시스템 에어리어 네트웍에서의 동기화 기법
(Synchronization of Network Interfaces in System Area Networks)

송 효 정 †
(Hyo Jung Song)

요 약 많은 Application이 QoS(Quality of Service)를 요구한다. 본 논문에서는 시스템 에어리어 네트웍에서 QoS의 일종인 성능 예측성에 촛점을 맞춘다. 예측성을 만족시키기 위해 네트웍상의 모든 통신 스트림이 정적으로 스케쥴된다고 가정할 때, 각 네트웍 인터페이스는 정해진 스케쥴을 제 때에 실행하여야 하며 이를 위해서는 각 인터페이스의 시간베이스가 모두 동기화 되어야 한다. 본 논문에서는 이 동기화 문제를 다루었다. 동기화 문제를 해결하기 위해서, 본 논문에서는 링크 레벨 흐름 제어 신호를 이용해서 기 제안된 FM-QoS[1] 기법을 확장하였다. FM-QoS는 네트웍이 하나의 스위치로만 구성될 때가능한 동기화 기법이다. 본 논문에서는 임의의 망구조에서 1) 네트웍 인터페이스들이 동기화되기 위한 흐름 제어 신호의 조건 (동기화 스케쥴로 칭함)을 규정하고, 2) 동기화 스케쥴의 정확도를 분석하였다. 제안한 방식의 실례를 보이기 위해서, 하나의 스위치로 구성된 네트웍과 여러 개의 스위치로 구성된 트리 구조 네트웍에서 각기 동기화 스케쥴을 예시하고 이의 정확도를 수치적으로 분석하였다. 분석된 정확도가 실제 시스템에서 어느 정도의 값을 갖는지를 이해하기 위해서, Myrinet 스위치[2]로 구성된 망에 대해서 실험하였다. 실험 중 관찰된 최대 정확도의 에러는 9.2 마이크로 세컨드이며, 이 수치를 바탕으로 본 논문은 제안한 동기화 알고리즘이 시스템 에어리어 네트웍에서 효과적이라고 결론지었다.
키워드 : 동기화, 링크 레벨 흐름 제어, 시스템 에어리어 네트워크

Abstract Many applications in cluster computing require QoS(Quality of Service) services. Since performance predictability is essential to provide QoS service, underlying systems must provide predictable performance guarantees. One way to ensure such guarantees from network subsystems is to generate global schedules from applications' network requests and to execute the local portion of the schedules at each network interface. To ensure accurate execution of the schedules, it is essential that a global time base must be maintained by local clocks at each network interface. The task of providing a single time base is called a synchronization problem and this paper addresses the problem for system area networks.

To solve the synchronization problem, FM-QoS[1] proposed a simple synchronization mechanism called FBS(Feedback-Based Synchronization) which uses built-in flow control signals. This paper extends the basic notion of FM-QoS to a theoretical framework and generalizes it: 1) to identify a set of built-in network flow control signals for synchrony and to formalize it as a synchronizing schedule, and 2) to analyze the synchronization precision of FBS in terms of flow control parameters. Based on generalization, two application classes are studied for a single switch network and a multiple switch network. For each class, a synchronizing schedule is proposed and its bounded skew is analyzed. Unlike FM-QoS, the synchronizing schedule is proven to minimize the bounded skew value for a single switch network.

To understand the analysis results in practical networks, skew values are obtained with flow control parameters of Myrinet-1280/SAN[2]. We observed that the maximum bounded skew of FBS is 9.2μsec or less over all our experiments. Based on this result, we came to a conclusion that FBS was a feasible synchronization mechanism in system area networks.
Key words : synchronization, link level flow control, system area networks, cluster computing

† 정 회 원 : 삼성전자 기술총괄 책임
hyojung_song@hanmail.net
논문접수 : 2005년 1월 24일
심사완료 : 2005년 3월 18일

1. Introduction

Recent dramatic advances in low-cost off-the-shelf computing components have made cluster computing an attractive way to achieve high-performance[3]. To provide supercomputer-class performance, the components are linked with powerful communication subsystems like Myricom Myrinet[2] and Tandem ServerNet[4]. However, the simple increase in the performance of the communication subsystems does not satisfy many cluster applications which require quality of service guarantees such as reliability, performance predictability and availability[5]. This paper focuses on performance predictability since it is essential to applications that require guaranteed service[5].

Performance predictability in networks is achieved through careful management of network resources. Existing resource management strategies can be classified into three categories: virtual circuits[6], physical circuits[4](e.g., in Tandem ServerNet) and software-based global scheduling[1] (e.g., in FM-QoS in Myrinet). The basic idea of the virtual circuit approach is to virtualize every end-to-end communication stream and to manage network resources at the granularity of such streams. The fine granularity of this approach enables highly flexible synchronization schemes, serving a wide variety of application classes[4]. However, the management cost of virtual circuits makes its usability questionable in a large-scale system.

The physical circuit approach applied in Tandem ServerNet manages network resources at the granularity of physical link of a switch. This approach allocates different amount of output link bandwidth to each input link. The coarser granularity of management reduces implementation complexity, but it fails to control the resource coupling among end-to-end streams that are multiplexed over a single physical link, thereby limiting the deliverable network performance guarantees.

While the above two approaches augment switch (or router) hardware with a resource management mechanism, the software-based global scheduling approach uses unmodified network hardware with global scheduler software which reserves a set of time slots for each end-to-end communication stream at connection setup. The advantage of this approach is that it does not require any change in switch hardware. This approach, however, poses two challenges: the first is building the global scheduler and the second is enforcing the schedules it generates. While the global scheduler can easily be built on the basis of existing reservation protocols[5], the enforcement mechanism remains a significant challenge. The difficulty with an enforcement mechanism is due to potential asynchrony between the local clocks of the network interfaces. In this scenario, even correct schedules may fail to deliver predictable performance due to unexpected resource conflicts between communication streams. In this paper we address the problem of maintaining a global time base among multiple network interfaces.

Depending on the level at which synchronization activity is performed, synchronization can be hardware-based[7] or software-based[8]. While exchanging clock tick information at a software level may incur large synchronization inaccuracies with software overhead, providing a single time base at the hardware level may be too costly. In order to avoid the disadvantages of both methods, feedback-based synchronization(FBS) in FM-QoS[1] was designed at the firmware level, which does not require a change in network interface hardware. Consider two network interfaces: one with a fast clock called NICf and the other with a slow clock called NICs. The basic idea of FBS is to force a packet from NICf to be blocked by another packet from NICs for a necessary amount of time, and to make the firmware of NICf detect the blockage via network flow control feedback. The firmware of NICf then pauses the local clock while its packet is blocked, and resumes the clock when the packet restarts. As a result, the fast local clock of NICf is synchronized with the slow clock of NICs by the interaction of the two packets. The goal of FBS is to force all the clocks in the system to slow to progress at the same rate as the slowest one.

The key challenge of FBS is to define the proper interaction among packets to ensure synchrony. The interaction can be expressed as a schedule to specify when and to whom each network interface sends a packet. FM-QoS calls the schedule a synchronizing schedule and exemplifies it for a single switch network. To provide a general FBS framework, this paper identifies requirements of synchronizing schedules for arbitrary network topologies. This paper also derives a synchronization skew model in terms of flow control parameters, such as routing delay and buffer length.

Based on the synchronizing schedule and the synchronization skew model, two network classes are examined : a single switch network and a multiple switch network. For each class, a synchronizing schedule is proposed and its bounded skew is analyzed. With the practical values for flow control parameters of Myrinet[2], we observe the bounded skew is less than 9.223 μsec for all experiments.

The rest of the paper is organized as follows: Section 2 describes the network model. Section 3 addresses the definition and requirements of the synchronizing schedule. Section 4 presents the synchronization skew model. In the context of Myrinet networks, Section 5 characterizes synchronization skew and overhead of FBS. Section 6 compares our work with others, and Section 7 summarizes this paper.

2. Background : Network Model

In this section, we describe wormhole networks as the assumed network model throughout the paper, because they are popular for system area networks due to their simplicity and cost-effectiveness[3,9].

2.1 Network Operation

A network is a set of network elements, each of which is either a network interface or a router[2]. Each network element is connected by a link, and has a small slack buffer associated with each input link. A network interface injects/consumes a packet to/from a network, and injected packets are transferred through routers. When a router receives the header of a packet, it assigns an available output link to the header. The packet is then transmitted through the established link in units of flits which are the smallest size of information in wormhole networks. If an output link is already in use by another packet, the whole packet stops along the partially established path. When the output link is released, the packet resumes transmission.

During blockage and resumption, link-level flow controls regulate data flow at upstream routers to prevent buffer overflow and underflow. The link-level flow control scheme, called stop and go flow control[2], operates based on the occupancy of each slack buffer, using STOP and GO flits: assuming the size of slack buffer is bl, 1) the STOP flit is sent to a sender element when the receiver buffer occupancy increases to ks(called the high watermark) to inhibit the sender from sending flits, and 2) the GO flit is sent to the sender when the occupancy drops to kg(called the low watermark) to allow the sender to resume sending ($bl \geq ks \geq kg \geq 0$). The parameters ks and kg are chosen so that the delays to propagate the STOP/GO flits do not cause overflow or underflow. Assuming the sender keeps sending flits at the maximum rate, bl-ks data flits are in transit over a link while the STOP flit is being transferred. Similarly, kg is the maximum number of flits that the receiver consumes after it issues the GO flit and before the resultant data flits arrive at the receiver. In stop and go flow control, when a packet header is blocked, the STOP flit is propagated up to the source network interface[1), keeping the source network interface from injecting remaining flits of the packet. After the network link is unblocked, the GO flit is propagated to the source network interface in the same manner, allowing it to resume injecting the packet.

2.2 Detailed Network Operation

This subsection explains the operation of a network element in detail. A network interface has a local clock, and its basic unit is a time slot. Network interfaces inject packets at scheduled time slots initiated by a header flit and terminated by a tail flit. The period over which network interfaces inject a flit is called a character period, cp. Therefore, time slot length is defined as cp multiplied by the number of flits per packet.

A router logically consists of three functional

1) This paper assumes that the packet length is long enough such that intermediate buffers along the path cannot accommodate a whole packet.

units. These units are a flow controller, a router arbiter and a crossbar[10]: a flow controller performs the stop and go flow control, a router arbiter assigns an available output link for an incoming header, and a crossbar transfers a flit through the assigned path. Because the router arbiter operates only for a header not for data flits, the times for header and data flits to traverse a router are different. Each of them is modeled as rd(routing delay) and sd(switching delay), respectively. Besides payload flits like a header and data flits, control flits such as STOP/GO flit block/issue operations at the previous network element after a certain delay. The delay is modeled by ld + 2 * fc, where ld (link delay) is the propagation delay of a single flit over a physical link, and fc(flow controller delay) is the delay for a flow controller to process the control flit. Note that the STOP/GO flits are processed by both flow controllers at a sender and a receiver side.

3. Feedback-based Synchronization

This section describes the basic concept of feedback-based synchronization(FBS) proposed in FM-QoS[1],and then generalizes it by introducing the notion of a synchronizing schedule. First, the requirements of synchronizing schedules are identified, and then two synchronizing schedules are proposed for two specific networks satisfying the requirements.

3.1 Basic Concept of Feedback-based Synchronization

Suppose there are two network interfaces NICs(with a slow clock) and NICf (with a fast clock) and when the local clock of NICf turns into time slot 2, NICs is still at time slot 1 as shown in Figure 1. Synchrony between NICf and NICs can be achieved by stalling the progression of time slot 2 of NICf until NICs is also ready to advance to time slot 2. Consider a packet A sent from NICs at time slot 1 and a packet B sent from NICf at time slot 2. Assume the two packets are destined to the same destination network interface NICd and link L is an input link of NICd. If the skew of the two clocks is not more than one time slot long and hence packet A acquires link L earlier, then packet B should be blocked until packet A releases link L at the end of time slot 1. Blocking of packet B stalls NICf from injecting remaining flits of packet B, prolonging the corresponding time slot. As a result, NICf 's clock is synchronized with NICs 's clock. This synchronization is made possible by stop and go flow control where the STOP and GO flit are propagated from NICd to NICf to stop and resume packet B, respectively.

The goal of FBS is to slow down all the network interfaces to the slowest one using the above technique. A communication pattern(a series of packets) which has this effect is called synchronizing schedule. A synchronizing schedule specifies when and to whom each network interface sends a packet. The interaction of these packets should stall all interfaces local clocks to match the slowest interface's clock. In the example of Figure 1, the synchronizing schedule is that NICs injects a packet to NICd at time slot 1 and NICf injects another packet to NICd at time slot 2.

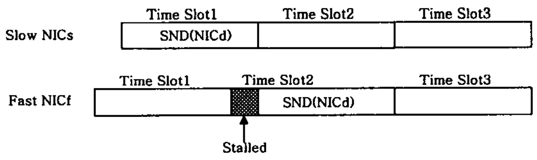


Figure 1 An example where fast NICf is slowed down to slow NICs

3.2 Synchronizing Schedule of Feedback-based Synchronization

In order for a synchronizing schedule to achieve a synchrony, the schedule should make a packet from every fast network interface be blocked by another one from the slowest network interface. We call this the dependency requirement. Besides the dependency requirement, the synchronizing schedule should not allow multiple network interfaces to access the same link at the same time slot. Otherwise, some of the multiple interfaces would be stalled even if they have the same time base. We term this property the conflict-free requirement. To describe the two requirements clearly, following notation is adopted.

- U : a set of network interfaces,
- M : a set of messages,
- s, f, d, kj (for some integer j) : some network interfaces in U,
- ts : some time slot,
- $s \Rightarrow_{ts} d$: a message which is injected by NICs and destined to NICd at time slot ts,
- $s \rightarrow_d^M f$ (NICs directly precedes NICf with respect to M) :
  $s \rightarrow_d^M f$ holds if M contains $s \Rightarrow_{ts} d$ and $f \Rightarrow_{ts+1} d$. TIME($s \rightarrow_d^M f$) is defined by ts and $\rightarrow_d^M$ is called a direct precedence relation. If obvious, M is omitted.
- $s \rightarrow_i^M f$ (NICs indirectly precedes NICf with respect to M) :
  $s \rightarrow_i^M f$ holds if M contains $s \rightarrow_d^M k1$, $k1 \rightarrow_d^M k2$, ..., $ki \rightarrow_d^M f$ such that TIME($s \rightarrow_d^M k1$) < TIME($k1 \rightarrow_d^M k2$) < ... < TIME($ki \rightarrow_d^M f$). $\rightarrow_i^M$ is called an indirect precedence relation and M is omitted if obvious.
- $s \rightarrow^M f$ (NICs precedes NICf with respect to M) :
  $s \rightarrow^M f$ holds if $s \rightarrow_d^M f$ or $s \rightarrow_i^M f$. $\rightarrow^M$ is called a precedence relation and M is omitted if obvious.

Consider NICs (with a slow clock) and NICf (with a fast clock) in U. The $s \rightarrow_d f$ implies that NICf is slowed down to NICs by a direct interaction of two packets injected from NICs and NICf, as in the example of Figure 1. Meanwhile, the $s \rightarrow_i f$ corresponds to the case where there are other network interfaces NICk1, NICk2..., NICki in U, and NICk1 is directly slowed down by NICs, then NICkj is slowed down by NICk(j-1), subsequently(2<=j<=i), and finally NICf is slowed down by NICki. In either case, NICf is synchronized to NICs and it is represented as $s \rightarrow f$. With the above notation, a synchronizing schedule is defined as a set of messages, M, which satisfies both the dependency requirement and conflict-free requirement.

Requirement 1 : Dependency requirement

A set of messages M satisfies the dependency requirement if $\forall$ s, f($\neq$s) $\in$ U, $s \rightarrow^M f$ where U is a set of all network interfaces in a network.

Because any network interface can be the slowest one at any given time, a synchronizing schedule should make any network interface precede all others.

Requirement 2 : Conflict-free requirement

A set of messages M satisfies the conflict-free requirement if for any two messages of $s1 \Rightarrow_{ts} d1$ and $s2(\neq s1) \Rightarrow_{ts} d2$ in M, a path from NICs1 to NICd1 and a path from NICs2 to NICd2 do not overlap.

The identification of requirements of a synchronizing schedule allows to find a synchronizing schedule in any network topology. Section 3.3 and Section 3.4 propose a synchronizing schedule for single switch networks and multiple switch networks, respectively.

3.3 Synchronizing Schedule for a Single Switch Network

This section first defines a building block communication pattern(BBP). Using BBP, a simple synchronizing schedule(SSS) is defined for a single switch network and a proof that SSS satisfies the two requirements is given. Finally, SSS is illustrated with an example.

When all elements in U are uniquely identified in some way and the ith element is represented by $idx_U(i)$ ($0 \leq i < |U|$), a BBP is defined by Definition 1 for some time slot ts. In this case, BBP is denoted by BBP(U, ts), and it is said to be applied to U at time slot ts.

Definition 1 BBP(U,ts)

BBP(U,ts) = { $s \Rightarrow_{ts+t} d$ | s= $idx_U(i)$, d= idx_U ((i + t(1+t)/2) mod |U|), $\forall$ 0 $\leq$ i,t $\leq$ |U|-1 }.

When BBP is applied to U at time slot ts, and U is a set of all network interfaces in a single switch network, BBP is called a simple synchronizing schedule(SSS) and is denoted by SSS(U,ts).

Theorem 1. *The SSS(U,ts) satisfies the conflict-free requirement and the dependency requirement.*

Due to the space limit, we omit the proof here. See Appendix A.1 for the proof. □

Lemma 1. *The BBP(U,ts) makes any two network interfaces in U directly precede each other, i.e., $\forall$ s,f($\neq$s) $\in$ U, $s \rightarrow_d^{BBP(U,ts)} f$.*

Due to the space limit, we omit the proof here. See Appendix A.2 for the proof. □

Example 1. *SSS ({NIC0, NIC1, NIC2, NIC3, NIC4, NIC5, NIC6, NIC7}, 0)*

When U={NIC0, NIC1, NIC2, NIC3, NIC4, NIC5, NIC6, NIC7} is the set of all network interfaces connected through a single switch and idx_U (i) =

NICi ($0 \leq i \leq 7$), Table 1 lists every messages of SSS(U,0) by time slots when messages are injected. For any network interface, e.g., NIC7, it is obvious that the following relations hold : NIC7 $\rightarrow_d$ NIC6, NIC7 $\rightarrow_d$ NIC5, NIC7 $\rightarrow_d$ NIC4, NIC7 $\rightarrow_d$ NIC3, NIC7 $\rightarrow_d$ NIC2, NIC7 $\rightarrow_d$ NIC1, NIC7 $\rightarrow_d$ NIC0 (refer to entries in boldface in the table). Since this is true for all interfaces in U, it follows that SSS fulfills the dependency requirement. SSS also satisfies the conflict-free requirement because each network interface injects a packet to a different destination at any given time slot, and packets in a single crossbar switch network destined for different destinations always use disjoint paths. □

3.4 Synchronizing Schedule for a Multiple Switch Network

Since any network topology can be converted into a spanning tree[11], we chose a tree as the base network topology in a multiple switch FBS framework. When network interfaces and switches are represented as leaves and nonterminal nodes in a tree, respectively, SSS does not satisfy the conflict-free requirement. For example, when schedule in Table 1 is executed in the tree network depicted in Figure 2, packets from NIC0 and NIC1 are destined to NIC4 and NIC5, respectively, at time slot 7, which results in a link conflict.

To avoid link conflicts, a hierarchical synchronizing schedule(HSS) is designed which employs a BBP in a hierarchical fashion. HSS achieves synchrony using two separate phases: 1) a clock gather phase in which only few selected network interfaces are synchronized to the slowest one, and 2) a clock distribute phase in which the selected network interfaces synchronized in the previous phase distribute their synchronized time base to all others. Assuming NIC7 is the slowest network interface in the tree of Figure 2, Example 2 describes how HSS propagates the slowest time base of NIC7 to all others, assuming that HSS starts at time slot 0. Table 2 lists every messages in HSS shown in Example 2, by time slots when they are generated.

Example 2. *HSS for a tree in Figure 2*

• Clock gather phase:

Table 1 Messages in SSS({NIC0, NIC1, NIC2, NIC3, NIC4, NIC5, NIC6, NIC7}, 0)

time slot								
0	$0 \Rightarrow_0 0$	$1 \Rightarrow_0 1$	$2 \Rightarrow_0 2$	$3 \Rightarrow_0 3$	$4 \Rightarrow_0 4$	$5 \Rightarrow_0 5$	$6 \Rightarrow_0 6$	$7 \Rightarrow_0 7$
1	$0 \Rightarrow_1 1$	$1 \Rightarrow_1 2$	$2 \Rightarrow_1 3$	$3 \Rightarrow_1 4$	$4 \Rightarrow_1 5$	$5 \Rightarrow_1 6$	$6 \Rightarrow_1 7$	$7 \Rightarrow_1 0$
2	$0 \Rightarrow_2 3$	$1 \Rightarrow_2 4$	$2 \Rightarrow_2 5$	$3 \Rightarrow_2 6$	$4 \Rightarrow_2 7$	$5 \Rightarrow_2 0$	$6 \Rightarrow_2 1$	$7 \Rightarrow_2 2$
3	$0 \Rightarrow_3 6$	$1 \Rightarrow_3 7$	$2 \Rightarrow_3 0$	$3 \Rightarrow_3 1$	$4 \Rightarrow_3 2$	$5 \Rightarrow_3 3$	$6 \Rightarrow_3 4$	$7 \Rightarrow_3 5$
4	$0 \Rightarrow_4 2$	$1 \Rightarrow_4 3$	$2 \Rightarrow_4 4$	$3 \Rightarrow_4 5$	$4 \Rightarrow_4 6$	$5 \Rightarrow_4 7$	$6 \Rightarrow_4 0$	$7 \Rightarrow_4 1$
5	$0 \Rightarrow_5 7$	$1 \Rightarrow_5 0$	$2 \Rightarrow_5 1$	$3 \Rightarrow_5 2$	$4 \Rightarrow_5 3$	$5 \Rightarrow_5 4$	$6 \Rightarrow_5 5$	$7 \Rightarrow_5 6$
6	$0 \Rightarrow_6 5$	$1 \Rightarrow_6 6$	$2 \Rightarrow_6 7$	$3 \Rightarrow_6 0$	$4 \Rightarrow_6 1$	$5 \Rightarrow_6 2$	$6 \Rightarrow_6 3$	$7 \Rightarrow_6 4$
7	$0 \Rightarrow_7 4$	$1 \Rightarrow_7 5$	$2 \Rightarrow_7 6$	$3 \Rightarrow_7 7$	$4 \Rightarrow_7 0$	$5 \Rightarrow_7 1$	$6 \Rightarrow_7 2$	$7 \Rightarrow_7 3$

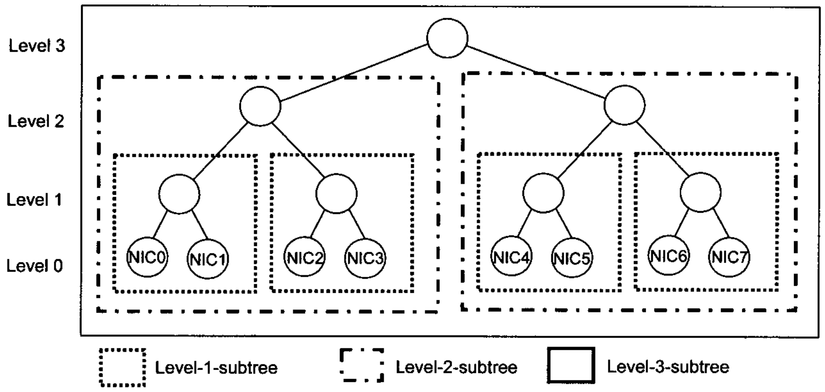


Figure 2 A tree with 4 levels

Table 2 Messages in HSS in Example 2

time slot								
0	$0 \Rightarrow_0 0$	$1 \Rightarrow_0 1$	$2 \Rightarrow_0 2$	$3 \Rightarrow_0 3$	$4 \Rightarrow_0 4$	$5 \Rightarrow_0 5$	$6 \Rightarrow_0 6$	$7 \Rightarrow_0 7$
1	$0 \Rightarrow_1 1$	$1 \Rightarrow_1 0$	$2 \Rightarrow_1 3$	$3 \Rightarrow_1 2$	$4 \Rightarrow_1 5$	$5 \Rightarrow_1 4$	$6 \Rightarrow_1 7$	$7 \Rightarrow_1 6$
2	$0 \Rightarrow_2 0$		$2 \Rightarrow_2 2$		$4 \Rightarrow_2 4$		$6 \Rightarrow_2 6$	
3	$0 \Rightarrow_3 2$		$2 \Rightarrow_3 0$		$4 \Rightarrow_3 6$		$6 \Rightarrow_3 4$	
4	$0 \Rightarrow_4 0$				$4 \Rightarrow_4 4$			
5	$0 \Rightarrow_5 4$				$4 \Rightarrow_5 0$			
6	$0 \Rightarrow_6 0$		$2 \Rightarrow_6 2$		$4 \Rightarrow_6 5$		$6 \Rightarrow_6 6$	
7	$0 \Rightarrow_7 2$		$2 \Rightarrow_7 0$		$4 \Rightarrow_7 6$		$6 \Rightarrow_7 4$	
8	$0 \Rightarrow_8 0$	$1 \Rightarrow_8 1$	$2 \Rightarrow_8 2$	$3 \Rightarrow_8 3$	$4 \Rightarrow_8 4$	$5 \Rightarrow_8 5$	$6 \Rightarrow_8 6$	$7 \Rightarrow_8 7$
9	$0 \Rightarrow_9 1$	$1 \Rightarrow_9 0$	$2 \Rightarrow_9 3$	$3 \Rightarrow_9 2$	$4 \Rightarrow_9 5$	$5 \Rightarrow_9 4$	$6 \Rightarrow_9 7$	$7 \Rightarrow_9 6$

- 1st step : define a level-1-subtree as a subtree with a root at level 1 where level 0 is the bottommost level. At time slot 0 and 1, four BBPs are concurrently applied to the leaves of each level-1-subtree. (NIC0 and NIC1, NIC2 and NIC3, NIC4 and NIC5, and NIC6 and NIC7). By Lemma 1, two network interfaces to which BBP is applied directly precede each other. The BBP applied to NIC6 and NIC7 induces NIC7 $\rightarrow_d$ NIC6.
- 2nd step: define a level-2-subtree as a subtree with a root at level 2. When a level-2-subtree contains two level-1-subtrees as in Figure 2, leaders of the level-2-subtree are defined as the network interfaces with the smallest index in each level-1-subtree. Two BBPs are applied to the leaders of each level-2-subtree at time slot 2: one to NIC0 and NIC2 and the other to NIC4 and NIC6. From the BBP applied to NIC4 and NIC6, NIC6 $\rightarrow_d$ NIC4 is achieved.
- 3rd step: define a level-3-subtree and its leaders similarly. At time slot 4 and 5, a BBP is applied to leaders NIC0 and NIC4, and therefore, NIC4 $\rightarrow_d$ NIC0. It subsequently induces NIC_7 $\rightarrow_i$ NIC_0 along with NIC7 $\rightarrow_d$ NIC6 (from the 1st step) and NIC_6 $\rightarrow_d$ NIC_4 (from the 2nd step).

By the clock gather phase which occurs from time slot 0 to time slot 5, NIC0 and NIC4 are synchronized to NIC7. In the clock distribute phase to occur from time slot 6, all others are indirectly synchronized to NIC7 via NIC0 and NIC4.

- Clock distribute phase
  - 1st step : at time slot 6 and 7, two BBPs are applied to NIC0 and NIC2, and to NIC4 an NIC6 in the same manner at the 2nd step in the clock gather phase. Therefore, NIC0 $\rightarrow_d$ NIC2.
  - 2nd step : at time slot 8 and 9, four SSSs are applied to NIC0 and NIC1, to NIC2 and NIC3, NIC4 and NIC5, and to NIC6 and NIC7 . Therefore, NIC0 $\rightarrow_d$ NIC1, NIC2 $\rightarrow_d$ NIC3, and NIC4 $\rightarrow_d$ NIC5. As a result, NIC7 precedes all other network interfaces either directly or indirectly. □

HSS consists of two groups of BBPs : one set performed during the clock gather phase and the other set during the clock distribute phase. Both phases consist of multiple steps, (refer to $\bigcup_{1 \le i \le m-1}$ GATHER(T, U, ts) and $\bigcup_{1 \le i \le m-2}$ DISTRIBUTE (T, U, ts) in Definition 2). Each step consists of multiple BBPs each of which is concurrently applied to a disjoint subtree. BBP makes selected leaves of a disjoint subtree directly precede each other. To prevent BBP from causing link conflicts, the size of the set of selected network interfaces is limited. Denoting the set of selected interfaces in a disjoint subtree rooted at node r as Leaders$_r$, this requirement is expressed as $|\text{Leaders}_r| \le i_r$, where i_r is defined as the number of children of node r. For that purpose, only one network interface is selected from each subtree of node r, e.g., the one with the smallest index.

In order to find the slowest time base and distribute it throughout a network, the clock gather phase and clock distribute phase progress in the opposite directions. To avoid packet conflicts between multiple BBPs at different steps or phases, HSS serializes each step and phase without overlapping in time. For the serialization, BBPs at the next step start after the longest time to execute BBP elapses (refer to G(ts,i) and D(ts,i) in Definition 2). The clock distribute phase serializes steps in a similar fashion.

Supposing a spanning tree T with m levels is found for some network topology with a set of network interfaces U, when HSS synchronizes U from time slot ts, it is represented by HSS(T, U, ts) . Assuming each node n_j of T is indexed from 0 to |T|-1, let smallest(N) be the node with the smallest index in N where N is some set of nodes. Defining level(j) to be the set of all nodes at level j ($0 \le j \le m-1$), and subtree(r) to a subtree whose root is a child node of r , HSS(T,U,ts) is defined as in Definition 2.

Definition 2 HSS(T,U,ts)

$\mathrm{HSS}(T,U,ts) = \cup_{1 \le i \le m-1} \mathrm{GATHER}_i(T, U, ts) \cup_{1 \le i \le m-2} \mathrm{DISTRIBUTE}_i(T, U, ts)$

where

$\mathrm{GATHER}_i(T, U, ts) = \cup_{r \in level(i)} \mathrm{BBP}(\mathrm{Leaders}_r, G(ts, i))$ where

$\mathrm{Leaders}_r = \{ \mathrm{smallest}(\mathrm{subtree}(r) \cap U), \forall\ \mathrm{subtree}(r) \}$,

$$G(ts,i) = \begin{cases} ts & \text{if } i=1 \\ G(ts,i-1) + \mathrm{MAX}_{q \in level(i-1)} |\mathrm{Leaders}_q|, & \text{otherwise} \end{cases}$$

$\mathrm{DISTRIBUTE}_i(T,U,ts) = \cup_{r \in level(i)} \mathrm{BBP}(\mathrm{Leaders}_r, D(ts, i))$ where

$$D(ts, i) = \begin{cases} G(ts, m-1) + \mathrm{Leaders}_{root}|, & \text{if } i=m-2, \\ D(ts, i+1) + \mathrm{MAX}_{q \in level(i+1)} |\mathrm{Leaders}_q|, & \text{otherwise.} \end{cases}$$

Theorem 2. *HSS satisfies the dependency requirement and the conflict-free requirement.*

Due to the space limit, we omit the proof here. See Appendix A.3 for the proof. □

Section 3 defined a synchronizing schedule of SSS and HSS. Both make the slowest network interface either directly(SSS) or indirectly(HSS) precede all others.

4. Analysis of Synchronization Skew of FBS

This section examines the synchronization precision of FBS. In Section 4.1, the skew of the $\rightarrow_d$ relation is analyzed. Based on that analysis, skews of $\rightarrow_i$ and synchronizing schedules are drawn. In Section 4.2 and Section 4.3, the skew model is applied to SSS and HSS. Given the upper bound of the clock skew, the interval with which FBS needs to repeat is examined in Section 4.4 to maintain a desired synchronization precision.

4.1 Synchronization Skew of FBS

Consider $s \rightarrow_d f$ as depicted in Figure 1 where packet B is blocked by packet A at input link L of NICd. Here synchrony is achieved in three steps: 1) after receiving the STOP flit, NICf stalls injecting remaining flits of packet B; 2) after NICs finishes time slot 1, link L is released; and 3) after link L is released, NICf resumes to inject the remaining flits. Each step incurs a timing gap which might prevent perfect synchrony: 1) GAP_1 occurring after NICf starts time slot 2 but before it receives a STOP flit, 2) GAP_2 occurring after NICs injects a tail flit of packet A but before link L is freed, and 3) GAP_3 occurring after link L is available but before NICf receives a GO flit. The three timing gaps are calculated by Equations (1)-(3).

With stop and go flow control, NICf receives a STOP flit after all buffers along the path are filled. According to the model in Section 2.2, each buffer generates a STOP flit when its occupancy reaches ks. Then, the buffer at upstream nodes receives the STOP flit after it sends out bl flits. Therefore, NICf receives the STOP flit after it injects bl * p2 flits into the network where p2 is the number of intermediate routers between NICf and NICd. From these observations, GAP_1 is calculated as follows:

$$GAP_1 = bl * p2 * cp. \qquad (1)$$

GAP_2 corresponds to the time required for the tail flit of packet A to traverse intermediate routers and links from NICs to link L. When the tail flit enters a buffer associated with an intermediate link, the number of flits ahead in the buffer is between 0 and (ks-2). This is because the packet is assumed to be long enough. Hence, the tail flit is paused at the source when its corresponding header is blocked, and it is not routed through buffers whose occupancy is ks or more. The time for the tail flit to traverse a router is the sum of the waiting time to reach a head of a buffer and the switching delay for the tail flit. If p1 is the number of intermediate routers between NICs and NICd, GAP_2 is :

$$p1 * (ld + sd) \le GAP_2 \le p1 * (ld + sd * (ks-1)). \qquad (2)$$

While blocked, the header of packet B is waiting at the router immediately preceding NICd, called router r_p. GAP_3 corresponds to the time for packet B to set up a new path at router r_p ,and for the resulting GO flit to be propagated to NICf. To shrink the buffer occupancy to kg, router r_p takes rd+(bl-kg-1)*sd, while each (p2-1) intermediate router takes (bl-kg) *sd. After each buffer shrinks, the GO flit propagates to the previous network element which takes (ld + 2*fc). From these, GAP_3 is given as :

$$GAP_3 = rd + (bl-kg-1) * sd + (p2-1) * (bl-kg) *sd + p2* (ld+2*fc) \quad (3)$$
$$= rd + (p2 * (bl-kg)-1) *sd + p2 * (ld + 2 * fc).$$

Since GAP_2 and GAP_3 slow NICf down and GAP_1 makes NICf faster, NICs and NICf have a synchronization skew of GAP_2 + GAP_3 - GAP_1 at the point when NICs and NICf complete time slot 2(refer to Figure 3). This is called skew of $s \rightarrow_d f$ and denoted by skew $(s \rightarrow_d f)$ with the following definition:

$$\text{skew } (s \rightarrow_d f) = | GAP_2 + GAP_3 - GAP_1 |. \quad (4)$$

From the boundary values of GAP_1, GAP_2 and GAP_3 in Equation(1)-(3), the bounded value of skew $(s \rightarrow_d f)$ is given as follows :

$$\text{skew } (s \rightarrow_d f) \leq MAX (|GAP_{min}(p1,p2)|, |GAP_{max}(p1,p2)|),$$

where

$$\begin{cases} GAP_{min}(p1,p2) = rd+sd*\{p1+p2*(bl-kg)-1\}+ld* (p1+p2) +2fc*p2-bl*p2*cp, & (5) \\ GAP_{max}(p1,p2) = rd+sd*\{p1*(ks-1)+p2*(bl-kg)-1\} +ld*(p1+p2)+2*fc*p2-bl*p2*cp. \end{cases}$$

For an indirect precedence relation of $s \rightarrow_i f$, the skew is the sum of skews of each $\rightarrow_d$ relation which constitutes $s \rightarrow_i f$ as follows :

$$\text{skew } (s \rightarrow_i f) = \text{skew } (s \rightarrow_d k1) + \Sigma_{1 \leq j \leq i-1} \text{skew}(kj \rightarrow_d k(j+1)) + \text{skew}(ki \rightarrow_d f), \quad (6)$$

if there exist $s \rightarrow_d k1$, $k1 \rightarrow_d k2$, ..., $ki \rightarrow_d f$ where $TIME(s \rightarrow_d k1) < TIME(k1 \rightarrow_d k2) <,...,< TIME(ki \rightarrow_d f)$. When FBS synchronizes a network with a synchronizing schedule of S*, the skew of the schedule S* is defined by the maximum skew of any precedence relation from S*.

$$\text{skew}(S*) = MAX_{\forall s \rightarrow^{s*} f} (\text{skew } (s \rightarrow f)). \quad (7)$$

4.2 Skew of SSS

Since $s \rightarrow_d^{SSS} f$ holds for all $s,f \in U$, skew(SSS) is bounded by Equation (5) with p1=p2=1. That is,

$$\text{skew(SSS)} \leq MAX(|GAP_{min}(1,1)|,|GAP_{max}(1,1)|) \quad (8)$$
$$= MAX (| rd + sd * (bl-kg) + 2 * ld + 2 * fc - bl * cp |, | rd + sd * (ks+bl-kg-2) + 2 * ld + 2* fc - bl * cp |).$$

We call the value in Equation (8) the bounded skew(SSS) and argue that SSS achieves the minimum bounded skew.

Theorem 3. For any synchronizing schedule S* which is applied in a single switch network, the bounded skew(S*) is not less than the bounded skew(SSS).

Due to the space limit, we omit the proof here. See Appendix A.4 for the complete proof. □

4.3 Skew of HSS

Consider a direct precedence relation induced at the i^{th} step of the clock gather phase($1 \leq i \leq m-1$), denoted as $s_g \rightarrow_d^{BBP(Leaders_{rg}, G(ts,i))} f_g$ for s_g, $f_g \in Leaders_{rg}$ and node n_{rg} at level i . Then, $s_g \rightarrow_d^{BBP(Leaders_{rg}, G(ts,i))} f_g$ is bounded by Equation (5) with $1 \leq p1, p2 \leq 2i-1$, because any two network interfaces in $Leaders_{rg}$ are separated by a maximum of 2i hops and a minimum of 2 hops. Since the proportional constant of p2 in Equation (5) can be either positive or negative depending on network flow control parameter values, the minimum value of GAP_{min}(p1, p2) is MIN(GAP_{min}(1, 1), GAP_{min}(1, 2i-1)). Similarly, the maximum value of GAP_{max}

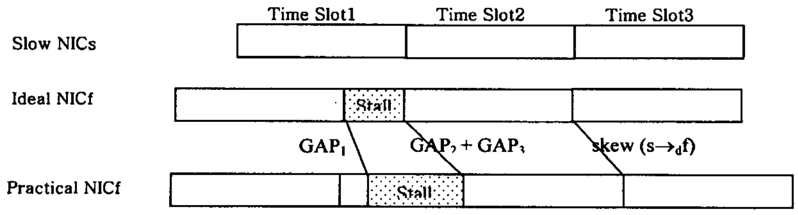


Figure 3 skew $(s \rightarrow_d f)$.

(p1,p2) is MAX(GAP_{max}(2i-1, 1), GAP_{max}(2i-1, 2i-1)). Therefore, the following inequality holds :

$$\mathrm{skew}(s_g \rightarrow_d^{BBP(Leaders\ rg,\ G(ts,i))} f_g) \leq \qquad (9)$$
$$\mathrm{MAX}(|\mathrm{MIN}(GAP_{min}(1,1),\ GAP_{min}(1,\ 2i-1))|,\ |\mathrm{MAX}(GAP_{max}(2i-1,1),\ GAP_{max}(2i-1,\ 2i-1))|).$$

Also consider a direct precedence relation induced at the ith step of the clock distribute phase denoted as $s_d \rightarrow_d^{BBP(Leaders\ rd,\ D(ts,i))'} f_d$ for s_d, $f_d \in Leaders_{rd}$ and node n_{rd} at level i ($1 \leq i \leq m-2$). Then, skew($s_d \rightarrow_d^{BBP(Leaders\ rd,\ D(ts,i))} f_d$) is bounded by the same value in Equation (9)

Since $s \rightarrow^{HSS} f$ incurs at most (m-1) $\rightarrow_d$ relations from BBP($Leaders_{rg}$, G(ts,i)) and m-2 $\rightarrow_d$ relations from BBP($Leaders_{rd}$, D(ts,i')) for node n_{rg} at level i ($1 \leq i \leq m-1$) and node n_{rd} at level I' ($1 \leq i' \leq m-2$), skew(HSS) is bounded by

$$\mathrm{skew}(HSS) \leq$$
$$\mathrm{MAX}(|\mathrm{MIN}(GAP_{min}(1,1),\ GAP_{min}(1,\ 2(m-1)-1))|,$$
$$|\mathrm{MAX}(GAP_{max}(2(m-1)-1,1),\ GAP_{max}(2(m-1)-1,\ 2(m-1)-1))|\ ) \qquad (10)$$
$$+2\sum_{1\leq i\leq m-2} \mathrm{MAX}\ (|\mathrm{MIN}(GAP_{min}\ (1,1),\ GAP_{min}(1,\ 2i-1))|,$$
$$|\mathrm{MAX}(GAP_{max}(2i-1,1),\ GAP_{max}(2i-1,\ 2i-1))|)$$

where m is the number of levels in the tree. We call the value in Equation (10) the bounded skew (HSS), and it grows as the depth of the tree increases. With that reason, it is preferred to minimize the depth of a used spanning tree.

4.4 Synchronization Interval

FBS requires a clock skew to be less than one time slot length at the point of synchronization. Otherwise, in the example of Figure 1, fast NICf would hold the link L earlier than slow NICs and synchrony would not be achieved. For that purpose, the skew of any network interface is always to be kept below the (1/2) * time slot length. Due to clock drift, t time slots accumulate the skew of (t * time slot length * drift rate). To ensure the accumulated skew plus synchronization skew remains below (1/2) * time slot length, FBS should be executed periodically. We call this period the synchronization interval, si. With a given synchronizing schedule of S*, si is calculated in units of time slots as follows:

$$si = \lfloor\ (\ (1/2) - \mathrm{skew}(S*)\ /\ \text{time slot length})\ /\ \text{drift rate}\ \rfloor\ . \qquad (11)$$

where time slot length is determined by the cp * (the number of flits in a packet) as defined in Section 2.1. The periodic execution of a synchronizing schedule consumes time slots which would otherwise be used for application communication streams. Given a synchronizing schedule of S*, the time overhead is given by Equation (12). For a fixed clock drift rate and time slot length, the time overhead will increase as either skew(S*) or the number of time slots to execute S* gets larger.

time overhead =
(the number of time slots to execute S*) / si =
drift rate *(the number of time slots to execute S*) / ((1/2)- (skew(S*)) /time slot length). (12)

5. Numerical Results of Synchronization Skew

5.1 Results of SSS

This subsection presents results of SSS in a single switch network. Unless otherwise noted, all results are measured with packet length of 2K bytes where the flit size is one byte[3] and with default parameters used in Myrinet-1280/SAN[2] (ld=17ns, cp =6.25ns, sd =2.0ns, rd =100ns, fc =3.26ns, bl =64Bytes, ks =53Bytes and kg=17Bytes). With the default parameters, the bounded synchronization skew is 237 nsec. Table 3 presents synchronization interval and time overhead varying the clock drift rates from 100ppm to 500ppm. Since the number of time slots to execute SSS is same as the switch size, time overhead grows linearly with the size of the switch.

Because FBS synchronizes network interfaces using the flow control signals, we examine the effect of rd, cp and bl on synchronization performance. The following experiments show the

Table 3 Analysis results with packet length=2K Bytes and the default parameters in [2]

Drift Rate	Synchronizing interval (in time slots)	Time overhead (%)		
		4-port switch	8-port switch	16-port switch
100ppm	4810	0.08	0.17	0.33
200ppm	2405	0.17	0.33	0.67
300ppm	1603	0.25	0.5	1.0
400ppm	1202	0.33	0.67	1.33
500ppm	962	0.42	0.83	1.66

synchronization performance while independently varying each flow control parameter. Figure 4 shows the behavior of bounded synchronization skew as rd increases from 50 nsec to 140 nsec. Interestingly, increasing rd from 50 nsec to 140 nsec reduces the synchronization skew from 298 nsec to 197 nsec. This is due to the characteristics of the default flow control parameters : $GAP_{min}(p1,p2)$ and $GAP_{max}(p1,p2)$ in Equation (5) have negative values as the result of the large value of bl * cp. A larger value of rd offsets the negative value to some extent, resulting in smaller synchronization skews.

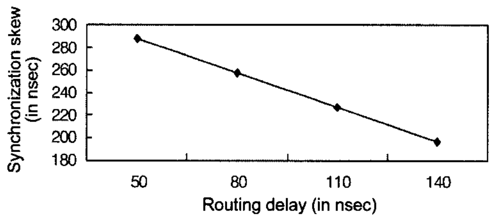
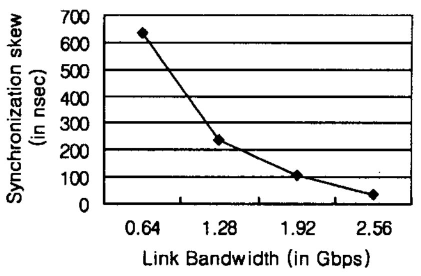


Figure 4 Synchronizing skew with various routing delay (rd) from 50nsec to 140nsec

The left one of Figure 5 illustrates the bounded synchronization skews as cp is decreased from 12.5 nsec to 3.13 nsec. As described in Section 2, cp is the amount of time required to inject a flit into the network. The link bandwidth and the value of cp vary inversely ; cp values of 12.5 nsec, 6.25 nsec, 4.17 nsec and 3.13 nsec correspond to the link bandwidths of 0.64Gbps, 1.28Gbps, 1.92Gbps and 2.56Gbps, respectively. Because the term of bl * cp dominates the synchronization skew, smaller values of cp (i.e., higher link bandwidth) results in smaller synchronization skews. This behavior implies that FBS is scalable in terms of the link bandwidth, since the synchronization precision of FBS improves with faster networks.Increasing the buffer length from 64 Bytes to 256 Bytes, the right one of Figure 5 illustrates that bounded synchronization skews grow from 237 nsec to 1.05 micro sec. The increase can be attributed to a deep buffer which prohibits flow control signals from propagating promptly, thereby limiting the synchronization precision.



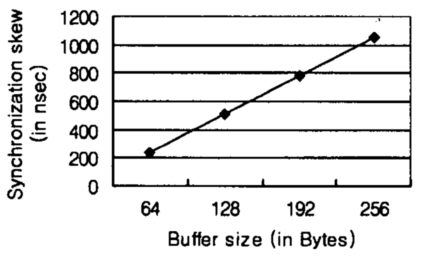


Figure 5 Synchronizing skew with varying link bandwidths from 0.64Gbps to 2.56Gbps, and buffer size from 64Bytes to 256Bytes.

From all the experiments performed with SSS, we observe that the FBS provides synchronization with the micro sec level precision, and the largest observed synchronization skew is 1.05 micro sec when the buffer length is 256Bytes.

5.2 Results of HSS

This subsection presents the analysis of HSS. For simplicity, we focus on a full tree. We assume that each nonterminal node of the tree, e.g. router, uses only one link to reach to the next higher level and all other links are used for other routers/ network interfaces at a lower level. If k-port switch is used, the tree contains up to $k * (k-1)^{m-2}$ hosts.

With the default parameters used in Section 5.1, the bounded synchronization skews are 1.42 micro sec and 4.02 micro sec for a tree with 3 levels and 4 levels, respectively. Using the parameters, Table 4 shows the synchronizing interval and the time overhead for a tree with 4 levels using 8-port switches. The number of time slots to execute the HSS, given by (the number of levels in a tree-2) * 2 * (the switch size-1)+1* (the switch size), is 36 time slots for such a tree. Comparing the experimental results with the SSS analysis in Section 5.1, the tables indicate that the synchronization cost

Table 4 Experimental results for a tree with 4 levels when the packet length of 2 Kbytes and the default parameters in [2] are used.

Drift rate	Synchronizing interval (in time slots)	Time overhead (%)	
		4-port switch	8-port switch
100ppm	1781	0.89	2.02
200ppm	890	1.79	4.04
300ppm	593	2.69	6.07
400ppm	445	3.59	8.08
500ppm	356	4.49	10.11

of HSS grows as the depth of the tree is increased. Even with the increased synchronization cost, however, at the drift rate of 300ppm which is found for most Myrinet networks, the largest time overhead is 6.07% with 4 levels. This is still impressive, considering those trees accommodate 392 hosts.

Figure 6 shows how bounded synchronization skew changes while varying link bandwidth from 0.64 Gbps to 2.56 Gbps. Figures 6 suggests the high link bandwidth yields the low synchronization skew. The largest synchronization skew is 9.2 micro sec with the link bandwidth of 640Mbps, which is also the largest observed synchronization skew over all experiments of SSS and HSS. It suggests that FBS achieves synchronization precision on the order of micro sec for a broad set of practical networks.

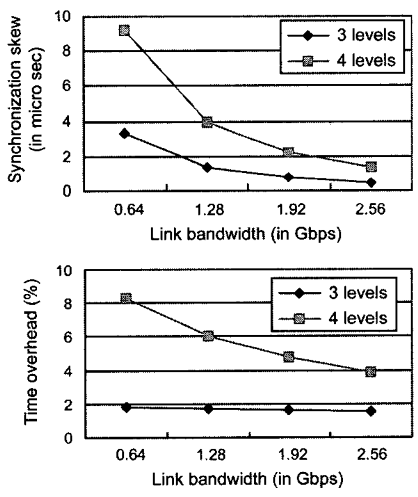


Figure 6 Synchronization skew with varying link bandwidth from 0.64Gbps to 2.56Gbps for 8×8 switches

6. Related Work

This section discusses previous efforts on synchronization. FM-QoS[1], Isotach Networks[12] synchronize multiple network interfaces of Myrinet networks. FM-QoS investigated the basic idea of FBS and illustrated its potential by implementing a prototype for a single Myrinet switch. However, FM-QoS is not enough to generalize the framework of FBS : it neither provided a synchronizing schedule for arbitrary network topologies, nor analyzed the synchronization skew.

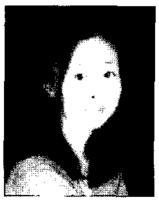
Isotach networks provide a single logical time base where each switch has a token manager to exchange tokens with all network interfaces. When the token manager receives token i from all the network interfaces, representing a logical point in time i, it sends token i +1 to all network interfaces in turn. The primary drawback of Isotach networks is that network resources cannot be fully utilized by application communication streams since the network interface serving as a token manager does not perform application communications. Using an Isotach network for an 8x8 Myrinet switch, 12.5% network bandwidth is wasted. This cost is significant, considering that the largest overhead of FBS is 0.83% as shown in Table 3 for an 8x8 Myrinet switch.

7. Conclusion

This paper addressed the synchronization problem of providing a global clock for multiple network interfaces in system are networks. The basic notion of FBS framework in FM-QoS[1] was generalized to arbitrary networks, the idea of synchronizing schedules were formalized and the skew of the FBS model was analyzed. Based on the generalization, two network cases were studied : a single switch network and a multiple switch network. For each of the networks, the simple synchronizing schedule(SSS) and the hierarchical synchronizing schedule(HSS) were proposed with their analyzed bounded skews. Given flow control parameter values, the analyses directly lead to the synchronization performance values. Using practical flow control parameter values in [2], we observed the synchronization skew of FBS is on the order of micro sec and its largest value is 9.223 micro sec. Since many applications require guaranteed service, especially multimedia applications which need performance predictability on the order of a micro-sec precision, the synchronization precision of FBS is sufficient.

References

[1] K. Connelly. FM -QoS : Real-time communications using self-synchronizing schedules. In Proceedings of SC'97, November 1997.
[2] Myrinet on VME protocol specification standard. VITA 26-199x Draft 1.1, August 1998. VITA Standards Organization.
[3] A. Chien et al. Design and evaluation of an HPVM-based Windows NT supercomputer. The International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS, 13(3):201--219, Fall 1999.
[4] R. Horst. TNet : A reliable system area network for I/O and IPC. In Proceedings of the IEEE Symposium on Hot Interconnects, 1994.
[5] I. Foster et al. A distributed resource management architecture that supports advance reservations and co-allocation. In Proceedings of the International Workshop on Quality of Service, 1999.
[6] Jon C.R. Benett and H. Zhang, Hierarchical packet fair queuing algorithms. In Proceedings of ACM SIGCOMM, August 1996.
[7] N. Vasanthavada and P. NN. Marinos. Synchronization of fault-tolerant clocks in the presence of malicious faults. IEEE Transactions on Computers, 37(4):440-448, 1998.
[8] F.Schmuck and F.Cristian. Continuous clock amortization need not affect the precision of a clock synchronization algorithm. In Proceedings of the ACM Symposium on Principles of Distributed Computing, pages 133-143, 1990.
[9] A. Chien and Others. HPVM software distributions, 1999. Available from http://www-csag.ucsd.edu/projects/hpvm/sw-distributions/index.html .
[10] J.H. Kim. Bandwidth and Latency Guarantees in Low-cost, High Performance Networks. PhD thesis, University of Illinois at Urbana-Champaign, January 1997.
[11] E. Horowitz and S.Sahni. Fundamental of Computer Algorithms. Computer Science Press, 1978.
[12] P. F. Reynolds, C.Williams Jr., and Jr. R.R. Wagner. Isotach networks. IEEE Transactions on Parallel and Distributed Systems, 8(4), 1997.



송 효 정
1988년~1992년 연세대학교 컴퓨터 과학과 졸업. 1992년~1994년 KAIST 전산과 졸업(석사). 1994년~1998년 KAIST 전산과 졸업(박사). 1998년~2000년 UC San Diego PostDoc