Adapting a Main-Stream Internet Switch Architecture for Multi-Hop Real-Time Industrial Networks

Qixin Wang*, Sathish Gopalakrishnan†
* Department of Computing, The Hong Kong Polytechnic University
† Department of Electrical and Computer Engineering, University of British Columbia
Email: csqwang@comp.polyu.edu.hk, sathish@ece.ubc.ca

***Abstract*—As real-time industrial control systems scale up, single real-time local area network (LAN) is no longer sufficient; instead, we need real-time switches to merge many real-time LANs into real-time wide area networks (WANs). However, nowadays commercially-off-the-shelf WAN switches are designed for best-effort Internet traffic rather than real-time traffic. To address this problem, we propose a real-time crossbar switch design that minimally modifies, and even simplifies the de facto industrial standard switch design of iSLIP. Specifically, we change the iSLIP request-grant-accept negotiation to deterministic grant. The switch runs periodically with an M cell-time clock-period. Every input port runs per-flow queueing, and every output port deterministically grants input port per-flow queues according to its own M cell-time clock-period schedule. The schedules are created offline. We prove that the global scheduling can be reduced to a preemptive open shop scheduling problem; as long as every input/output needs to send/fetch no more than M cells per M cell-time clock-period, all outputs schedules do not conflict; and the scheduling algorithm takes $O(N^4)$ time (N is the number of input/output ports). Such design serves real-time periodic/aperiodic traffic in a TDMA fashion. This simplifies analysis, provides isolation, and results in a close-form end-to-end delay bound. We implemented the proposed real-time switch using Xilinx FPGAs, and built a distributed control test bed upon the switched networks. Using the test bed, we carried out experiments to compare the implemented real-time switches and iSLIP switches. The results prove the necessity of using real-time switches for real-time industrial control.**

***Index Terms*—real-time, switch, industrial control, Cyber-Physical Systems**

I. Introduction

It is widely believed that Cyber-Physical Systems (CPS), the effort to converge computers with the physical world, is a theme for future computer science [1], [2], [3]. A representative application of CPS is distributed real-time industrial control, where distributed sensing, actuating, and control nodes are interconnected via an underlying real-time network. The traffic in such networks mainly includes periodic sensing/actuating flows and periodic video flows with constant *end-to-end* (E2E) delay bound requirements. For example, a typical sensing/actuating flow may generate a message of 1kbit every 10ms, and each message must be delivered to the receiver end within 50ms. In the following, we call such periodic flows with constant E2E delay bound requirements *real-time flows*; and focus on how to design switches to support such flows (i.e., the so called *real-time switches*). The following further explains our motivation.

Many works [3], [4], [5] have pointed out that as distributed real-time industrial control systems scale up, single real-time local area network (LAN) is no longer sufficient to integrate their distributed subsystems; instead, we need real-time switches to merge the many real-time LANs into real-time *wide area networks* (WAN). For example, nowadays airplane control involves hundreds of processors and peripherals, which already exceeds the capacity of a single LAN. This forces the avionics industry to push forward the *avionics full-duplex switched Ethernet* (AFDX) [6] and the Infiniband switched system area network architecture [7]. Same thing is happening to advanced manufacturing [8], factory fieldbus [9], [10], [11], advanced medical equipment systems [12], [13], smart power grid [14], vehicular electronics [15] etc. Even for wireless industrial control, real-time switches are needed to build the multi-hop wired backbones to connect wireless base stations: as Alves et al. [16], Willig et al. [17], Pellegrini et al. [18], and Wang et al. [4] pointed out, wired backbone converging multiple (centralized) wireless LANs might be the (most) promising architecture for wireless industrial control.

However, the majority of nowadays commercially available switches are tailored for best-effort Internet traffic rather than real-time systems. Specifically, there are two main approaches to building a switch: output queueing and input queueing.

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 is immediately routed to the queue at its destined output. Due to its simplicity, most QoS scheduling algorithms, such as WFQ [19], WF^{2}Q [20], Deficit Round-Robin [21] etc., assume output queueing [22].

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 the total capacity of all inputs. Suppose a switch has N inputs, each with a capacity of C, then the data bus connected to each output must provide a capacity of $N \times C$. We call this N *speed-up problem*. The N speed-up problem makes output queueing undesirable for high-speed switches or switches with large number of ports (N).

In contrast to output queueing, input queueing buffers packets in queues at the inputs. This avoids the N speed-up problem, 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% [23].

The widely adopted solution to the HOL problem is *virtual output queueing* (VOQ), where each input maintains N queues, one exclusively for each output (hence called the "*virtual output queue*" for that 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. According to our survey on switches in the market, of all these schemes, iSLIP [24], [25], [26] has become the de facto standard among switch manufacturers. However, though iSLIP efficiently utilizes the switch hardware and is simple to implement, it does not guarantee real-time. In fact, real-time high-performance switch design is still an open problem [27].

To address this problem, we propose a real-time switch design that minimally modifies, or even simplifies iSLIP. This design benefits switch manufacturers because iSLIP is already widely implemented in commercial products, and our proposed minor modifications/simplifications can be easily incorporated into the current manufacturing process. Our approach allows a switch to serve each link l for C_l units of time every M units of time. It can easily support flow isolation, and hence facilitates future extension to hierarchical scheduling [28], [29], [30], [31], [32], [33], [34], [35], [22], [36], [37].

In the following, Section II describes the iSLIP scheme; Section III proposes our switch design for industrial real-time communications; Section IV evaluates our design; Section V discusses related work; and Section VI concludes the paper.

II. Crossbar Switches and iSLIP

To support input queueing or VOQ, most high-performance switches use a crossbar fabric to connect inputs and outputs [38] (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 connected or disconnected 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.

iSLIP [24], [25] is a widely implemented 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 $O_1 \sim O_N$ (or an "$N \times N$ switch" in the following discussion). 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.

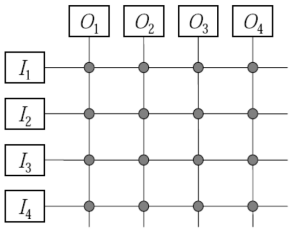


Fig. 1. Crossbar fabric that connects inputs with outputs.

With the above data structures, the basic iSLIP runs the following steps [24]:

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 grants 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 size of the matching.

The original iSLIP mechanism [24], [25] 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 [25], 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" [27]. 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_j, then the single hop delay bound is at least 102400 times that of a packet transmission time.

III. A Real-Time Switch Design

To support real-time communication, 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) [28], [39], [33], [34], [22], [36], [37]. One simple and widely implemented form is clock-driven scheduling [22], 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, iSLIP's request-grant-accept negotiation between inputs and outputs is for non-deterministic Internet traffic, which changes frequently. *If the traffic rarely changes and is periodic, as that of flows in real-time industrial control 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.

A. 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 [19] and WF^{2}Q [20], do), which may affect performance.

B. Traffic demand

All traffic demand in our real-time switch is abstracted by the clock-driven scheduling of VM-tasks (see Section III-E 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

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

Constraint 2: Feasible output utilization

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

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

C. 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 III-D).

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_i 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.

D. 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 [40].

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 Gonzalez and Sahni [40]. 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 [40], 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 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 gth 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.

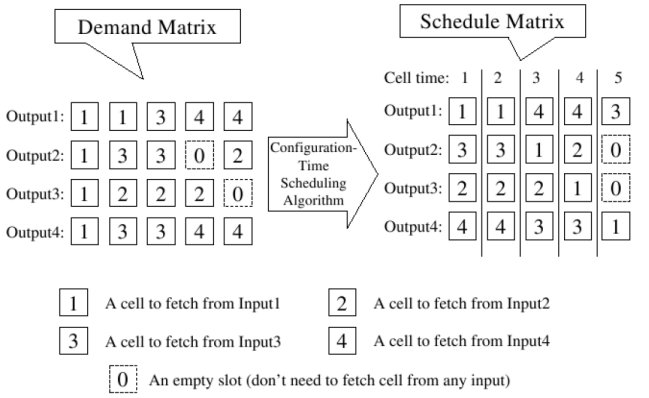


Fig. 2. An example illustrates the relationship between Demand Matrix, Configuration-Time Scheduling Algorithm, and Schedule Matrix, where number of ports $N = 4$, and a clock-period is $M = 5$ cell-time.

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.

Fig. 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 the 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 $O(N^2 \log N + NM + N^2 M) = O(N^2 M)$.

E. 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 dominating traffic is periodic, such as sensing, actuating, and video monitoring. Aperiodic traffic can be served by periodic VM-tasks [22]. As a result, we shall assume that all traffic is 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 uniform 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 that a real-time flow f needs to send, at the least, a message of E cells every T cell-time, denoted as $f = (T, E)$. Note that E and T may be real numbers instead of integers. Then we over provision f with VM-task $\tau_f = (M, C)$, where

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

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

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, Equation (4) assumes $T > M$, since when 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

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

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 that 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.

IV. Evaluation

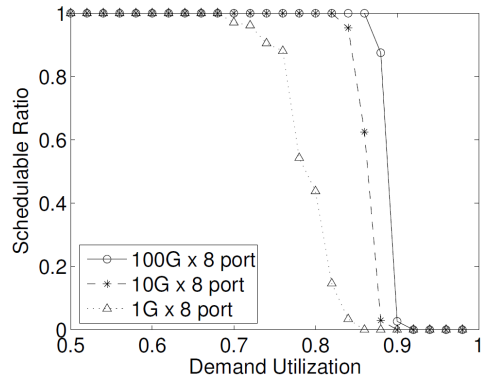
A. Efficiency of M Cell-Time Clock-Period

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 real-time industrial control traffic.

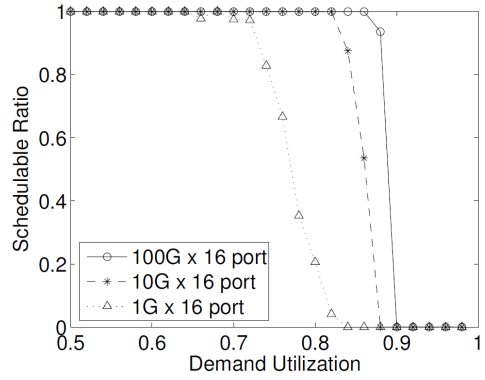
There are two types of real-time traffic in real-time industrial control: real-time sensing/actuating traffic and real-time video traffic. Real-time sensing/actuating traffic involves low data-throughput. A typical sensing/actuating flow generates a $1 \sim 5$kbit message every 10(ms). The maximal allowed E2E delay is usually 50ms [41], [42]. 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 [41], [42]. As in Section III-E, 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.

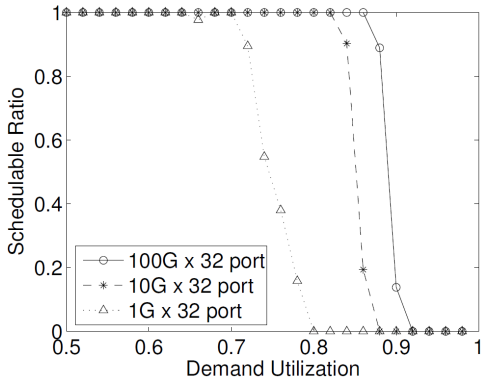
In each trial, we randomly add sensing/actuating or video flows to a switch (without exceeding port capacities); and the messages of each flow f are over-provisioned with VM-task



(a)



(b)



(c)

Fig. 4. Schedulability ratio for given switch utilization demand using the proposed real-time switch and M cell-time clock-period.

(M, C) as described in Equation (4) of Section III-E. 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.

B. E2E Delay

Besides utilization, real-time application users are more concerned with the switched network's E2E delay bound. The majority of industrial real-time applications are mission or safety critical. For such applications, users would not and should not choose a switched network whose E2E delay bound is unknown. This implicitly disqualifies the use of iSLIP switched networks, whose E2E delay bound is still an open problem.

Now the remaining question is, are the E2E delay bound provided by our real-time switch good enough? The following gives the answer.

We run the same simulation described in Section IV-A to evaluate the E2E delay upper bound statistics. We assume that the maximal hop count is 15. The E2E delay upper bound of our proposed real-time switch is given in Equation (5). The simulation result statistics are summarized in Fig. 5.

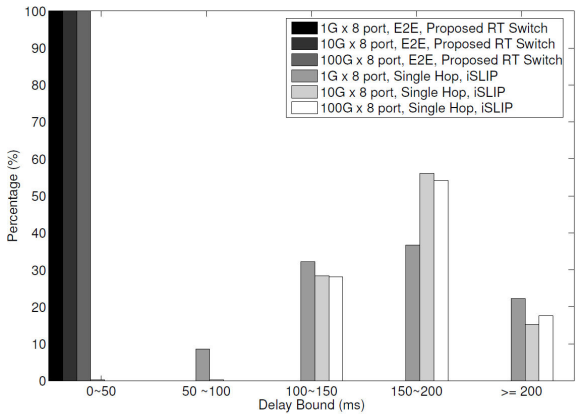
We see that using our proposed real-time switch, all E2E delays are within 50ms, which meets the demand of most industrial real-time traffic[2].

C. Efficiency of LS Algorithm

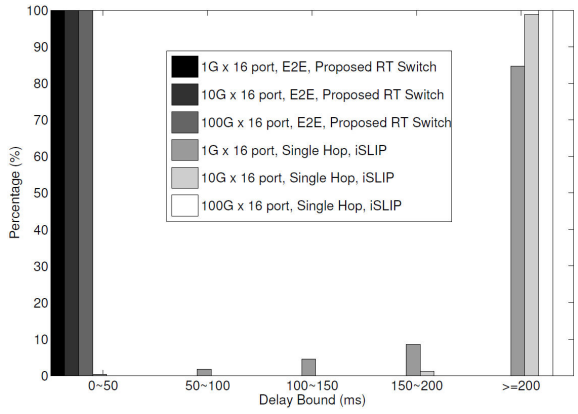
We also evaluate the efficiency of LS algorithm described in Fig. 3.

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 use the same traffic generator (uniform traffic random distribution,

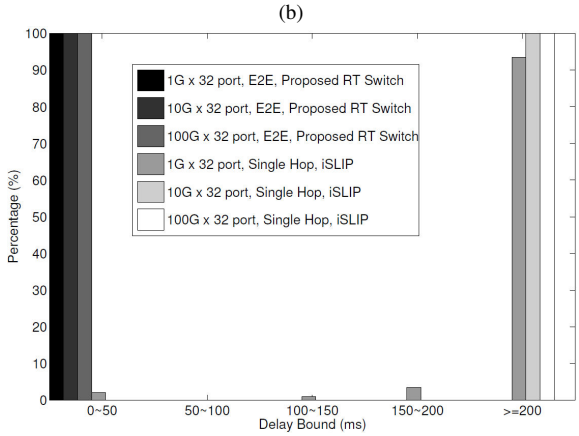
[2]To provide more information, Fig. 5 also plots the corresponding *single hop* delay bound statistics if we use iSLIP switches instead (see Equation (1)). According to [27]'s analysis, the single hop delay bound is tight when all queues in the iSLIP switch are backlogged. Meanwhile, according to [25]'s analysis, when traffic is uniformly distributed and the iSLIP switch is heavily loaded (e.g., over 50%), the scenario that all queues are backlogged happens very often. Therefore, under our simulation set up (uniform traffic distribution, and heavily loaded), the case that all queues are backlogged happens. When such a case happens, the iSLIP single hop delay bound given in Equation (1) becomes tight, and hence becomes a lower bound for iSLIP E2E delay.



(a)

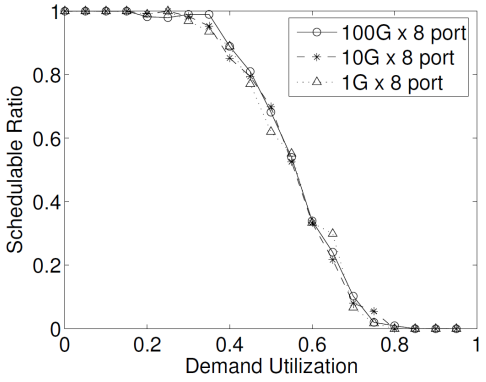


(b)

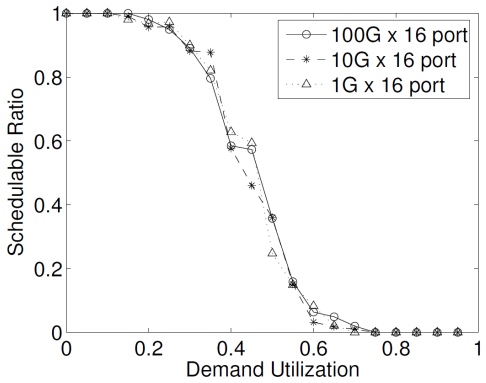


(c)

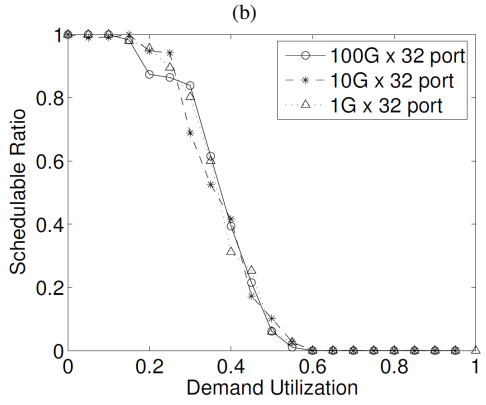
Fig. 5. E2E Delay Comparison. The iSLIP single hop delay bound statistics are also provided, please see footnote 2 for their meanings.



(a)



(b)



(c)

Fig. 6. LS Schedulability Ratio for Given Demand Matrix Utilization.

same sensing/actuating and video traffic pattern) used in Section IV-A to create 1000 feasible demand matrices, and check whether they are schedulable using the LS algorithm. The results are plotted in Fig. 6.

We find that LS schedulability is sensitive to the number of ports. As shown in Fig. 6 (a), (b), and (c), as the 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.

D. Experiment

We implemented the iSLIP switch and our real-time switch on Xilinx ML401 FPGAs [43] and built a test bed as shown in Fig. 7. The testbed uses a switched network to connect a Control Node with a Quanser 3DOF Helicopter [44] (see Fig. 8). The helicopter periodically (every 10msec) sends the control node its angular positions along the three movement axes (see Fig. 3-2): travel (λ), elevation (ε), and pitch (p) . The control node periodically (every 10msec) feeds back the control command.

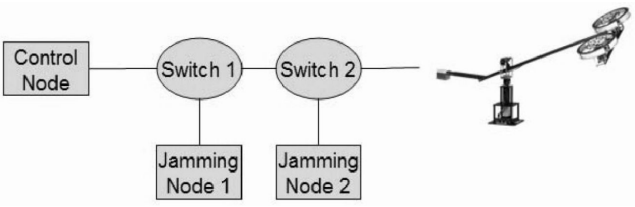


Fig. 7. Test Bed Layout (helicopter picture from [44]).

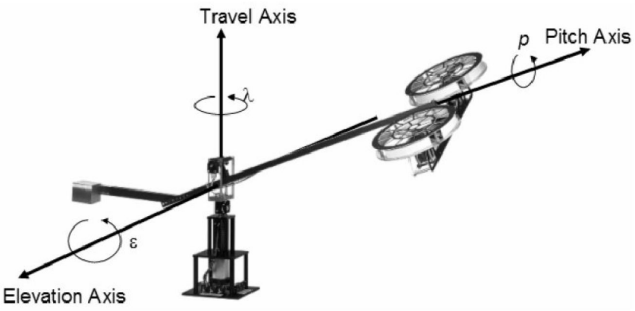


Fig. 8. Quanser 3 DOF Helicopter (picture from [44]). By applying voltages, the two propellers (the two blue circles in the right side of the figure) can turn/position the helicopter along three rotation axes: travel (λ), elevation (ε), and pitch (p).

According to Fig. 7, two additional jamming nodes are connected to Switch 1 and Switch 2 respectively. The jamming nodes inject jamming traffic to interfere the real-time flows between the control node and the helicopter. In our experiments, jamming traffic consists of continuous cells sent toward the helicopter.

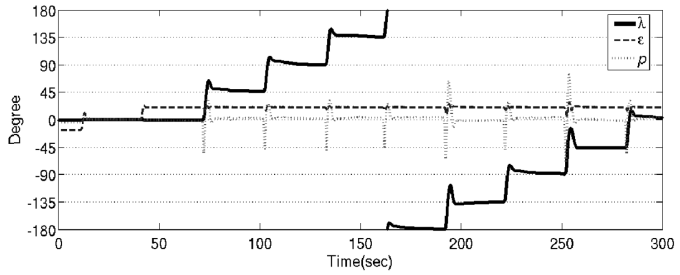
1) Demo: We carried out four trials (corresponding demo videos are also available on YouTube [45]) to demo the effectiveness of real-time switch. Each trial tries to fly the helicopter around its travel axis for one full circle, stopping at $\lambda = 0°, 45°, 90°, 135°, 180°, -135°, -90°, -45°, 0°$, while maintaining elevation angle ε around $0°$.

The first and second trial have no jamming traffic. Fig. 9 (a) and (b) show the traces of the helicopter during these two trials respectively. According to the figures, both real-time switch and iSLIP switch work fine.

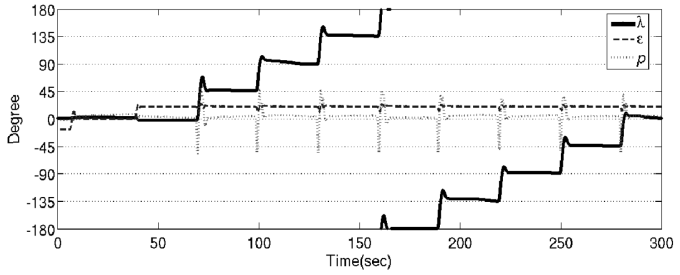
The third and fourth trial have jamming traffic. Fig. 9 (c) and (d) show the traces of the helicopter during these two trials respectively. In both trials, both jamming nodes in the test bed are turned on. We see the jamming traffic barely affects the real-time switch network; but for the iSLIP switch network, the helicopter cannot takeoff (elevation angle ε remains negative) and later loses control (an abrupt shoot up of pitch angle p), and we have to stop the system to prevent damaging the helicopter hardware.

2) Quantitative Comparison: We run more trials to carry out quantitative comparisons. In each trial, we first fly the helicopter to a reference position in the air; when helicopter stabilizes[3], we turn on jamming traffic and observe the helicopter for at least 10 more seconds; and then we stop.

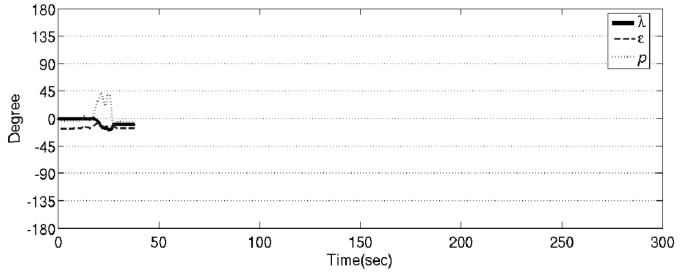
[3]Empirically, 10 seconds after taking off is way enough for the helicopter to stabilize around our reference position.



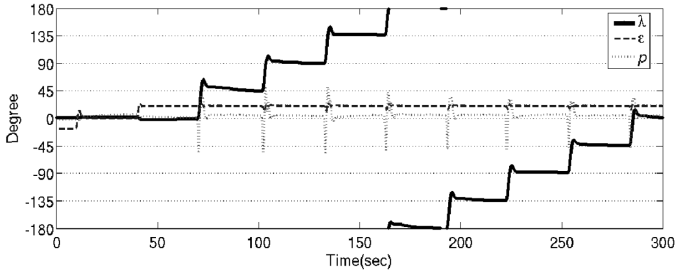
(a) Using iSLIP Switch, without Jamming



(b) Using Real-Time Switch, without Jamming



(c) Using iSLIP Switch, with Jamming

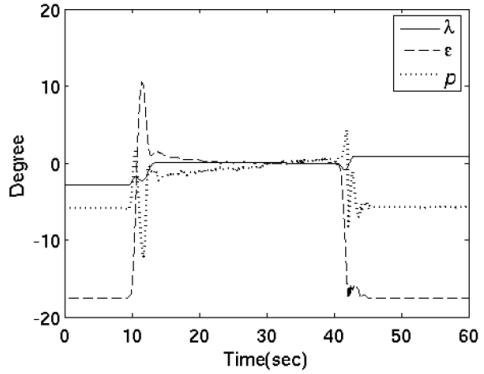


(d) Using Real-Time Switch, with Jamming

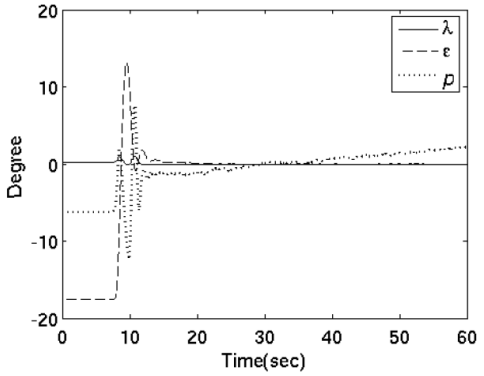
Fig. 9. Trace of Helicopter.

Ten trials are carried out using the iSLIP switch network and the real-time switch network respectively; and then the statistics of the twenty traces are compared.

Fig. 10 (a) and (b) show the helicopter traces of two example trials for iSLIP switch networks and real-time switch networks respectively. In the trial of Fig. 10 (a), the jamming traffic is turned on at the time instance of 38.0sec, when the helicopter has been stabilized for at least 10sec in the air. But since iSLIP cannot maintain real-time flow under jamming, the helicopter falls immediately once the jamming traffic is turned on. The same thing happens to all other nine trials using iSLIP switch networks. In the trial of Fig. 10 (b), the jamming traffic is turned on at the time instance of 36.5sec. Since real-time switch can isolate real-time flow from jamming traffic, the helicopter is not affected. The same thing happens to all other nine trials using real-time switch networks.



(a) A trial using the iSLIP switch network, jamming starts at 38.0sec.



(b) A trial using the real-time switch network, jamming starts at 36.5sec.

Fig. 10. Example Trials/Traces.

TABLE I
STATISTICS OF T_{fall}(SEC): TIME TO FALL AFTER JAMMING STARTS

	min	mean	max	std
iSLIP Switch	1.2	3.1	4.0	0.8
Real-Time Switch	∞	∞	∞	∞

The above fact is quantitatively described by Fig. 11 and Table I.

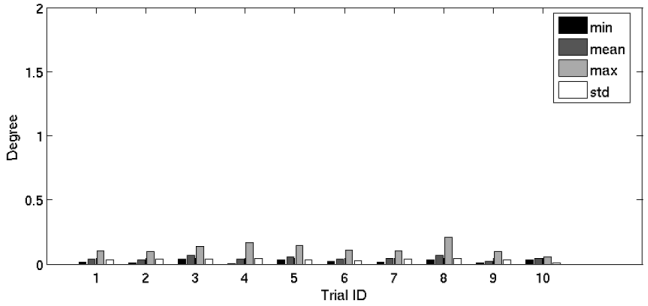
Fig. 11 (a) and (b) compare the statistics of helicopter traces 10sec before and 10sec after the jamming starts in the iSLIP switch network; while Fig. 11 (c) and (d) compare those of real-time switch network. The metric we compare is the absolute deviation d_ε of elevation angle ε, defined as

$$d_\varepsilon \stackrel{def}{=} |\varepsilon - \varepsilon_{ref}|,$$

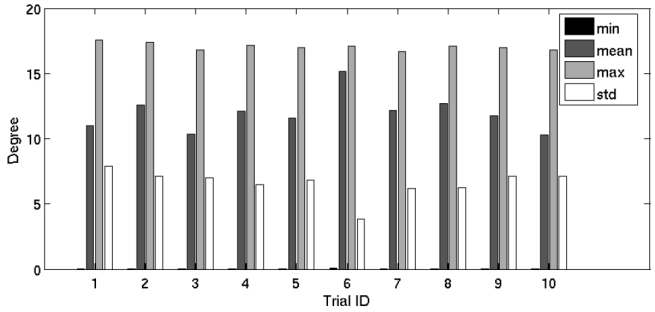
where ε is the sampled elevation angle and ε_{ref} is the elevation angle for the reference position. The sampling rate is 100Hz.

We choose d_ε because the elevation angle indicates whether the helicopter remains in the air (it is negative when the helicopter stays/hits the ground); and staying in the air instead of falling is the most basic and safety critical requirement for helicopter control.

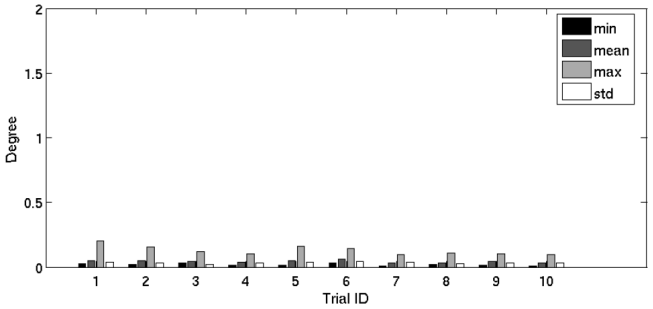
In all ten trials using iSLIP switch, the helicopter falls; while in all ten trials using real-time switch, the helicopter does not fall. Let T_{fall}(sec) indicate the time length between the start of jamming and the helicopter falls to the ground, Table I shows the statistics of T_{fall}.



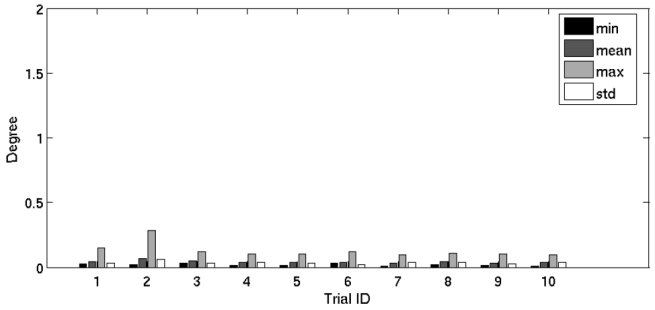
(a) iSLIP switch network: statistics of d_ε sampled during the 10sec period right before jamming starts.



(b) iSLIP switch network: statistics of d_ε sampled during the 10sec period right after jamming starts.



(c) Real-time switch network: statistics of d_ε sampled during the 10sec period right before jamming starts.



(d) Real-time switch network: statistics of d_ε sampled during the 10sec period right after jamming starts.

Fig. 11. Comparison of $d_\varepsilon \stackrel{def}{=} |\varepsilon - \varepsilon_{ref}|$ statistics in the iSLIP switch network and the real-time switch network.

V. Related Work

The main purpose of the paper is to pave way for the mainstream Internet switch vendors, who build iSLIP switches, to smoothly evolve/expand toward real-time industrial control and CPS. We believe such a smooth evolution path can attract more support for real-time industrial control and CPS from the Internet industry.

Besides iSLIP, the following gives more related work in the Internet switch industry.

Internet support for real-time communication has typically been restricted to prioritization in switches (also called "routers" if routing function is emphasized, or if parallel packet forwarding is not supported). The number of priority levels, however, is about 4 to 8 in conventional Internet switches, and this is insufficient for hard real-time guarantees. On the other hand, many switch designs for real-time systems require significant changes compared to commercially-available switches for Internet. But for most switch manufacturers, the desire to use existing solutions, or solutions with minimal changes, plays a key role in decision making due to cost and risk management considerations.

Prioritized bus and ring networks have been used in small real-time systems [46], [47], [48] but they are not designed for high-speed network backbones, such as those of WANs. Rexford, Hall and Shin [49] propose a switch for real-time communication but it was designed to support deadline-based scheduling, which imposes significant hardware changes. Additionally, their switch is not designed for high-speed network backbones either. Similarly, Venkatramani and Chiueh proposed a real-time switch for Ethernets [50], 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 switches for real-time systems, considerable effort has been devoted to analyzing the performance of high-speed switches and obtaining delay bounds [51], [52]. The scheduling of crossbar switches reduces to a matching on a graph, and fast algorithms for obtaining a matching have also been studied [53]. 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 switches for real-time systems using approximate bounds and designing networks of switches to meet end-to-end deadlines [27]. The work presented in this article complements such work; better switch 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 [54], [55], [56], [57], [58]. However, how to achieve the same hardware utilization efficiency as that of conventional input-queueing/VOQ crossbar switches is still an open problem. A variation of combined input-output queueing is combined input-crosspoint-output queueing [59], [60], [61], [62], where buffers are allocated in inputs, outputs, and the crossbar. However, such architecture has not yet been widely implemented by switch manufacturers.

Unlike the Internet switch industry, the industrial fieldbus industry [63] has been working on hard real-time support for a long time. Architectures like Profibus [10], Foundation Fieldbus [9], CAN bus [64], TTEthernet [65] etc. are already widely used. However, their support for hard real-time mainly focuses on local area networks; the support for hard real-time over multi-hop switched networks is not universal.

TTEthernet [65] is one fieldbus standard that supports hard real-time over multiple hops of switches. The core of TTEthernet is a global clock synchronization service installed on every participating node. With that service at hand, global time division multiple access control can be carried out to support hard real-time. However, TTEthernet is based on the assumption that the underlying multi-hop switched network has deterministic end-to-end delay bound. TTEthernet does not specify the detailed design of the switches. Therefore, our real-time switch can complement TTEthernet by providing a detailed switch design that matches its core assumption.

Profibus [10] is another fieldbus standard that supports hard real-time over multiple hops of switches; however, in that case, Profibus assumes that all nodes on the network exclusively use Profibus' specialized network stacks. Unlike our design, isolation of misbehaving jamming traffic, e.g. due to the use of non-Profinet network stacks, is not the focus of Profibus. Profibus neither concerns about how to plan a smooth evolution path for Internet switches to support multi-hop real-time.

Including Profibus, many fieldbuses' detailed designs for multi-hop real-time networking are quite proprietary. To break this limit, Dopatka and Wismuller [66] proposed a brand new open fieldbus architecture to support multi-hop real-time networking. Unlike Dopatka and Wismuller's work, our focus is to find a smooth evolution path for Internet switch vendors, particularly the large population of iSLIP switch vendors, to support multi-hop real-time networking. Hopefully, such a evolution roadmap can foster the convergence of real-time fieldbus networking and Internet, enabling more real-time applications (such as tele-presence [67]), and expanding their scale from factory-wide to global.

One of the most recent works on real-time industrial fieldbus is Santos et al.'s design of a synthesizable Ethernet switch with enhanced real-time features [68]. This design is based on shared bus switch architecture. However, again the focus is not for finding a smooth evolution path for the many iSLIP switch vendors toward multi-hop real-time networking, as iSLIP is a crossbar switch architecture intead of shared bus.

It is also brought to our attention recently that Leung and Yum proposed a TDM-based multibus packet switch [69] similar to our design. Compared to [69]'s design, our design extends fixed capacity allocation to arbitary capacity allocation, gives corresponding scheduling algorithms and schedulability test formulae, derives end-to-end delay bounds, and points out a smooth evolution path for iSLIP architecture.

The conference version of this paper is published in [70].

VI. 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 based on the most widely adopted crossbar switch architecture. The proposed switch can be implemented by making minimal modifications, or even simplifications, 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 incorportated into the manufacturing process.

Our real-time switch serves periodic and aperiodic traffic with real-time virtual machine tasks, which simplifies analysis, and provides isolation. Taking advantage of the fact 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 also implemented the proposed real-time switch using Xilinx FPGAs, and built a distributed control test bed upon the switched networks. Using the test bed, we carried out experiments to compare the implemented real-time switches and iSLIP switches. The results prove the necessity of using real-time switches for real-time industrial control.

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 in resource optimization issues.

VII. 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. Qixin Wang is supported by Dept. of Computing, The Hong Kong Polytechnic Univ. start up fund. Sathish Gopalakrishnan is 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. The authors thank anonymous reviewers for their advice on improving this paper.

We thank Prof. Lui Sha of UIUC and Prof. Xue Liu of Univ. of Nebraska Lincoln for their advice for our research. We also thank Olugbemiga Adekunle, Emiliano Betti, Bach Duy Bui, Patrick Drinkwine, Joseph F. Girotti, Jason Kemper, Rodolfo Pellizzoni for their contributions in implementing the test bed. We thank Mr. Yufei Wang and anonymous reviewers for their valueable comments.

References

[1] W. Wolf, "Cyber-physical systems," *IEEE Computer*, vol. 42, no. 3, Mar. 2009.

[2] E. Lee, "Cyber-physical systems design challenges," *UC Berkeley Tech. Report UCB/EECS-2008-8*, 2008.

[3] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," *Proc. of IEEE Intl' Conf. on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2008.

[4] 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*, vol. 6, no. 6, pp. 706–719, Jun. 2007.

[5] L. Sha and A. Agrawala, "Real time and embedded (RTE) GENI," *ACM SIGBED Review*, vol. 3, no. 3, Jul. 2006.

[6] *664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. ARINC Standard, http://www.arinc.com.

[7] *InfiniBand Trade Association*. http://www.infinibandta.org.

[8] *Factory Automation Systems Integrator ATS*. http://www.atsautomation.com.

[9] *Fieldbus Foundation*. http://www.fieldbus.org.

[10] *Profibus & Profinet International*. http://www.profibus.com.

[11] *1588-2008 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2008.

[12] *MD PnP Program*. http://mdpnp.org.

[13] N. C. O. for Networking, I. T. Research, and Development, *High-Confidence Medical Devices: Cyber-Physical Systems for 21st Century Health Care – A Research and Development Needs Report*, Feb. 2009.

[14] G. N. S. Prasanna, A. Lakshmi, S. Sumanth, V. Simha, J. Bapat, and G. Koomullil, "Data communication over the smart grid," *IEEE Int'l Symp. on Power Line Communications and Its Applications*, pp. 273–279, 2009.

[15] *FlexRay - The communication system for advanced automotive control applications*. http://www.flexray.com.

[16] M. Alves, E. Tovar, F. Vasques, G. Hammer, and K. Rother, "Real-time communications over hybrid wired/wireless PROFIBUS-based networks," *Proc. of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02)*, 2002.

[17] A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE (Invited Paper)*, vol. 93, no. 6, pp. 1130–1151, Jun. 2005.

[18] F. D. Pellegrini, D. Miorandi, S. Vitturi, and A. Zanella, "On the use of wireless networks at low level of factory automation systems," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 2, pp. 129–143, May 2006.

[19] A. K. Parekh, "A generalized processor sharing approach to flow control in integrated services network," Ph.D. dissertation, EECS Dept., M.I.T., Feb. 1992.

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

[21] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in *Proc. of SIGCOMM*, 1995, pp. 231–242.

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

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

[24] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. dissertation, EECS Dept., University of California at Berkeley, 1995.

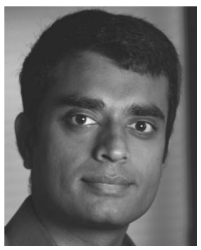
[25] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, Apr. 1999.

[26] I. Elhanany, M. Kahane, and D. Sadot, "Packet scheduling in next-generation multiterabit networks," *IEEE Computer*, vol. 34, no. 4, pp. 104–106, Apr. 2001.

[27] 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.

[28] I. Shin, M. Behnam, T. Nolte, and M. Nolin, "Synthesis of optimal interfaces for hierarchical scheduling with resources," *RTSS*, Dec. 2008.
[29] M. Bertogna, N. Fisher, and S. Baruah, "Static-priority scheduling and resource hold times," *WPDRTS*, 2007.
[30] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," *Proc. of IEEE RTSS'06*, 2006.
[31] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee, "Incremental schedulability analysis of hierarchical real-time components," *EMSOFT*, 2006.
[32] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," *EMSOFT*, 2004.
[33] 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.
[34] G. Lipari and E. Bini, "Resource partitioning among real-time applications," *Proc. of ECRTS*, 2003.
[35] A. Mock, X. Feng, and D. Chen, "Resrouce partition for real-time systems," *RTAS*, 2001.
[36] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," *Proc. of IEEE RTSS'99*, 1999.
[37] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," *Proc. of IEEE RTSS'97*, 1997.
[38] L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*, 2nd ed. Margan Kaufmann, 2000.
[39] R. Davis and A. Burns, "Hierarchical fixed priority preemptive scheduling," *Proc. of IEEE RTSS'05*, 2005.
[40] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the Association for Computing Machinery*, vol. 23, no. 4, pp. 665–679, Oct. 1976.
[41] B. Fisher *et al.*, "Seeing, hearing, and touching: Putting it all together," *SIGGRAPH'04 Course*, 2004.
[42] M. Glencross *et al.*, "Exploiting perception in high-fidelity virtual environments," *SIGGRAPH'06 Course*, 2006.
[43] *FPGA and CPLD Solutions from Xilinx, Inc.* http://www.xilinx.com.
[44] *Quanser Inc.* http://www.quanser.com.
[45] Q. Wang *et al.*, *E2E Real-Time Solution for Avionics and Control Demo2: Real-Time Switch vs. iSLIP Switch*. http://www.youtube.com/watch?v=hzKLutGgPXo.
[46] R. S. Raji, "Smart networks for control," *IEEE Spectrum*, vol. 31, pp. 49–55, Jun. 1994.
[47] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Real-time scheduling support in Futurebus+," in *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1990, pp. 331–340.
[48] 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.
[49] 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.
[50] 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.
[51] 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, 2004.
[52] D. Shah, P. Giaccone, and E. Leonardi, "Throughput region of finite-buffered networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 2, Feb. 2007.
[53] 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.
[54] H.-I. Lee and S.-W. Seo, "Matching output queueing with a multiple input/output-queued switch," *IEEE/ACM Trans. on Networking*, vol. 14, no. 1, pp. 121–132, February 2006.
[55] D. Pan and Y. Yang, "Pipelined two step iterative matching algorithms for CIOQ crossbar switches," *Proc. ACM/IEEE Symp. Architectures for Networking and Comm. Systems (ANCS'05)*, Oct. 2005.
[56] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," *Proc. of IEEE INFOCOM'00*, vol. 2, pp. 556–564, Mar. 2000.
[57] 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*, vol. 17, no. 6, pp. 1030–1039, Jun. 1999.
[58] I. Stoica and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch," *Proc. 6th IEEE/IFIP Int'l Workshop Quality of Service (IWQoS'98)*, pp. 218–224, 1998.
[59] D. Pan and Y. Yang, "Localized independent packet scheduling for buffered crossbar switches," *IEEE Transactions on Computers*, vol. 58, no. 2, pp. 260–274, Feb. 2009.
[60] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (CICQ) switches," *Proc. IEEE Int'l Conf. Comm. (ICC'04)*, vol. 2, pp. 1090–1096, Jun. 2004.
[61] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: combined input-once-cell-crosspoint buffered switch," *Proc. IEEE Workshop High Performance Switching and Routing (HPSR'01)*, Jul. 2001.
[62] D. Stephens and H. Zhang, "Implementing distributed packet fair queueing in a scalable switch architecture," *Proc. IEEE INFOCOM'98*, pp. 282–290, Mar. 1998.
[63] T. Sauter, J. Jasperneite, and L. L. Bello, "Towards new hybrid networks for industrial automation," *Proc. of ETFA'09*, 2009.
[64] *CAN Specification (version 2.0)*. Robert Bosch GmbH, 1991.
[65] *TTEthernet Specification*. TTTech Computertechnik AG, 2008.
[66] F. Dopatka and R. Wismuller, "Design of a realtime industrial - ethernet network including hot-pluggable asynchronous devices," *IEEE International Symposium on Industrial Electronics (ISIE 2007)*, 2007.
[67] G. M. Mair, "Towards transparent telepresence," *Proc. of ICVR'07*, 2007.
[68] R. Santos, R. Marau, A. Vieira, P. Pedreiras, A. Oliveira, and L. Almeida, "A synthesizable ethernet switch with enhanced real-time features," *IEEE IECON'09*, 2009.
[69] Y.-W. Leung and T.-S. Yum, "A TDM-based multibus packet switch," *IEEE Trans. on Communications*, vol. 45, no. 7, pp. 859–866, Jul. 1997.
[70] Q. Wang, S. Gopalakrishnan, X. Liu, and L. Sha, "A switch design for real-time industrial networks," *Proc. of IEEE RTAS'08*, pp. 367–376, Apr. 2008.

Qixin Wang (M'08) received the B.E. and M.E. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1999 and 2001, respectively, and the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign in 2008. He is currently an Assistant Professor in the Department of Computing at the Hong Kong Polytechnic University. His research interests include real-time/embedded systems and networking, wireless technology, and their applications in industrial control, medicine, and assisted living. He has received a best paper award from the IEEE Transactions on Industrial on Informatics (2008). Dr. Wang is a member of the ACM.



Sathish Gopalakrishnan is an Assistant Professor of Electrical and Computer Engineering at The University of British Columbia. He works on problems related to scheduling, resource management and reliability in computer systems, with an emphasis on real-time systems and wireless networks. He completed his graduate work at the University of Illinois at Urbana-Champaign (PhD in Computer Science and MS in Applied Mathematics) and his undergraduate work at the University of Madras. He has received a best paper award from the IEEE Transactions on Industrial Informatics (2008) and a best student paper award at the IEEE Real-Time Systems Symposium (2004).