ordered*.

Input I_i also maintains a pointer ρ_{ij} to S_{ij}^{in}, initially pointing to the first element of S_{ij}^{in}.

With the above settings, our proposed real-time switch only executes two steps at the beginning of the gth ($g = 1, 2, \ldots, M$) cell-time of each M cell-time clock-period:

Step 1 *Grant*. Output O_j grants the input indicated by the gth element of S_j^{out}.

Step 2 *Accept*. On receiving a grant from O_j, input I_j sends O_j the head cell (or null if the queue is empty) of per-flow VOQ indicated by pointer ρ_{ij}. ρ_{ij} is increased by 1 (modulo C_{ij}).

The "Request" step in the original iSLIP disappears; and because $S_1^{out} \sim S_N^{out}$ are conflict-free, a "Grant" is always accepted , which eliminates the need of N iterations. Therefore, our real-time switch incurs $O(1)$ computation during runtime, and is simpler than iSLIP.

3.4 Configuration-time scheduling

During configuration-time, we need to work out conflict-free round-robin schedules $S_1^{out} \sim S_N^{out}$. In this section, we show that any feasible VM-task set has a conflict-free schedule that can be computed in polynomial time.

Theorem 1 *A VM-task set $\{(M, C_{ijk})\}$ has conflict-free schedules $S_1^{out} \sim S_N^{out}$ if and only if the VM-task set is feasible (see Constraint 1 and 2 for the definition of "feasible"); and any feasible VM-task set can be scheduled within $O(N^4)$ time, where N is the number of input (also output) ports.*

Proof: 1) Sufficiency: The scheduling of feasible VM-task set $\{(M, C_{ijk})\}$ can be reduced to a *preemptive open shop scheduling* (POSS) problem [12].

The preemptive open shop scheduling problem involves n tasks, denoted by the set $\{\tau_i\}$, and η machines ($n \geq 1, \eta \geq 1$). τ_i has η subtasks, represented by the set $\{\tau_{ij}\}$, such that τ_{ij} has to be executed on machine j. Tasks can be preempted, and no restrictions are placed on the order in which the subtasks are executed. No machine can operate on more than one task at a time, and no task can execute on more than one machine at the same time. If t_{ij} is the time required by subtask τ_{ij} on machine j, we can obtain the following quantities:

$$T_j = \sum_{i=1}^{n} t_{ij} = \text{ total time on machine } j, \forall 1 \leq j \leq \eta,$$

$$L_i = \sum_{j=1}^{\eta} t_{ij} = \text{ total time for task } i, \forall 1 \leq i \leq n.$$

The optimal finish time for all operations is $\alpha = \max_{i,j}\{T_j, L_i\}$, which can always be achieved according to the scheduling algorithm suggested by Gonozalez and Sahni [12]. The scheduling algorithm has a time complexity of $O(\beta^2)$, where β is the number of non-zero subtasks.

Regard all VM-tasks forwarding cells from I_i to O_j as one VM-task (M, C_{ij}), where $C_{ij} \stackrel{def}{=} \sum_{k=1}^{K_{ij}} C_{ijk}$; and regard each output O_j ($j = 1, 2, \ldots, N$) as a POSS machine. For each given I ($I = 1, 2, \ldots, N$), regard VM-task subset $\{(M, C_{ij}) | i == I\}$ as a POSS task that runs $C_{I1}, C_{I2}, \ldots, C_{IN}$ time units on POSS machine $O_1, O_2, \ldots, O_N$ respectively. According to the POSS algorithm proposed by Gonzalez and Sahni [12], any feasible VM-task set $\{(M, C_{ij})\}$ can always finish within $\alpha = M$ time units, i.e., any feasible VM-task set $\{(M, C_{ij})\}$ is schedulable; and the scheduling complexity is $O(N^4)$ since $\beta \leq N^2$.

2) Necessity: According to the definition given in Constraint 1 and 2, any infeasible VM-task set either exceeds the capacity of an input, or an output, hence is not schedulable.

■

Although Gonzalez and Sahni's POSS algorithm is polynomial and optimal (in the sense it schedules any feasible VM-task set), its implementation is non-trivial. In the following, we propose a sub-optimal but simpler scheduling algorithm, which has straight-forward graphical meaning.

As in the proof of Theorem 1, we first regard all VM-tasks forwarding cells from I_i to O_j as one VM-task (M, C_{ij}), where $C_{ij} \stackrel{def}{=} \sum_{k=1}^{K_{ij}} C_{ijk}$. We can graphically represent the VM-task set $\{(M, C_{ij})\}$ ($i, j = 1, 2, \ldots, N$) as a *demand matrix* (see Fig. 2):

Definition 1 (Demand matrix) *A demand matrix $\mathbf{D} = \{d_{jg}\}$ is a $N \times M$ matrix, with each element $d_{jg} \in \{0, 1, 2, \ldots, N\}$. In the jth ($j = 1, 2, \ldots, N$) row, C_{ij} elements are colored i ($i = 1, 2, \ldots, N$) respectively; the remaining elements are colored* 0*, meaning* empty slots*; and the elements in the row are arbitrarily ordered.*

In a demand matrix, each non-zero element in the jth row indicates the input from which output O_j shall fetch a cell during a M cell-time clock-period.

Naturally, each demand matrix has the following property:

Property 1 (Feasible demand matrix) *Suppose the demand matrix* $\{d_{jg}\}_{N\times M}$ *represents a VM-task set* $\{(M, C_{ij})\}$. *Then* $\{(M, C_{ij})\}$ *is feasible if and only if for each non-zero color* $i \in \{1, 2, \ldots, N\}$, *the demand matrix has no more than* M *elements colored in* i. *Such a demand matrix is called a* feasible demand matrix.

In addition, a demand matrix can represent a schedule.

Definition 2 (Schedule (matrix)) *We can regard a demand matrix* $\mathbf{D} = \{d_{jg}\}_{N\times M}$ *as a schedule if each element* d_{jg} *(*$d_{jg} \neq 0$*) implies that output* O_j *grants input* $I_{d_{jg}}$ *at the* g*th cell-time of each* M *cell-time clock-period, and no two elements in each column of* $\mathbf{D}$ *have the same non-zero color. We shall also call such demand matrix a* schedule matrix.

The jth ($j = 1 \sim N$) row of a schedule matrix represents schedule S_j^{out}. Since a schedule matrix one-to-one mapps to a valid schedule, "schedule matrix" and "schedule" become interchangeable terms.

With the help of the schedule matrix, configuration-time scheduling now has graphical meaning: given a feasible demand matrix $\mathbf{D}$, configuration-time scheduling permutates the elements in each row of $\mathbf{D}$ to produce a schedule (a matrix where no two elements in each column have the same non-zero color). Fig. 2 illustrates the relationship between demand matrix, scheduling algorithm, and schedule matrix.

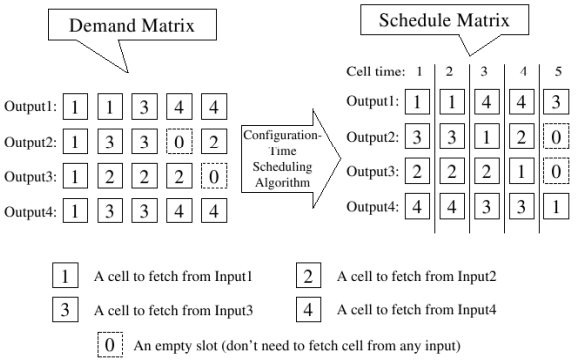


Figure 2. An example illustrates the relationship between Demand Matrix, Configuration-Time Scheduling Algorithm, and Schedule Matrix, where number of ports $N = 4$, and a clock-period is $M = 5$ cell-time.

With the help of the above graphical tools, we can devise many simpler sub-optimal scheduling algorithms. In Fig. 3, we propose the *least slack* (LS) algorithm. The term "*slack*" means the following: if a row of a demand matrix has κ elements colored c, then color c has a slack of $(M - \kappa)$ in this row.

1. **LeastSlack**($\mathbf{D}$/* the $N \times M$ demand matrix, passed by copy */):
2. Initiate schedule matrix $\mathbf{S}$ as an $N \times M$ empty matrix.
3. **while** $\mathbf{D}$ has non-zero colored element **begin**
4. Of all rows of $\mathbf{D}$, pick the non-zero color c that has least slack (break ties arbitrarily).Denote the corresponding row index as j.
5. Move the elements of color c in the jth row of $\mathbf{D}$ to the earliest (i.e., empty slots with the smallest column indices) and conflict-free empty slots in the jth row of $\mathbf{S}$.
   break the **while** loop if cannot find any conflict-free empty slot.
6. **end**.
7. **if** all non-zero colored elements of $\mathbf{D}$ are removed, **return** $\mathbf{S}$;
8. **else return** cannot find schedule.

Figure 3. Least Slack (LS) Scheduling. The term "conflict-free" means no two non-zero colored elements in each column of a matrix have the same color.

For LS-scheduling algorithm, let tuple (r, c) correspond to the slack of color c in the rth row of demand matrix. During initialization, we shall create and sort these N^2 tuples into a list $\mathcal{L}$ with ascending slack, which takes $O(N^2 \log N + NM)$ time. Then Step 3 only takes $O(1)$ time: just to check whether $\mathcal{L}$ is empty; and Step 4 only takes $O(1)$ time: just remove the head of $\mathcal{L}$. Step 5 takes $O(M)$ time, if we maintain an $N \times M$ boolean array F for $\mathbf{S}$ with F_{cg} indicating whether the gth column of $\mathbf{S}$ already has an element colored c. The **while** loop from Step 3 to Step 6 loops at the most N^2 times. Therefore, the time complexity of LS-scheduling is $O(N^2 \log N + NM + N^2 M) = O(N^2 M)$.

3.5 E2E Delay Guarantee

In this section, we analyze the E2E delay guarantee provided by our proposed real-time switch for industrial real-time applications. In these applications, the dominate traffics are periodic traffics such as sensing, actuating, and video monitoring. Aperiodic traffics can be served by periodic VM-tasks [20]. As a result, we shall assume all traffics are periodic in the following analysis.

We assume that all the switches in the industrial network comply with the proposed real-time switch scheme. We also assume that all switches adopt the same clock-period of $\mathcal{P} \equiv 1$ (ms) and have the same per port capacity. Assume a unanimous cell size of 500 bits[1]. If the per port capacity is 1Gbps, 10Gbps, or 100Gbps, then a clock-period of 1ms corresponds to an M of 2000, 20000, and 200000 cell-time respectively.

Suppose a real-time flow f at least needs to send a message of E cells every T cell-time, denoted as $f = (T, E)$. Note E, T may be real numbers instead of integers. Then

[1]Real-world switches usually use cell size of 512 bits. We use cell size of 500 bits for narrative simplicity.

we over provision f with VM-task $\tau_f = (M, C)$, where

$$C = \left\lceil \frac{E}{\lfloor T/M \rfloor} \right\rceil. \quad (4)$$

That is, each message of f is forwarded as $R \stackrel{def}{=} \lfloor T/M \rfloor$ packets, and each packet consists of C cells. Note, since M cell-time equals 1ms, for most industrial real-time applications, $T > M$.

Suppose f traverses H hops of our proposed real-time switches, each schedules a VM-task of (M, C) to forward the packets of f.

To derive the E2E delay, we start from the first hop. Since the first hop forwards exactly C cells for flow f in any consecutive M cell-time, whenever a new message of f arrives, the first packet of the message takes at the most $M + 1$ cell-time to be forwarded, the additional 1 is because the packet may arrive during the middle of a cell-time. After that, the switch forwards a next packet every additional M cell-time, until all R packets are forwarded. Same thing happens in the following switches. Therefore, the worst case E2E delay D (ms) for the message is

$$\begin{aligned} D &= \sum_{h=1}^{H} (M+1)\delta + (R-1)M\delta \\ &= (H + R - 1)\mathcal{P} + H\delta, \end{aligned} \quad (5)$$

where δ (ms) is one cell-time in the unit of millisecond.

The first item of Equation (5) is the worst case E2E delay for the first packet. After the first packet arrives at the receiver end, every additional M cell-time, a subsequent packet arrives, until all R packets arrive.

Note the above analysis can be easily extended to cases where the proposed real-time switches have different per port capacities, which are not discussed in this paper due to page limits.

4 Evaluation

4.1 Efficiency of M Cell-Time Clock-Period

A natural question on the proposed real-time switch is: how efficient is it to enforce a unanimous M cell-time clock-period?

We evaluate this in the context of industrial real-time control/automation traffic.

There are two types of real-time traffic in industrial real-time control/automation: real-time sensing/actuating traffic and real-time video traffic.

Real-time sensing/actuating traffic involves low data-throughput. A typical sensing/actuating flow generates a $1 \sim 5$kbit message every 10(ms). The maximal allowed E2E delay is usually 50ms [10, 11].

Real-time video traffic involves high data-throughput. A typical video flow generates one message (a.k.a. "frame") every 30ms, and the message size is in the worst case $120 \sim 240$kbits. And usually the E2E delay for each video frame is also 50ms [10, 11].

As in Section 3.5, we assume a fixed cell size of 500bits/cell, and we always pick M so that M cell-time equals 1ms.

In the following, we run 1000 trials for each type of switch settings: with per port capacity of 1Gbps, 10Gbps, and 100Gbps; and number of input ports (which is also the number of output ports) of 8, 16, and 32.

In each trial, we randomly add sensing/actuating or video flows to a switch (without exceeding port capacities); and the messages of each flow f are over-provisioned with VM-task (M, C) as described in Equation (4) of Section 3.5. For each flow set, we calculate its switch utilization demand, and check whether the flow set is schedulable using the M cell-time clock-period. Note that the switch utilization demand is calculated using each flow's original message period and message size, not the over-provisioned VM-task (M, C); and switch utilization equals the average utilization of all inputs of the switch (assume all inputs has the same capacity). Fig. 4 plots the schedulability ratio (i.e. probability) for given switch utilization demand.

We find that our real-time switch achieves good schedulability and switch utilization. When the switch utilization demand is below 70%, a flow set is empirically always schedulable in all settings. Particularly, for high-speed switches with per port capacity of 10Gbps and 100Gbps, the switch utilization can reach nearly 85% and 90% for all settings to provide a 100% schedulable ratio (empirically).

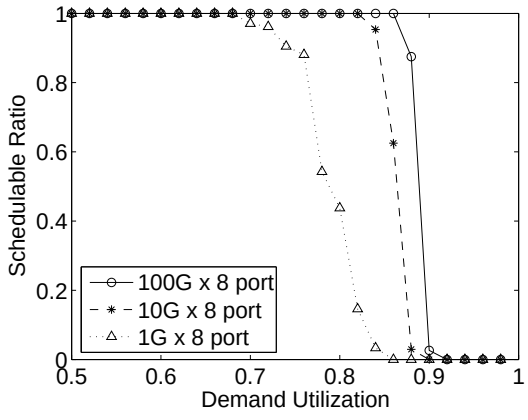
We also find that the M cell-time clock-period schedulability ratio improves as per-port capacity increases. Take Fig. 4 (a) for example: a switch utilization demand of 86% corresponds to a schedulability ratio of 0, 96%, and 100% when the per port capacity is 1Gbps, 10Gbps, and 100Gbps respectively.

On the other hand, the schedulability ratio deteriorates as the number of ports increases. For example, the 1Gbps curves of Fig. 4 (a), (b), and (c) shows that when the switch utilization demand is 80%, the schedulability is 43%, 22%, and 0 for 8 port, 16 port, and 32 port switches respectively. This is intuitive because more ports means more contention.

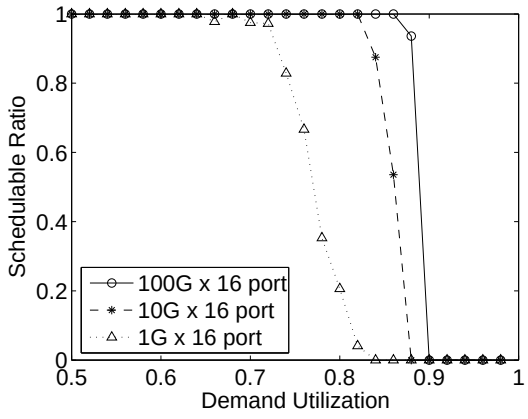
4.2 E2E Delay

The same simulation study described in Section 4.1 also provides E2E delay upper bound statistics. We compare them with those of iSLIP.

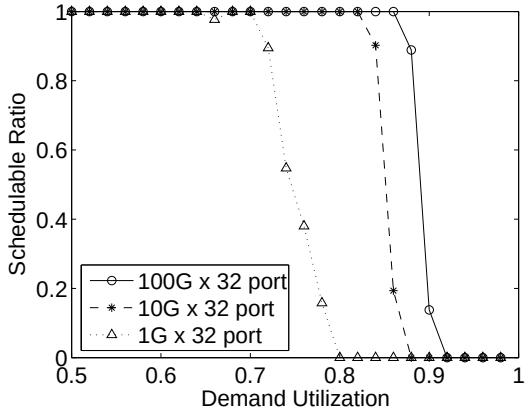
We assume the maximal hop count is 15. The E2E delay



(a)



(b)



(c)

Figure 4. Schedulability Ratio for Given Switch Utilization Demand using the Proposed Real-Time Switch and M Cell-Time Clock-Period

upper bound of our proposed real-time switch is given in Equation (5). A tight iSLIP E2E delay bound, however, is still an open problem. To make the comparison optimistic on the iSLIP side, we use the iSLIP *single hop* delay bound given in Equation (1) as its E2E delay bound.

The result statistics are shown in Fig. 5.

We see that using our proposed real-time switch, all E2E delays are within 50ms, which meets the demand of most industrial real-time traffic. Using iSLIP switches, however, most of the time even the single hop delay bound may exceed 100ms, 150ms, or even 200ms. Therefore, our proposed real-time switch provides better E2E delay guarantees.

4.3 Efficiency of LS Algorithm

Lastly, we evaluate the efficiency of LS algorithm described in Fig. 3.

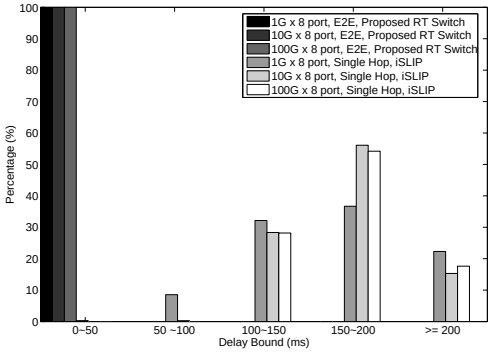
We know that Gonzalez and Sahni's POSS algorithm is optimal in the sense that it can schedule any feasible demand matrix. LS is a simpler, but sub-optimal algorithm. For any feasible demand matrix, POSS provides a schedulability ratio of 100%. We compare this with LS's schedulability ratio. We still try three different numbers of ports: 8, 16, and 32. For each number of ports, we try three different per port capacity: 1Gbps, 10Gbps, and 100Gbps. For each setting, we randomly generate 1000 feasible demand matrices, and check whether they are schedulable using LS algorithm. The results are plotted in Fig. 6.

We find that LS schedulability is sensitive to number of ports. As shown in Fig. 6 (a), (b), and (c), as number of ports increases from 8, to 16, and to 32, the LS-algorithm can schedule more than half, about half, and less than half of the randomly generated feasible matrices. This is intuitive because more number of ports means a demand matrix has more colors to conflict with each other in each column.

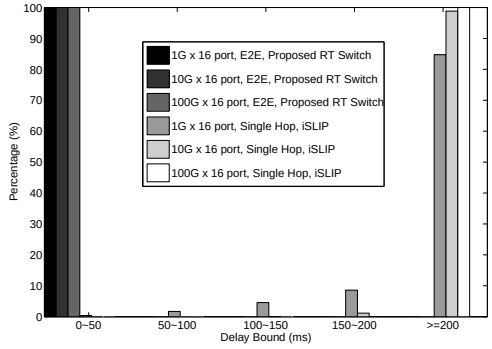
We also see that LS schedulability is not sensitive to per port capacity: in all of Fig. 6 (a), (b), and (c), different per port capacity of 1Gbps, 10Gbps, and 100Gbps result in similar curves. This is probably because the number of colors that can conflict is fixed, given the number of ports is fixed.

5 Related Work

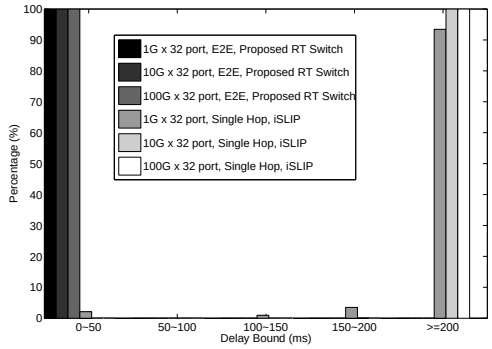
Network infrastructure for hard real-time communication has typically been restricted to prioritization in routers. The number of priority levels, however, is about 4 to 8 in conventional Internet routers, and this is insufficient for hard real-time guarantees. Additionally, many router designs for real-time systems have required significant changes when compared to commercially-available routers for Internet traffic. The desire to use existing solutions, or



(a)

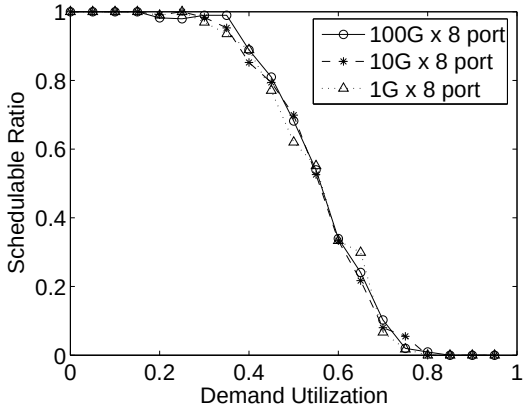


(b)

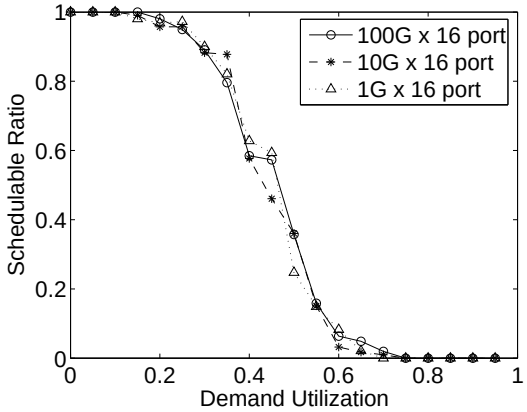


(c)

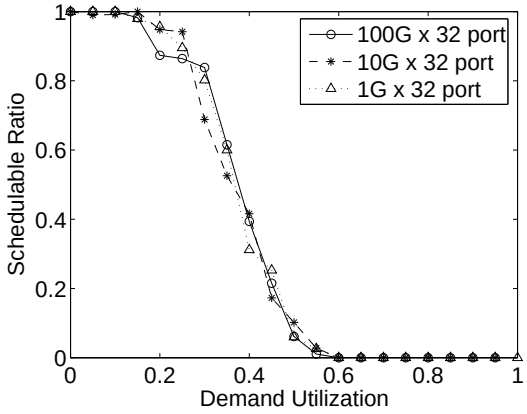
Figure 5. E2E Delay Comparison



(a)



(b)



(c)

Figure 6. LS Schedulability Ratio for Given Demand Matrix Utilization

solutions with minimal hardware changes, has been a dominant interest for industrial networks from the viewpoint of purchasing and maintenance costs.

Prioritized bus and ring networks have been used in small real-time systems [25, 28, 14] but they are not designed for high-speed network backbones, such as those of WANs. Rexford, Hall and Shin [26] propose a router for real-time communication but it was designed to support deadline-based scheduling, which imposes significant hardware changes. Additionally, their router is not designed for high-speed network backbones either. Similarly, Venkatramani and Chiueh proposed a real-time switch for Ethernets [34], which is neither designed for high-speed network backbones.

While there has been some effort, such as by Rexford, Hall and Shin, to design new routers for real-time systems, considerable effort has been devoted to analyzing the performance of high-speed switches and routers and obtaining delay bounds [30, 29]. The scheduling of crossbar switches reduces to a matching on a graph, and fast algorithms for obtaining a matching have also been studied [7]. These results use stochastic traffic patterns and provide asymptotic performance bounds that are not sufficient for industrial systems that require greater predictability.

Some related work concerns the use of COTS routers for real-time systems using approximate bounds and designing networks of switches to meet end-to-end deadlines [13]. The work presented in this article complements such work; better router architectures result in reduced message delays, which in turn reduces the cost of networks that can guarantee end-to-end requirements.

There are also efforts on emulating output queueing using input queueing or combined input-output queueing [4, 18]. However, to achieve the same hardware utilization efficiency as that of conventional input-queueing/VOQ crossbar switches is still an open problem.

The work presented in this article provides a mechanism for guaranteeing a task a certain amount of communication slots in a fixed time interval. The router design we have articulated is a building block for obtaining end-to-end delay bounds, and for enabling hierarchical scheduling policies and associated analysis [31, 19, 17, 8].

6 Conclusion

The convergence of computer and physical world is the theme for next generation networking research. This trend calls for real-time industrial network infrastructure, which needs high-speed real-time WAN to serve as its backbone. However, nowadays commercially available high-speed WAN switches (routers) are designed for best-effort Internet traffic. A real-time switch design for the aforementioned networks is missing.

In this article, we propose a real-time switch design on the most widely adopted crossbar switch architecture. The proposed switch can be implemented by making minimal modification, or even simplification, to the well-known iSLIP crossbar switch scheme. This benefits switch manufacturers since iSLIP is already widely implemented in commercial products, and the minor modifications can be easily incorporated into the manufacturing process.

Our real-time switch serves periodic and aperiodic traffic with real-time virtual machine tasks, which simplifies analysis, provides isolation, and facilitates future hierarchical scheduling and flow aggregation. Taking advantage that most industrial real-time network flows rarely change, the switch only needs to be configured to a real-time schedule at startup-time (aperiodic flows, which may change more frequently, are encapsulated by their real-time virtual machine tasks), and a polynomial time algorithm is found to schedule any feasible flow set. During runtime, our real-time switch incurs only $O(1)$ computation, which fits the need of high-speed networking.

Simulation results show that, for typical industrial real-time network traffic, our switch can achieve high utilization and guarantee small end-to-end delays.

We believe that it is essential to capture the true workload characteristics of applications, such as the predictability of network traffic in industrial control applications, to design efficient infrastructure for these applications. Further, changes in workload, which are infrequent and involve planned outages, can be accommodated via simple reconfiguration. As future work, we will extend our switch design to support run-time adaptation, hierarchical scheduling, and flow aggregation. We are also interested in better analyses for end-to-end delay bounds, and resource optimization issues.

7 Acknowledgement

This work is supported in part by NSF CCR 03-25716, NSF CNS 06-49885 SGER, by ONR N00014-05-0739, and by a grant from Lockheed Martin and a grant from Rockwell Collins. Sathish Gopalakrishnan and Xue Liu are supported by NSERC Discovery Grants. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] Working group summary: Critical physical infrastructure. *NSF Cyber-Physical Systems Workshop*, October 2006.

[2] Working group summary: Scientific foundations and education. *NSF Cyber-Physical Systems Workshop*, October 2006.

[3] J. C. R. Bennett et al. WF^2Q: Worst-case fair weighted fair queueing. *Proc. of INFOCOM'96*, pages 120–128, 1996.

[4] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching ouput queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, June 1999.

[5] R. Davis and A. Burns. Hierarchical fixed priority preemptive scheduling. *Proc. of IEEE RTSS'05*, 2005.

[6] R. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. *Proc. of IEEE RTSS'06*, 2006.

[7] S. Deb, D. Shah, and S. Shakkottai. Fast matching algorithms for repetitive optimization: an application to switch scheduling. In *Proceedings of the Conference on Information, Sciences and Systems*, 2006.

[8] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. *Proc. of IEEE RTSS'97*, 1997.

[9] I. Elhanany, M. Kahane, and D. Sadot. Packet scheduling in next-generation multiterabit networks. *IEEE Computer*, 34(4):104–106, Apr. 2001.

[10] B. Fisher et al. Seeing, hearing, and touching: Putting it all together. *SIGGRAPH'04 Course*, 2004.

[11] M. Glencross et al. Exploiting perception in high-fidelity virtual environments. *SIGGRAPH'06 Course*, 2006.

[12] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4):665–679, Oct. 1976.

[13] S. Gopalakrishnan, M. Caccamo, and L. Sha. Switch scheduling and network design for real-time systems. In *Proc. of IEEE Real-Time and Embedded Technology and Applications (RTAS)*, Apr. 2006.

[14] S. Gopalakrishnan, L. Sha, and M. Caccamo. Hard real-time communication in bus-based networks. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 2004.

[15] R. Gupta and K. G. Shin. Working group summary: Infrastructure and building blocks. *NSF Cyber-Physical Systems Workshop*, October 2006.

[16] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division swtich. *IEEE Transactions on Communications*, 35:1347–1356, Dec. 1987.

[17] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. *Proc. of IEEE RTSS'99*, 1999.

[18] H.-I. Lee and S.-W. Seo. Matching output queueing with a multiple input/output-queued switch. *IEEE/ACM Trans. on Networking*, 14(1):121–132, February 2006.

[19] G. Lipari and E. Bini. Resource partitioning among real-time applications. *Proc. of ECRTS*, 2003.

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

[21] N. McKeown. The *i*SLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2), Apr. 1999.

[22] N. W. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, EECS Dept., University of California at Berkeley, 1995.

[23] A. K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Network*. PhD thesis, EECS Dept., M.I.T., Feb. 1992.

[24] L. L. Peterson and B. S. Davie. *Computer Networks: A System Approach*. Margan Kaufmann, second edition, 2000.

[25] R. S. Raji. Smart networks for control. *IEEE Spectrum*, 31:49–55, June 1994.

[26] J. Rexford, J. Hall, and K. G. Shin. A router architecture for real-time communication in multicomputer networks. *IEEE Transactions on Computers*, 47(10):1088–1101, Oct. 1998.

[27] L. Sha and A. Agrawala. Real time and embedded (RTE) GENI. *ACM SIGBED Review*, 3(3), July 2006.

[28] L. Sha, R. Rajkumar, and J. P. Lehoczky. Real-time scheduling support in Futurebus+. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 331–340, Dec. 1990.

[29] D. Shah, P. Giaccone, and E. Leonardi. Throughput region of finite-buffered networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(2), Feb. 2007.

[30] D. Shah, P. Giaccone, E. Leonardi, and B. Prabhakar. Delay bounds for combined input and output switches with low speedups. *Performance Evaluation*, 55(1-2), 2004.

[31] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of the 24th IEEE International Real-Time Systems Symposium (RTSS 2003)*, Dec. 2003.

[32] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proc. of SIGCOMM*, pages 231–242, 1995.

[33] M. Spong and K. Nahrstedt. Working group summary breakout session on tele-interaction. *NSF Cyber-Physical Systems Workshop*, October 2006.

[34] C. Venkatramani and T. Chiueh. Design and implementation of a real-time switch for segmented Ethernets. In *Proceedings of the International Conference on Network Protocols*, October 1997.

[35] Q. Wang, X. Liu, W. Chen, L. Sha, and M. Caccamo. Building robust wireless LAN for industrial control with the DSSS-CDMA cell phone network paradigm. *IEEE Transactions on Mobile Computing*, 6(6):706–719, June 2007.