Real-Time Syst (2013) 49:117–135
DOI 10.1007/s11241-012-9169-6

Design of a crossbar VOQ real-time switch with clock-driven scheduling for a guaranteed delay bound

Kyungtae Kang · Kyung-Joon Park · Lui Sha · Qixin Wang

Published online: 28 November 2012

Abstract Most commercial network switches are designed to achieve good average throughput and delay needed for Internet traffic, whereas hard real-time applications demand a bounded delay. Our real-time switch combines clearance-time-optimal switching with clock-based scheduling on a crossbar switching fabric. We use real-time virtual machine tasks to serve both periodic and aperiodic traffic, which simplifies analysis and provides isolation from other system operations. We can then show that any feasible traffic will be switched in two clock periods. This delay bound is enabled by introducing one-shot traffic, which can be constructed at the cost of a fixed delay of one clock period. We carry out simulation to compare our switch with the popular *i*SLIP crossbar switch scheduler. Our switch has a larger schedulability region, a bounded lower end-to-end switching delay, and a shorter clearance time which is the time required to serve every packet in the system.

Keywords Real-time switch · Bounded delay · Schedulability analysis · Clock-driven scheduling

This work was supported by the research fund of Hanyang University (HY-2011-N).

K. Kang
Department of Computer Science and Engineering, Hanyang University, Ansan Kyeonggi-do, Korea
e-mail: ktkang@hanyang.ac.kr

K.-J. Park (✉)
Department of Information and Communication Engineering, DGIST, Daegu, Korea
e-mail: kjp@dgist.ac.kr

L. Sha
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: lrs@illinois.edu

Q. Wang
Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
e-mail: csqwang@comp.polyu.edu.hk

1 Introduction

By seamlessly integrating sensing, networking, and computation with control of physical devices/objects, *Cyber-Physical Systems* (CPS) are expected to transform the way we interact with the physical world (Stankovic et al. 2005; Sha et al. 2008).

A large number of typical CPS applications are distributed real-time applications, such as telepresence, smart grid, X-by-wire vehicle/avionics, factory automation/flexible manufacturing, robotic collaboration, etc. (Poovendran et al. 2012; Wang 2008). These distributed real-time CPS applications need real-time networks as their system integration venue. Meanwhile, as these applications rapidly scale up, the underlying real-time networks must also expand, evolving from single-hop to multi-hop switched networks.

To build *real-time* multi-hop switched networks, we must have real-time switches. Unlike single-hop real-time networks, where several mature switch/bus standards dominate (Profibus & Profinet International 2012; TTEthernet Specification 2008); the switch architecture for real-time multi-hop networks is still quite open and undecided. Nonetheless, one family of switch architectures are of particular importance: the switch architectures built upon the *crossbar VOQ* (see Sect. 2) infrastructure (Elhanany et al. 2001; McKeown 1999; Peterson and Davie 2000; Gopalakrishnan et al. 2006). The crossbar VOQ infrastructure has many critical features.

Feature 1 It supports full parallelism of N ports.
Feature 2 It avoids the N-speedup problem (see Sect. 2) of output queueing.
Feature 3 It addresses the *Head-Of-Line* (HOL) blocking problem (see Sect. 2) of direct input queueing.

Because of the above features, crossbar VOQ becomes a de facto standard infrastructure for nowaday's high performance switch architectures. Particularly, a crossbar VOQ switch architecture, *i*SLIP (Elhanany et al. 2001; McKeown 1999), has become a mainstream crossbar VOQ switch architecture as well as a mainstream Internet switch architecture (Gopalakrishnan et al. 2006; Wang and Gopalakrishnan 2010).

However, existing crossbar VOQ switch architectures are typically designed for Internet traffic. Internet traffic is transient and requires no deterministic end-to-end delay guarantee. In contrast, real-time networking traffic is typically persistent, and requires deterministic end-to-end delay bound. This design philosophy mismatch makes existing crossbar VOQ switch architectures unfit for real-time multi-hop networks. A typical example is that the end-to-end delay bound of *i*SLIP switched networks is still an open problem (Gopalakrishnan et al. 2006).

Therefore, in order to exploit the features of crossbar VOQ and to lay smooth evolution path for crossbar VOQ switch manufacturers toward the real-time multi-hop switched networks market, it is important to propose a different crossbar VOQ switch architecture, i.e., a crossbar VOQ *real-time* switch architecture.

In this paper, we address this demand of a real-time switch architecture. Multi-hop networks of our proposed crossbar VOQ real-time switches shall provide end-to-end delay bound. We achieve this goal by carrying out clock-driven scheduling and combining crossbar VOQ with an additional buffer. The additional buffer isolates packets

of different clock-periods. This simplifies the scheduling, and guarantees that packets arriving in each clock-period are switched within the next clock-period. On the other hand, *clearance-time* (see Sect. 3.2 for definition) optimal scheduling is carried out to minimize the duration of clock-period, and hence the end-to-end delay bound. We conduct extensive simulations to evalute our proposal. The results show that our crossbar VOQ real-time switch can provide high schedulablity, large throughput, and low end-to-end delay bound.

The remainder of this paper is organized as follows: In Sect. 2, we introduce crossbar switch architecture and discuss the *i*SLIP crossbar switch scheduler. In Sect. 3, we present a real-time switching algorithm, based on clock-driven scheduling, which can guarantee a bounded delay for any feasible traffic. We report on a numerical study of this algorithm in Sect. 4, and then go on to assess its schedulability and delay performance. We review related work on switch design in Sect. 5. Finally we present our conclusions and suggest some future research avenues in Sect. 6.

2 Background

2.1 Crossbar VOQ switches

There are mainly three possible approaches to build a switch: output queueing, input queueing, and *Virtual Output Queueing* (VOQ).

In output queueing, packets are only queued at the output ports. Though simple to model and analyze, output queueing is rarely used in high performance switches due to its inborn N-speedup problem (Gopalakrishnan et al. 2006): Suppose a switch as N input/output ports, and each input port's capacity is C; if at a time instant all input ports work at full capacity and all incoming packets route to the same output port, then the switch internal fabric capacity at the output port must be no less than $N \times C$, otherwise some packets will get lost as the input port cannot buffer the packet. Because of N-speedup problem, even though a switch manufacturer can make switch circuit fabric of capacity C, the output queueing switch can only declare a per input port capacity of C/N.

Because of N-speedup problem, nowadays switches ususally adopt input queueing instead. Under input queueing, packets are only queued at the respective input ports. Output ports do not have buffers. Input ports and output ports are typically connected via a *crossbar* fabric as shown in Fig. 1, in which N input ports are connected to N output ports via the $N \times N$ crossbar fabric. Each intersection of the crossbar fabric (the grey dots in the figure) can be connected/disconnected electronically. At any time instant, at most N input ports can be connected to N output ports, enabling N parallel network flows.

An important rule of crossbar is as follows:

Crossbar Rule *At any time instant, one input port can only be connected to at most one output port, and vice versa.*

Corresponding to this rule, input queueing typically assumes that packets are of a network-wide standard fixed size; such packets are also called "*cells*", and messages

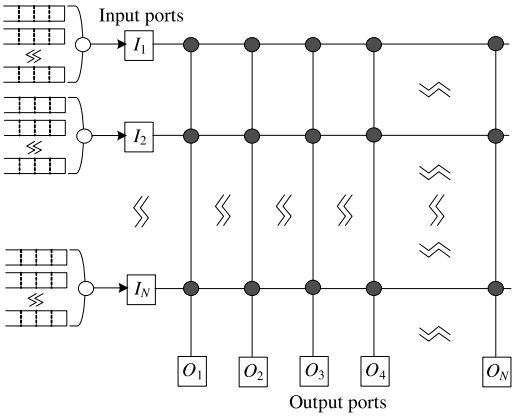


Fig. 1 $N \times N$ crossbar VOQ switch fabric

of various sizes are fragmented into cells when being transferred in the network. The time cost to transmit a packet from an input port to an output port is therefore also fixed, which is called a "*time-slot*", a.k.a. *cell-time*. An input queueing switch schedules the crossbar periodically, and every time-slot is a period. At the beginning of each time-slot, the input queueing switch find a matching between input ports and output ports, and connect/disconnect crossbar intersections accordingly. During the time-slot, the header packet (if there is one) in each matched input queue is transferred to the matched output port via crossbar.

Input queueing, however, has a well-known problem of *Head-Of-Line* (HOL) blocking; unless a packet becomes its input queue's header packet, it must wait in the queue even if its destination output port is idle. Study shows that HOL blocking can reduce switch throughput by over 40 % (Karol et al. 1987).

The HOL problem is addressed by *Virtual Output Queueing* (VOQ). In VOQ, each input port maintains N queues instead of just one: one queue for each output respectively. These input port queues are hence called "*virtual output queues*".

VOQ eliminates HOL blocking, but packets from virtual output queues at different inputs may still contend for the same output port. Various specific crossbar VOQ switch architectures have been proposed to reduce this contention, so as to improve hardware utilization. To the best of our knowledge, one mainstream architecture is *i*SLIP (Elhanany et al. 2001; McKeown 1999), which we will describe in Sect. 2. Although *i*SLIP utilizes switch hardware efficiently and is simple to implement, its deterministic delay bound is very hard to derive. Specifically, its known single-hop delay bound is quite large, and deriving a meaningful end-to-end delay bound of *i*SLIP switched networks is still an open problem (Gopalakrishnan et al. 2006). Therefore, *i*SLIP cannot serve as a crossbar VOQ *real-time* switch architecture. In fact, to our best knowledge, how to design a crossbar VOQ real-time switch remains an open problem.

We will now analyze crossbar VOQ switch in more detail. Let queue (i, j) hold the packets traveling from input i to output j. In addition, let $A_{ij}(t)$ denote the number of packets arriving at queue (i, j) at time-slot t, and let $Q_{ij}(t)$ denote the number of packets in the queue (i, j) during the same time-slot. Then the switching decision

variable $S_{ij}(t)$ at time-slot t can be defined as follows:

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

Note that the switching matrix $[S_{ij}]$ must comply with the crossbar rule. The number of packets in queue (i, j) at the next time-slot $t + 1$ is then

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

The maximum operation on the right-hand side of this equation is present to cover the case in which $Q_{ij}(t) = 0$ and $S_{ij} = 1$. Informally, we can say that the purpose of a real-time switch scheduling algorithm is to generate a crossbar rule compliant switching matrix $[S_{ij}]$ at the beginning of each time-slot.

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

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

It is well known (Peterson and Davie 2000) that the stability region of a switch can be expressed as the set of rate matrices that satisfy the following $2N$ inequalities:

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

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

In other words, every queue in the crossbar VOQ switch is certain to maintain finite length, as long as the arrival rates remain in the stable region. If inequality (2), (3) do not hold for any i or j, the corresponding queue is going to grow without limit. It is not a simple task to develop a crossbar VOQ switch scheduling algorithm that is efficient, able to serve any periodic and aperiodic traffic, and sure to stay in the stability region. Switching algorithms that can handle any traffic within the stability region are called *throughput-optimal*.

We will now introduce a graph-theoretic approach to the design of crossbar VOQ switch scheduling algorithms. We can transform the crossbar VOQ packet scheduling problem into a matching problem in bipartite graphs as follows. At each time-slot, let G denote a bipartite graph connecting the input ports to the output ports. Each port in the switch is considered as a node in G (thus *port* and *node* will be used interchangeably). There will be an edge connecting input port i and output port j in G if there are packets waiting at input port i that are destined for output port j. A crossbar VOQ scheduling algorithm is in charge of finding a matching M in G. Each resulted

matching corresponds to a switching matrix $[S_{ij}]$ (see (1)), and determines which input ports will send packets to which output ports for the current time-slot.

Now, let Q_i denote the total number of packets waiting at the input port i and Q_j denote the total number of packets (at all the inputs) destined for output port j. Thus $Q_i = \sum_{j=1}^{N} Q_{ij}$ and $Q_j = \sum_{i=1}^{N} Q_{ij}$. In the later part of this paper, we will use Q_i and Q_j as weights of corresponding nodes in the bipartite graph G, to assist scheduling algorithm design.

2.2 *i*SLIP crossbar VOQ switch

*i*SLIP (McKeown 1999) is a popular scheduling mechanism for virtual-output queue crossbar switches, which has now been extended in several directions, with variations such as weighted *i*SLIP and prioritized *i*SLIP. Although *i*SLIP is old academically, it is the current de facto standard switch architecture implemented in most commercial-off-the-shelf switches (Wang and Gopalakrishnan 2010).

Commercial *i*SLIP switches are typically based on a combination of ideas from several of these variations. According to McKeown (1999), *i*SLIP can achieve 100 % throughput for uniform traffic, meaning that every output reaches maximum capacity. In this situation, a complete matching in the bipartite graph between the inputs and outputs defined by the crossbar fabric is found at every time-slot. If the traffic is not uniform, McKeown's results suggest that *i*SLIP adapts to a fair scheduling policy which never discriminates against any input queue.

Although *i*SLIP is simple to implement and uses the switch hardware efficiently, it does not provide any guarantee of real-time performance, and the determination of usefully tight delay bounds for *i*SLIP remains an open problem (Gopalakrishnan et al. 2006). The best delay bound currently available is still very pessimistic (Gopalakrishnan et al. 2006). For example, if every input to an $N \times N$ *i*SLIP switch has periodic real-time traffic going to every output, the known single-hop delay bound $\delta_{i\text{SLIP}}$ for packets from input I_i to output O_j is

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

where $C_{i,j,k}$ is the transmission time for each packet of the kth real-time flow from I_i to O_j. See how badly this works even for some reasonable numbers: If N is 32, $C_{i,j,k}$ is the same for all links and flows, and there are 100 real-time flows going from I_i to O_j, then the single-hop delay bound is a factor of 102,400 greater than the packet transmission time.

3 Design of a real-time switch

The key issue in the design of a real-time packet switch is to handle contention in a way that bounds the buffering delay. This in turn requires the switching delay to be bounded. In this section, we present a real-time switching algorithm for the widely-adopted VOQ architecture that can guarantee a bounded delay with any feasible traffic.

Our aim is to address real-time applications in which the dominant traffic, from applications such as sensing, actuating, and video monitoring, is periodic. Any aperiodic traffic can be served by periodic virtual-machine (VM) tasks (each VM-task is denoted by (L, C), which serves traffic C times during each clock period of L time-slots) (Lipari and Bini 2003; Deng and Liu 1997; Kuo and Li 1999; Davis and Burns 2005, 2006). So we assume all traffic is *periodic* in the following analysis. One simple extension to aperiodic traffic is as follows: An aperiodic flow can be wrapped and served by a periodic server at the source end. The server injects C packets every L time slots into the network. If there happens to be aperiodic flow data, the packet is loaded with data, otherwise the packet is padded with zeros. Note that, in this simple extension, there will be a certain level of waste of resource for processing packets with zero-size data. More advanced extension with high efficiency will be a subject of future work.

3.1 Clock-driven scheduling as a VM-task

We begin by introducing the concept of clock-driven scheduling (Liu 2000) as a VM-task. A widely used approach to the scheduling of real-time VM-tasks is clock-driven scheduling (Liu 2000). Suppose a VM-task (L, C) and let f_k^{ij} denote the kth real-time flow from input port I_i to output port O_j, where $k = 1, 2, \ldots, K_{ij}$, and K_{ij} is the number of flows from I_i to O_j. Additionally, we associate f_k^{ij} with a VM-task (L, C_{ijk}), which is written in this way because f_k^{ij} has C_{ijk} packets to be served. Using clock-driven scheduling, the delay to f_k^{ij} will be bounded as long as the switching algorithm can guarantee a worst-case bound on the time required to forward all the C_{ijk} packets of the flow f_k^{ij}.

Let C_{ij} denote the total number of packets to be forwarded from I_i to O_j, so that $C_{ij} = \sum_{k=1}^{K_{ij}} C_{ijk}$. The sets of VM-tasks $\{(L, C_{ijk})\}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$, $k = 1, \ldots, K_{ij}$ must meet the stability condition expressed by (2) and (3). In addition, we quote the feasibility condition from Wang et al. (2008) for the real-time traffic during each clock period of L time-slots as follows:

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

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

The traffic admitted into the switch (more specifically, all traffic admitted into the switch's input ports) must comply with this schedulability test of (2) and (3). Hence all packets admitted are schedulable. *We do not consider infeasible traffic which does not meet these conditions, and is naturally unschedulable.*

In summary, our real-time switch serves each traffic flow as a VM-task, and each VM-task is served by a slot-by-slot switching algorithm that minimizes the clearance time for any feasible traffic. We will explain this in the next section. Note that our

method of clock-driven scheduling naturally makes the network operate in a synchronous manner, which is critical in reducing the amount of buffering required by a switch.

3.2 Clearance-time-optimal switching policies

In order to design a switch with a limited delay, we need a clearance-time-optimal switching policy that minimizes the clearance time.

The notion of *clearance time* is as follows. We say that a switch has *one-shot traffic*, if the number of packets in every queue in the switch is initially given, and there are no further arrivals. Clearance time is defined as the time required to serve every packet in the switch under one-shot traffic. *Clearance-time-optimal switching policy* is a switching policy that minimizes clearance time.

As we explained in the previous section, the crossbar constraints mean that no more than one packet can be switched at any port. Consequently, an obvious lower bound on the clearance time T_{clear} is defined by (7):

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

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

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

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

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

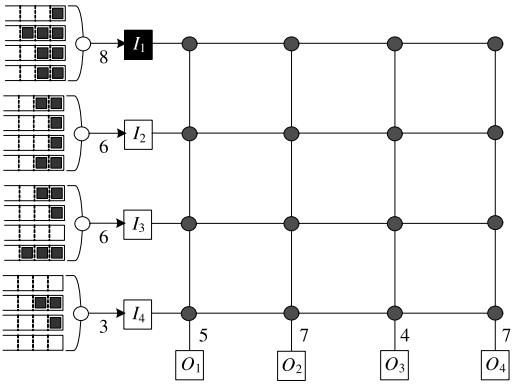


Fig. 2 Illustration of critical-port switching with a 4×4 switch fabric

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

An illustrative example of a critical-port switching algorithm is shown in Fig. 2. In this figure the number next to each input port I_i and output port O_j denotes their current weights Q_i and Q_j respectively. While the meaning of Q_i is obvious, we must explain that Q_j denotes the total number of packets at any input port *destined* for output port O_j. In addition, the jth sub-queue (numbered from the top) at each input port I_i is the current number of packets Q_{ij} destined for output port O_j. Thus it should be apparent that each Q_j is the sum of the lengths Q_{ij} of the sub-queues at each input port i. For example, the weight applied to output port O_1 is $Q_{O_1} = Q_{11} + Q_{21} + Q_{31} + Q_{41} = 1 + 2 + 2 + 0 = 5$. In the state of the queues shown in Fig. 2, input port I_1 has a larger weight than any other input or output port, making I_1 the unique critical port. Hence, a critical-port switching algorithm must now serve I_1 with highest priority.

3.3 Design of a real-time switch with clock-driven scheduling

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

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

Algorithm 1 Clock-based switching algorithm

```
for each clock period do
    Switch the packets that arrived in the previous clock period using Algorithm 2
end for
```

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

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

Among many possible realizations of critical-port policies, we adopt lazy heaviest-port first (LHPF) matching (Gupta et al. 2009), which we will now explain: First, the threshold th of a matching M is defined as the lowest integral weight for which M matches all the ports. For example, a perfect matching that switches every port with packets in its queue has $th = 1$. An LHPF matching has the lowest threshold of all possible matchings. Algorithm 2 is an implementation of LHPF matching.

As a simple illustration of an LHPF matching, we revisit Fig. 2, where the smallest port weight is 3 of I_4. Hence, even when a matching matches all the ports, its threshold will be 3 by the definition of the threshold. In fact, by observation, we can easily find a matching that matches every port, e.g., (I_1, O_4), (I_2, O_2), (I_3, O_1), (I_4, O_3). Consequently, this matching is an LHPF matching with the threshold of 3. It should be noted that LHPF matching in Fig. 2 is not unique since any matching that matches every port is an LHPF matching.

The LHPF class of policies has optimal throughput (Gupta et al. 2009), which means that the length of the queue at every switch is guaranteed to remain finite for any traffic that satisfies the stability condition of (2) and (3), even if that traffic does not satisfy the feasibility condition of (5) and (6). This extends the schedulablity region of the proposed switching algorithm, and this will be clarified in the numerical study in the next section.

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

It has been shown (Mekkittikul and McKeown 1998) that the members of LHPF class of policies have a time complexity of less than $O(N^{2.5})$, compared to $O(N^3)$ (Karp and Hopcroft 1973) for maximum weight matching (Karp and Hopcroft 1973; Shah and Wischik 2006). The class of LHPF matching algorithms contains policies which are simple to implement, making them suitable for a low-complexity delay-efficient scheduler with theoretical guarantees on the throughput. Additionally, these policies are clearance-time-optimal because any critical-port policy is clearance-time-optimal as proved in Proposition 1.

4 Performance evaluation

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

4.1 Schedulability

First, we compare the schedulability of *i*SLIP and our scheme by simulation. We set the clock period L to 1 ms, the capacity of each port is 1, 10, or 100 Gb/s, the number of input ports N is 4, 8, or 16, and the packet size is fixed[1] at 10 kb. The curves in Figs. 3, 4, and 5 show average results for 1000 simulation runs. We generate random traffic with uniform distribution for every input port in each simulation run. Note that our algorithm guarantees a deterministic delay bound for any feasible traffic while no algorithm can guarantee any deterministic bound for infeasible traffic. Instead of obvious schedulability of feasible traffic, as meaningful measures, we are showing the schedulability ratio and the clearance time in our simulation study over the entire region of demand utilization that contains both feasible and infeasible regimes.

Demand utilization is the ratio between the average traffic at a port and the port capacity. Schedulability depends on the distribution of traffic across the input ports, and is less than unity due to the crossbar constraint explained in Sect. 2: it is the contention among ports that stops every packet at an input port from being switched in a single time-slot. Figures 3, 4, and 5 show how the schedulability drops as the

[1] In case messages are of different sizes, before their injection into the MAC layer of the source end network interface, they will be fragmented into the fixed standard-size packets. Hence in the physical layer of the network, packets are all of the same standard size. This is already a common practice in industrial fieldbuses (Wang and Gopalakrishnan 2010; Wang et al. 2008; Dopatka and Wismuller 2007; Leung and Yum 1997).

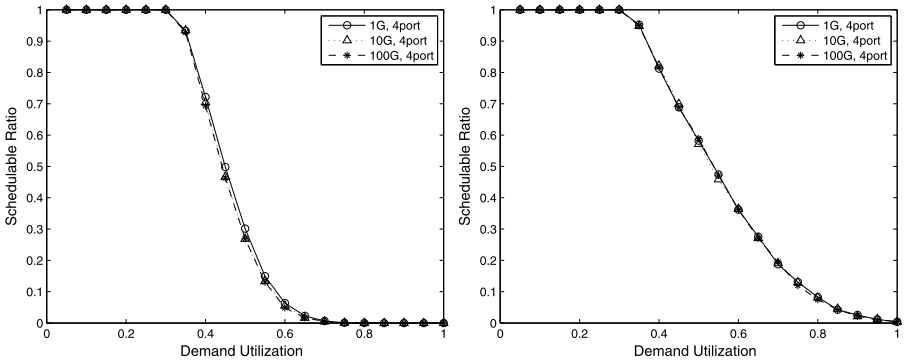


Fig. 3 Schedulability of *i*SLIP and the proposed scheme with port capacities of 1, 10, and 100 Gb/s (4 input ports)

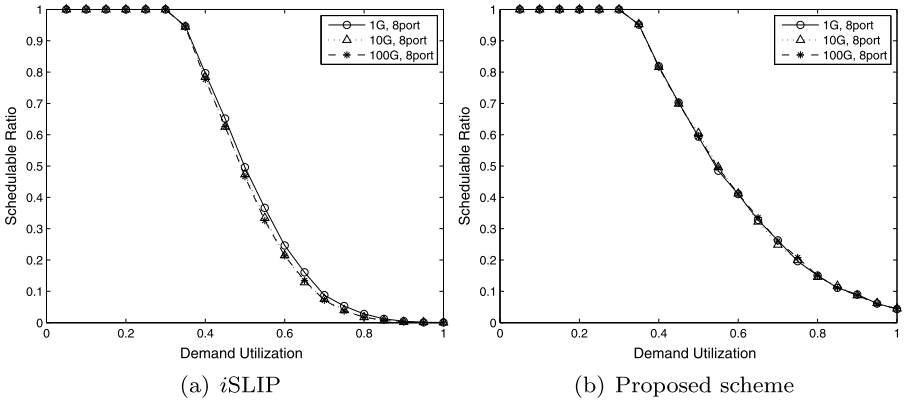


Fig. 4 Schedulability of *i*SLIP and the proposed scheme with port capacities of 1, 10, and 100 Gb/s (8 input ports)

demand utilization increases, and eventually the traffic cannot be scheduled at all. In these simulations periodic traffic, corresponding in quantity to the demand utilization, was generated and assigned across all the inputs. We expect infeasible traffic is likely to be generated more often as the demand utilization increases, and the schedulability of the switch decreases accordingly.

Figures 3, 4, and 5 also indicate that our approach has better schedulability than *i*SLIP in all cases, although the difference becomes smaller as the number of input ports increases. This is of minor importance because our primary application scenario is an embedded system, in which the number of input ports is usually small. A more significant way in which our switch design is an improvement on *i*SLIP is that it guarantees all feasible traffic will be switched in two clock periods, as we have shown. As a result, within the range of demand utilization for which all the traffic arriving at the switch is expected to be feasible, our switching algorithm bounds the delay incurred at the switch. For example, we can infer from these figures that if the clock

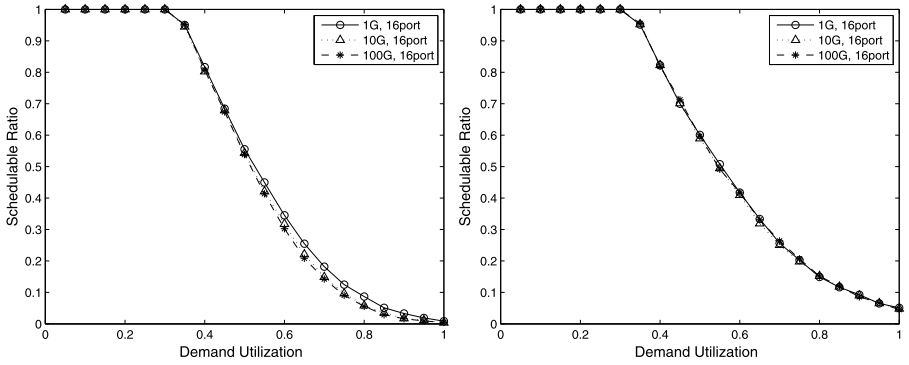


Fig. 5 Schedulability of *i*SLIP and the proposed scheme with port capacities of 1, 10, and 100 Gb/s (16 input ports)

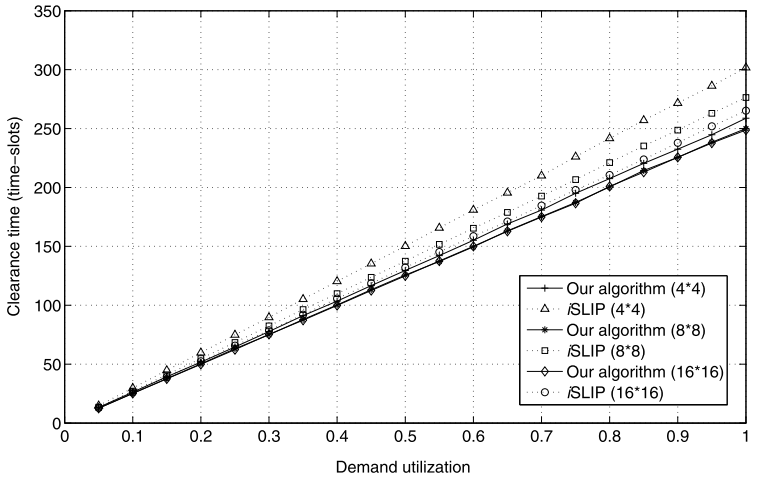


Fig. 6 Comparison between the clearance time of *i*SLIP and the proposed scheme when the number of input ports N is 4, 8, and 16 and the port capacity is 1 Gb/s

period is 1 ms and the demand utilization is less than 0.3, all the traffic arriving at the switch is expected to pass through in 2 ms regardless of the port capacity of the switch and the number of ports.

4.2 Clearance time

We now compare the clearance time of the proposed scheme to that of the *i*SLIP scheme using the one-shot traffic model described in Sect. 3.2, while focusing on the port capacity of 1 Gb/s. Figure 6 shows the clearance time of each scheme against demand utilization. Again we have averaged 1,000 simulation runs.

Figure 6 shows that the clearance time increases as the number of input ports increases because of increased contentions, as we would expect. We can also see

that the clearance time increases almost linearly with the demand utilization for both schemes, but the gradient is steeper with *i*SLIP, which means that our switch algorithm provides better performance in terms of the switching delay.

4.3 End-to-end switching delay

Let $\delta_{SW,i}$ be the switching delay introduced by the ith switch between the source of a packet traversing a network and its destination. The propagation delays between switches are negligible. Thus, if we assume that the traffic input to all the intermediate switches between source and destination is feasible, the end-to-end switching delay δ_{E2E} using the proposed switches for any feasible traffic can be bounded as follows:

$$\delta_{\mathrm{E2E}} = \sum_{i=1}^{H} \delta_{\mathrm{SW},i} \leq 2HL, \tag{8}$$

where H is the hop-count (the number of switches that each packet traverses), and L is the clock period of switches between source and destination (here we assume all switches have the same clock period and the length of time slot is globally the same). This bound holds because any feasible traffic satisfying (5) and (6) is guaranteed to be switched in two clock periods at each switch.

The clock period of L time-slots is much shorter than the delay constraint of a typical real-time application. Assuming that the clock periods of all switches are 1 ms and the maximum hop count is 15, any feasible traffic is always switched in 2 ms at each switch, and the resulting end-to-end switching delay will be less than 30 ms. This period is significantly less than the end-to-end delay of 50 ms allowable in typical real-time control and automation applications such as sensing and actuation (Fisher et al. 2004) or video monitoring (Fisher et al. 2006). Conversely, the *single-hop* delay in an *i*SLIP switch, obtained from (4), may reach up to 100 ms in a similar situation.

5 Related work

There has been extensive research on the design of Internet switches, for example, McKeown (1999), Gupta et al. (2009), Weller and Hajek (1997), Rexford et al. (1998), Neely et al. (2007). Internet switches typically try to meet delay-related QoS constraints by prioritizing shared queues in switches: Flows of the same priority share the same queue. For example, Internet switches usually have 4 to 8 priorities. Some more recent switches deploy even more sophisticated mechanisms such as rate limitation and server-based traffic management (Cisco 2012). Considerable effort has been devoted to analyzing the performance of high-speed Internet switches and routers, and obtaining statistical or average delay bounds (Shah et al. 2004, 2007; Deb et al. 2006). But Internet switches are designed for Internet traffic, which is transient and best-effort, rather than requiring deterministic real-time delay bound.

However, through the decades of evolution of Internet switch architecture, the crossbar VOQ switch architecture family becomes the de facto standard due to crossbar VOQ's three critical features (see Feature 1–3 in Sect. 1). To maintain the three

critical features of crossbar VOQ, and to lay a smooth evolution path for crossbar VOQ manufacturers toward the multi-hop real-time switched networks market, it is necessary to propose a crossbar VOQ *real-time* switch architecture.

The closest existing candidate is the *i*SLIP (McKeown 1999) crossbar VOQ switch architecture. However, *i*SLIP switched networks end-to-end delay bound is still an open problem (Gopalakrishnan et al. 2006).

There is a very well-known crossbar real-time switch architecture: The so-called *TDMA Crossbar Real-Time* (TCRT) real-time switch architecture (Wang and Gopalakrishnan 2010; Wang et al. 2008; Dopatka and Wismuller 2007; Leung and Yum 1997; Chang et al. 1999; Chen et al. 2011; Rao et al. 2012). However, this architecture requires *per-flow queueing*, hence is not a crossbar *VOQ* switch architecture.

Consider the number of input/output ports of a switch N as a constant, the routing overhead for per-flow queueing is $O(\log n)$, while that for VOQ is $O(1)$, where n is the number of flows passing through an input port. Therefore, our crossbar VOQ real-time switch architecture is more scalable.

Meanwhile, Qixin et al. (Wang and Gopalakrishnan 2010) implemented the TCRT switch architecture using FPGA (E2E Real-Time Solution for Avionics and Control Demo 2012). This gives insights on the feasibility of implementing our crossbar VOQ real-time switch architecture. Compared to TCRT switch architecture, which carries out per-flow queueing, our design is more scalable. Also, TCRT switch architecture corresponds to a scheduling time complexity of $O(N^4)$; while ours is $O(N^{2.5})$, where N is the number of input/output ports. The only thing that our design may incur more complexity is the use of double bufferring for each VOQ. Nevertheless, this is not difficult to implement. Therefore, in summary, the difficulty of implementing our crossbar VOQ real-time switch architecture should be comparable to (or even simpler than) that of implementing the TCRT switch architecture, which is already proven to be feasible by Qixin et al. (Wang and Gopalakrishnan 2010).

There are other real-time switch architectures/standards. Sha et al. (1990) and Gopalakrishnan et al. (2004) have proposed prioritized bus/ring based real-time switches. These are single-hop real-time swtich architectures that does not belong to the crossbar VOQ family. Rexford et al. (1998) proposed a router for real-time communication, but this was designed to support deadline-based scheduling, which also requires a significantly different infrastructure than crossbar VOQ. The same remark applies to the real-time Ethernet switch proposed by Venkatramani et al. (1997), and the shared medium real-time switch architecture proposed by Santos et al. (2010).

TTEthernet Specification (2008) is an industrial fieldbus standard that supports hard real-time over multi 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 already has deterministic end-to-end delay bound. TTEthernet standard itself is open to various detailed switch designs. Therefore, our crossbar VOQ real-time switch can complement TTEthernet by providing a detailed switch design that matches the TTEthernet's specifications.

PROFINET (Profibus & Profinet International 2012) is another fieldbus standard that supports hard real-time over multi hops of switches. However, in that case,

PROFINET requires that all nodes on the network exclusively use PROFINET's proprietary network stacks. PROFINET therefore does not lay a smooth evolution path for generic crossbar VOQ switch manufacturers. In contrast, our switch architecture is a member of the de fact standard of crossbar VOQ switch family. The evolution path for crossbar VOQ switch manufacturers is much easier.

6 Conclusions and prospects

The convergence of the computer and the physical world is the primary theme of research on next-generation networking. This convergence requires a real-time network infrastructure, which in turn needs high-speed real-time WANs as its backbone. However, today's commercially available high-speed WAN switches are designed for best-effort transmission of Internet traffic. The real-time switches needed for real-time networks are not yet in existence.

In this article, we have proposed a real-time switch design which is based on the application of clock-driven scheduling to the most widely adopted crossbar switch architecture. Our switch is designed for periodic traffic, but aperiodic traffic can also be handled if it is packaged as real-time virtual-machine tasks. This simplifies analysis, provides isolation from other system operations, and facilitates hierarchical scheduling and flow aggregation.

Our real-time switch design has several features that distinguish it from previous proposals: First, it bounds the delay to any periodic feasible traffic. Second, as we have shown by extensive simulations, its utilization and clearance time that compare favorably with the *i*SLIP crossbar scheduler, which is already widely implemented in commercial products.

Nevertheless, several issues remain for future research. First of all, it is essential to have an accurate profile of the workload generated by real-time applications, and to know how predictable it really is, if we are to design efficient infrastructure for these applications. Further, we need to ensure that changes in workload, even though they are infrequent and occur during planned outages, can be accommodated by simple reconfiguration techniques. In future work, we plan to extend our switch design to support run-time adaptation, hierarchical scheduling, and flow aggregation. The pursuit of this goal could usefully draw on a number of recent research contributions to networking research.

References

Chang C-S, Chen W-J, Huang H-Y (1999) On service guarantees for input buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann. In: Proc of IEEE IWQoS 1999, pp 79–86

Chen L, Liu X, Wang Q, Wang Y (2011) A real-time multicast routing scheme for multi-hop switched fieldbuses. In: Proc of INFOCOM 2011, pp 3209–3217

Cisco (2012) Catalyst 3560 series switching solutions. http://www.cisco.com

Davis R, Burns A (2005) Hierarchical fixed priority preemptive scheduling. In: Proc IEEE real-time systems symposium (RTSS 2005), Miami, FL, USA. IEEE Comput Soc, Los Alamitos, pp 389–398

Davis R, Burns A (2006) Resource sharing in hierarchical fixed priority pre-emptive systems. In: Proc IEEE real-time systems symposium (RTSS 2006), Rio de Janeiro, Brazil. IEEE Comput Soc, Los Alamitos, pp 257–270

Deb S, Shah D, Shakkottai S (2006) Fast matching algorithms for repetitive optimization: an application to switch scheduling. In: Proc conference on information, sciences and systems (CISS 2006). Princeton, New Jersey, USA, pp 1266–1271. IEEE Information Theory Society

Deng Z, Liu JW-S (1997) Scheduling real-time applications in an open environment. In: Proc IEEE real-time systems symposium (RTSS 1997), San Francisco, CA, USA. IEEE Comput Soc, Los Alamitos, pp 308–319

Dopatka F, Wismuller R (2007) Design of a realtime industrial Ethernet network including hot-pluggable asynchronous devices. In: Proc IEEE international symposium on industrial electronics (ISIE 2007), Vigo, Spain. IEEE Comput Soc, Los Alamitos, pp 1826–1831

E2E Real-Time Solution for Avionics and Control Demo (2012) Remote control via real-time switch. http://www.youtube.com/watch?v=f-Q5sFbyfIg

Elhanany I, Kahane M, Sadot D (2001) Packet scheduling in next-generation multiterabit networks. Computer 34(4):104–106

Fisher B, Fels S, MacLean K, Munzner T, Rensink T (2004) Seeing, hearing, and touching: putting it all together. In: Proc international conference on computer graphics and interactive techniques, Los Angeles, CA, USA. ACM, New York

Fisher B, Fels S, MacLean K, Munzner T, Rensink T (2006) Exploiting perception in high-fidelity virtual environments. In: Proc international conference on computer graphics and interactive techniques, Boston, MA, USA. ACM, New York

Gopalakrishnan S, Sha L, Caccamo M (2004) Hard real-time communication in bus-based networks. In: Proc IEEE real-time systems symposium (RTSS 2004), Lisbon, Portugal. IEEE Comput Soc, Los Alamitos, pp 405–414

Gopalakrishnan S, Caccamo M, Sha L (2006) Switch scheduling and network design for real-time systems. In: Proc IEEE real-time and embedded technology and applications symposium (RTAS 2006), San Jose, CA, USA. IEEE Comput Soc, Los Alamitos, pp 289–300

Gupta GR, Sanghavi S, Shroff NB (2009) Node weighted scheduling. In: Proc international joint conference on measurement and modeling of computer systems (SIGMETRICS 2009), Seattle, WA, USA. ACM, New York, pp 97–108

Karol M, Hluchyj M, Morgan S (1987) Input versus output queueing on a space-division packet swtich. IEEE Trans Commun 35(12):1347–1356

Karp R, Hopcroft J (1973) An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J Comput 2:225–231

Kuo T-W, Li C-H (1999) A fixed-priority-driven open environment for real-time applications. In: Proc IEEE real-time systems symposium (RTSS 1999), Phoenix, AZ, USA. IEEE Comput Soc, Los Alamitos, pp 256–267

Leung Y-W, Yum T-S (1997) A TDM-based multibus packet switch. IEEE Trans Commun 45(7):859–866

Lipari G, Bini E (2003) Resource partitioning among real-time applications. In: Proc Euromicro conference on real-time systems (ECRTS 2003), Porto, Portugal. IEEE Comput Soc, Los Alamitos, pp 151–158

Liu JW-S (2000) Real-time systems. Prentice Hall, New York

McKeown N (1999) The *i*SLIP scheduling algorithm for input-queued switches. IEEE/ACM Trans Netw 7(2):188–201

Mekkittikul A, McKeown M (1998) Practical scheduling algorithm to achieve 100 % throughput in input-queued switches. In: Proc IEEE INFOCOM, San Francisco, CA, USA. IEEE Comput Soc, Los Alamitos, pp 792–799

Neely MJ, Modiano E, Cheng Y-S (2007) Logarithmic delay for $N \times N$ packet switches under the crossbar constraint. IEEE/ACM Trans Netw 15(3):657–668

Peterson LL, Davie BS (2000) Computer networks: a system approach. Morgan Kaufmann, San Mateo

Poovendran R, Sampigethaya K, Gupta SKS, Lee I, Prasad KV, Corman D, Paunicka J (2012) Special issue on cyber-physical systems. Proc IEEE 100(1):6–12

Profibus & Profinet International (2012) http://www.profibus.com

Rao L, Wang Q, Liu X, Wang Y (2012) Analysis of TDMA crossbar real-time switch design for AFDX networks. In: Proc INFOCOM 2012, pp 2462–2470

Rexford J, Hall J, Shin KG (1998) A router architecture for real-time communication in multicomputer networks. IEEE Trans Comput 47(10):1088–1101

Santos R, Vieira A, Pedreiras P, Oliveira A, Almeida L, Marau R (2010) Flexible, efficient and robust real-time communication with server-based Ethernet switching. In: Proc IEEE international workshop on factory communication systems (WFCS 2010), Aveiro, Portugal, pp 131–140

Sha L, Rajkumar R, Lehoczky JP (1990) Real-time scheduling support in Futurebus+. In: Proc IEEE real-time systems symposium (RTSS 1990), Lake Buena Vista, FL, USA. IEEE Comput Soc, Los Alamitos, pp 331–340

Sha L, Gopalakrishnan S, Liu X, Wang Q (2008) Cyber-physical systems: a new frontier. In: Proc IEEE international conference on sensor networks, ubiquitous, and trustworthy computing (SUTC 2008), Taichung, Taiwan. IEEE Comput Soc, Los Alamitos, pp 1–9

Shah D, Wischik D (2006) Optimal scheduling algorithms for input-queued switches. In: Proc IEEE INFOCOM, Barcelona, Catalunya, Spain. IEEE Comput Soc, Los Alamitos, pp 1–11

Shah D, Giaccone P, Leonardi E, Prabhakar B (2004) Delay bounds for combined input and output switches with low speedups. Perform Eval 55(1–2):113–128

Shah D, Giaccone P, Leonardi E (2007) Throughput region of finite-buffered networks. IEEE Trans Parallel Distrib Syst 18(2):251–263

Stankovic JA, Lee JA, Mok A, Rajkumar R (2005) Opportunities and obligations for physical computing systems. Computer 38(11):23–31

TTEthernet Specification (2008). TTTech Computertechnik AG

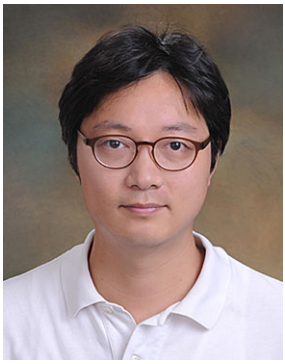
Venkatramani C, Chiueh T (1997) Design and implementation of a real-time switch for segmented Ethernets. In: Proc IEEE international conference on network protocol (ICNP 1997), Atlanta, GA, USA. IEEE Comput Soc, Los Alamitos, pp 152–161

Wang Q (2008) Real-time and embedded systems building blocks for cyber-physical systems. Ph.D. dissertation, Department of Computer Science, UIUC

Wang Q, Gopalakrishnan S (2010) Adapting a main-stream Internet switch architecture for multi-hop real-time industrial networks. IEEE Trans Ind Inform 6(3):393–404

Wang Q, Gopalakrishnan S, Liu X, Sha L (2008) A switch design for real-time industrial networks. In: Proc IEEE real-time and embedded technology and applications symposium (RTAS 2008), St. Louis, MO, USA. IEEE Comput Soc, Los Alamitos, pp 367–376

Weller T, Hajek B (1997) Scheduling nonuniform traffic in a packet-switching system with small propagation delay. IEEE/ACM Trans Netw 5(6):813–823



Kyungtae Kang received the B.S. degree in computer science and engineering, followed by the M.S. and Ph.D. degrees in electrical engineering and computer science, from Seoul National University, Korea, in 1999, 2001, and 2007,respectively. From 2008 to 2010, he was a postdoctoral research associate at the University of Illinois at Urbana-Champaign. In 2011, he joined the Department of Computer Science and Engineering at Hanyang University, where he is currently an assistant professor. His research interests lie primarily in systems, including operating systems, networked systems, sensor systems, distributed systems, and real-time embedded systems. His recent research interest is in the interdisciplinary area of cyber-physical systems. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Kyung-Joon Park received his B.S. and M.S. degrees from the School of Electrical Engineering and Ph.D. degree from the School of Electrical Engineering and Computer Science, Seoul National University, Korea. He is currently an Assistant Professor in the Department of Information and Communication Engineering, DGIST, Korea. He was a Postdoctoral Research Associate in the Department of Computer Science, University of Illinois at Urbana-Champaign, USA from 2006 to 2010. He was with Samsung Electronics, Suwon, Korea as a Senior Engineer, from 2005 to 2006. His current research interests include modeling and analysis of cyber physical systems and design of medical-grade protocols for wireless health care systems.

Lui Sha received the Ph.D. degree from Carnegie Mellon University in 1985. He is the Donald B. Gillies chair professor of computer science at the University of Illinois at Urbana-Champaign. His work on real-time computing is supported by most of the open standards in real-time computing and has been cited as a key element to the success of many national high technology projects including GPS upgrade, the Mars Pathfinder, and the International Space Station. He is a fellow of the IEEE and the ACM.

Qixin Wang received the B.E. and M.E. degrees from Dept. of Computer Sci. and Tech., Tsinghua Univ. (Beijing, China) in 1999 and 2001 respectively; and the Ph.D. degree from the Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign in 2008. He joined the Dept. of Computing in the Hong Kong Polytechnic University in 2009 as an assistant professor. His research interests include cyber-physical systems, real-time/embedded systems/networks, and their applications in industrial control, medicine and assisted living. He has authored/coauthored more than 20 papers in leading publication venues in these fields, including a featured article in IEEE Transactions on Mobile Computing 2008 May Issue and an article winning 2008 best paper award of IEEE Transactions on Industrial Informatics. He is a member of the IEEE and the ACM.