

“Time-driven Priority” Flow Control for Real-time Heterogeneous Internetworking

Yoram Ofek and Chung-Sheng Li

IBM T. J. Watson Research Center

P.O.Box 218, Yorktown Heights, NY 10598

e-mail: ofek,csli@watson.ibm.com

Moti Yung

CertCo, 55 Broad St., NY 10004

e-mail: moti@certco.com

Abstract

*We consider real-time traffic in a heterogeneous internetworking environment with IP routers, MAC bridges, Hubs, Switched LANs etc. We assume that the current routing protocols remain unchanged. However, in this environment, in order to provide quality of service (QoS): bandwidth, delay, constant-bounded jitter and no-loss due to congestion, we suggest a new flow control function called **time-driven priority**, which is an internal traffic shaping mechanism.*

We show how it supports two classes of connections: constant bit rate (CBR) with deterministic guarantees, and variable bit rate (VBR) with statistical multiplexing. The mechanism does not require to identify and separate the packet flows of different real-time sessions/connections inside the network. As a result, it achieves lower switching complexity when compared with other internal traffic shaping methods. As consequences of the time-driven priority mechanism we further achieve: (1) QoS parameters which are independent of the connection bandwidth, (2) QoS parameters which are independent of the existing heterogeneous internetworking asynchronous data traffic, and (3) the capability for policing and securing the network QoS.

Published In: IEEE INFOCOM'96, March 1996.

1 Introduction

Our objective is to enhance the performance of real-time sessions or connections in a heterogeneous internetworking environment. The motivation is the assumption that the current asynchronous (“best effort”) data services on such networks may not be sufficient for interactive real-time (e.g., voice and video) services. This may be proven to be a real concern, see for example, a recent report on an “*Internet hiccup*”¹.

Similar objective, but on a smaller scale, is behind *IsoEthernet*, which has been standardized by the IEEE 802.9a [1]. IsoEthernet combines CSMA/CD (IEEE 802.3) with N-ISDN and H.320 over existing Ethernet infrastructure. In a similar fashion, integrated multimedia services on the *Internet* and *switched legacy LANs* require the development of new **network flow control** and **scheduling** mechanisms so that in real-time services, quality of service (QoS) parameters are guaranteed by the network.

We introduce the *time-driven priority* function for flow control and *shaping* of real-time traffic inside the network. It is constructed by three basic components: (i) inter-switch synchronization, (ii) preemptive priority scheme and (iii) packet forwarding scheme. It facilitates real-time connections with both **deterministic** and **probabilistic** QoS guarantees, and supports constant bit rate (CBR) and variable bit rate (VBR) packet streams.

The mechanism is part of the physical and link layers. Thus, its operation is *transparent* to the network layer routing protocols, mechanisms for “best effort” (non real-time) traffic, and other protocols in higher layers. Inside the network, time-driven priority does not need to separate the real-time connections into different queues with per packet scheduling which is a function of the connection rate. Therefore, our solution has lower switching and buffering complexity compared to other internal shaping methods.

In heterogeneous internetworking arbitrary amount of *fixed delay* is required by every switch for *protocol processing in software*. The forwarding and synchronization embedded in our solution imply that adding such predefined constant delays can be done without increasing the end-to-end delay jitter.

Time-driven priority has two unique consequences for the network QoS:

(1) **bandwidth independent QoS**, namely, the delay, loss and jitter parameters are independent of the bandwidth assigned to each connection and the link utilization, and

¹“The Internet was hit by a “brownout” that delayed Internet connections on 9/5/95. The “brownout” may prove to be a harbinger of problems to come.” **Computerworld**, Sep 11, 1995, page 1.

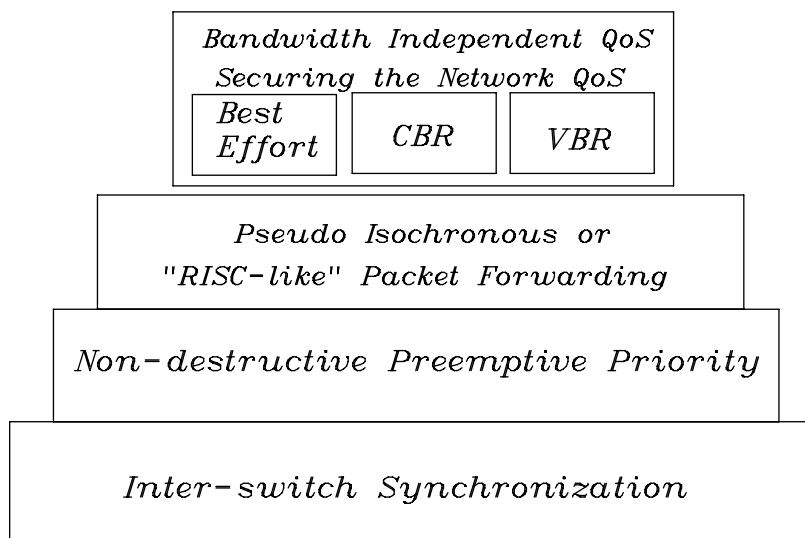


Figure 1: Time-driven Priority Componentets

(2) **securing the network QoS**, namely, (i) continuous protection and policing of network resources against misbehaving users and/or faulty applications, and (ii) session/connection isolation from one another.

1.1 Related works

This paper follows the early work on the *Synchronous Optical Hypergraph* [12, 13]. This early work proposed a technique for voice integration, using (what we later called) *immediate forwarding* [11], with global synchronization [15], over the *Synchronous Optical Hypergraph*. There are key differences between the two works, one of them is that instead of forwarding packets/cells over *point-to-point edges*, in the *Synchronous Optical Hypergraph* the forwarding is performed over *hyper-edges*, which are *passive optical stars*.

In order to avoid confusion, in Appendix A we explain the differences between time-driven priority and stop-and-go queueing [7].

2 Time-driven Priority

The basic objective of the time-driven priority design is to satisfy real-time multimedia services, while minimizing the impact on existing internetworking “*best effort*” data services. Thus, in the context of this paper, there is no need to change the later. However, since *time-driven priority* separates in time between “*best effort*” and real-time traffic, it will be possible, in the future, to modify the later without affecting the former services developed in this paper.

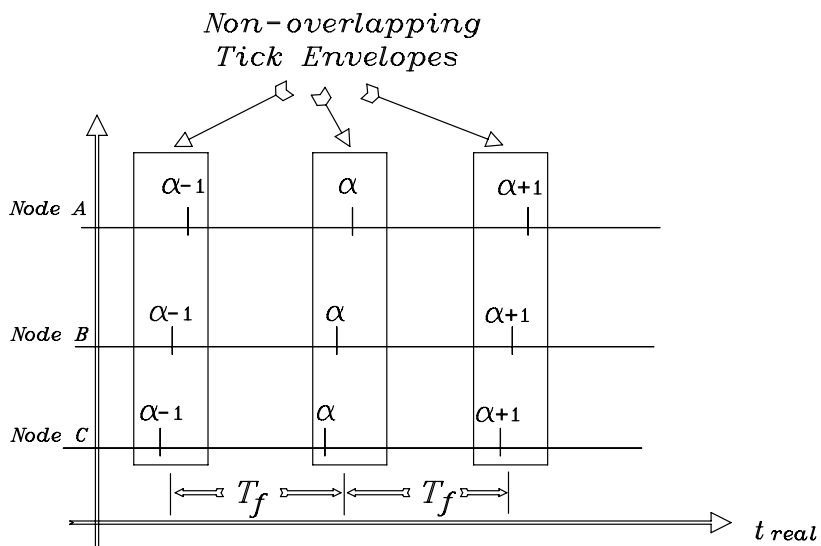


Figure 2: Inter-switch Synchronization Requirements

The idea of time-driven priority is to give higher priority to real time traffic in a *periodic fashion*, and only to packets which have arrived in the previous time interval. As a result, our design will provide the following properties for real-time traffic: (i) bounded delay - independent of the asynchronous data traffic, (ii) constant bound on the jitter, which is independent of the network size, and (iii) either *deterministic* no-loss or *probabilistic* control of the loss inside the network. For example, for real-time service it will be possible to ensure *deterministic loss-free due to congestion inside the network*. Moreover, this can be achieved under a full link utilization and without adversely affecting the QoS parameters.

The time-driven priority is constructed from three components, which are shown in Figure 1: (i) inter-switch synchronization, (ii) link layer non-destructive preemptive priority of real-time packets over, possibly long, data packets, and (iii) pseudo-isochronous or “RISC-like” forwarding of real-time packets in order to control the jitter and loss due to congestion inside the network.

2.1 Inter-switch Synchronization

The objective of the inter-switch synchronization is to provide the timing information needed for the time-driven priority, which is used for periodically forwarding real-time packets. As a result, it is possible to preserve the inter-packet delay between successive packets which belong to the same real-time connection.

2.1.1 Timing among neighboring nodes

The basic requirements for inter-switch synchronization are shown in Figure 2. Where we show three *neighboring nodes* A , B and C . The nodes *tick* or increment their copy of an abstract global clock every time unit which is called a *time frame*. In this example, the three nodes tick to $\alpha - 1$, then to α and then to $\alpha + 1$ and so on². We define the *tick envelope* as the maximum possible *phase difference* of the same tick in neighboring nodes, when it is projected on the real-time axis.

The inter-switch synchronization has two basic requirements:

R1: Tick envelope. All *neighboring nodes* should increment their clock within every *non-overlapping* tick envelope and to the *same value* which can be used as a *time stamp*.

R2: Time frame - T_f . It is defined in the network as the time interval between two successive tick envelopes. It has a well-defined nominal value, which can be computed in every node by measuring the time interval of a sufficiently large number of successive ticks [10]. In each node, the time frame is defined as the interval between two successive ticks.

There are several methods to achieve inter-switch synchronization. We characterize them as either *centralized* or *distributed*. In the centralized synchronization the nodes receive the timing information from one clock source. Possible solutions are timing distribution via satellites³, or to use the 8KHz clock provided by the public circuit-switched network.

Next, we briefly describe a distributed inter-switch synchronization method [14], which has the important property that the **slowest clock** determines the actual time frame interval, T_f . In other words, the local copies of the global clock on all the switching nodes are *frequency locked* to the slowest clock. This has the important consequence that if an external source clock is included in the system, such that, it is, by design, the slowest clock in the system, then it will determine the time frame interval in the entire system.

2.1.2 Distributed inter-switch synchronization

The global clock is generated from an ensemble of *high-speed local clocks*, which reside in the network's nodes. Let the rates of the n high-speed local clocks be represented by $\{c_1, c_2, \dots, c_n\}$ ⁴. The time is divided into discrete units referred to as *time frames*. Each node maintains a local *frame counter* (FC),

²The time values $\alpha - 1, \alpha, \dots$ are used inside the network for monitoring and securing the network operation.

³This can be done by using the global positioning satellite (GPS) system, in a similar manner to what has been reported in [4, 20, 2]

⁴We assume that the manufacturer's specification guarantees that any c_i is bounded below by $c_{min}^{nominal}$ (the slowest possible clock rate in the system) and bounded above by $c_{max}^{nominal}$ (the fastest possible clock rate in the system). We also

which is incremented at the start of each new time frame. This *increment operation* is called a *tick*. The local clock is used for measuring the duration of a time frame, which is *at least* K_f (in local clock cycles).

Definition 2.1 Maximum time frame: *The actual maximum time frame duration in a given network is*
 $T_{max} = K_f / c_{min}$.

The objective of the synchronization algorithm is that the slowest local clock in the network, the clock with c_{min} , will determine the global clock time interval to be exactly the maximum time frame duration - T_{max} .

Definition 2.2 Steady-state: *In a steady-state, the node with the slowest local clock ticks by measuring exactly K_f local clock cycles, and its time frame duration is T_{max} ; i.e., its time frame duration is not adjusted by the inter-switch synchronization algorithm.*

Note that on other nodes the tick can occur after more than K_f local clock cycles, but not less (see [14] for details).

2.1.3 The TICK control signal semantics

The steady-state algorithm which generates the inter-switch synchronization is based on exchanging control signals called TICKs. The TICK control signals are sent from each node, every time frame, **only to its neighbors**. The node that sends the TICK control signal indicates to the receiving node that the event of a tick on the sending node occurs immediately after it receives this signal; i.e., the sending node is “telling” the receiving node that “*immediately after this TICK signal is received the sending node will tick*”. The delays in this synchronization algorithm are computed such that **the sending node never ticks before its TICK signal gets to the receiving node**.

2.1.4 Timing properties

For the correctness and stability of the synchronization algorithm we should determine the bounds on the delay of the TICK control signal. Figure 3 illustrates the timing bounds on a TICK signal, which is sent from node i to node j , where t_{ij}^{max} and t_{ij}^{min} are the maximum and minimum delay from node i to node j , respectively. The requirements on TICK signal timing are summarized by the following two properties:

assume that $c_{max}^{nominal} = c_{min}^{nominal} (1 + \rho)$, where ρ is the constant clock drift given by the clock manufacturer. (ρ indicates the maximum possible deviation from $c_{min}^{nominal}$, which is the slowest possible clock.) In a given network, we represent the frequency of the *actual slowest clock* by c_{min} and its period $\tau_{max} = \frac{1}{c_{min}}$.

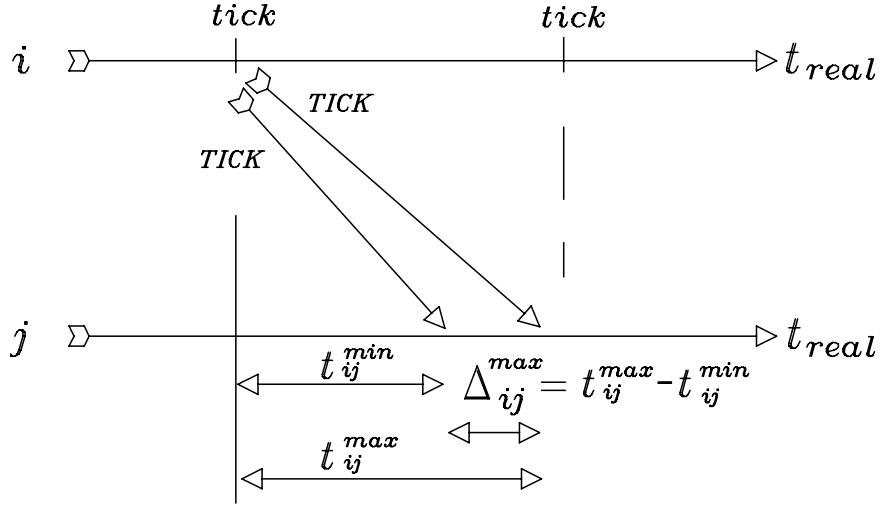


Figure 3: T_α and T_β Timing Properties

$$(T_\alpha) \Delta_{ij}^{max} = t_{ij}^{max} - t_{ij}^{min} < t_{ij}^{min}.$$

(T_β) TICK from node i to j arrives at node j always before node i ticks.

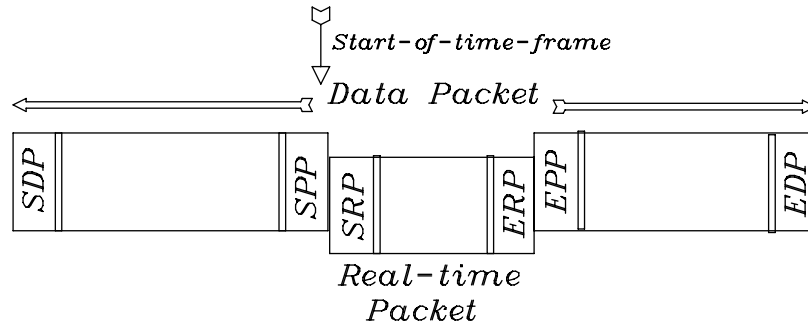
The reason behind the above two properties is to ensure that the TICK signals always travel faster than nominally expected. Thus, the node with the slowest clock will always TICK last, and will determine the time frame interval in the network [14].

2.2 Preemptive Priority

Time-driven priority gives real-time traffic higher priority at the beginning of every time frame. However since the asynchronous data services remain unchanged, thus it is possible that an output port will start transmitting a long data packet just before the beginning of a time frame. Therefore, at the start of a time frame the real-time packets should be able to preempt, in a nondestructive manner, the transmission of data packets, as shown in Figure 4.

Intuitively, preemptive priority “mimics”, to a large extent, the multiplexing of the small fixed size ATM cells. Therefore, preemptive priority is not needed when fixed size ATM cells are used. The time driven preemption provides the following features for internetworking with real-time communications:

- (i) Data packets can be large, few Kbytes, without adversely affecting the delay and jitter of the real-time traffic.
- (ii) It will be possible to increase the data packet size as the link bandwidth increases. By so doing the number of data packets processed by a switch or a router per unit time, remains constant.



SDP - Start Data Packet
SPP - Start Preempt Packet
SRP - Start Real-time Packet
ERP - End Real-time Packet
EPP - End Preempt Packet
EDP - End Data Packet

Figure 4: Real-time Preemptive Priority

2.3 “RISC-like” Forwarding

2.3.1 Basic forwarding principle

The real-time packet forwarding is based on the recently proposed [11, 16] “RISC-like” or **pseudo-isochronous** forwarding principle: “*a real-time packet advances one hop in every time frame*”. This mechanism, which is performed by all routers (or switches) at every time frame, can be viewed as **pipelining the real-time traffic** inside the network. This is analogous to pipeline execution of instructions, which is one of the central component of RISC machine architecture.

Figure 5 is a simple example of the “RISC-like” forwarding technique. In this example, at time $\alpha - 1$ node *A* forward a packet *P2* to node *B*, which will forward this packet to node *C* only at time frame α . In this example we show a successive transmission of five real-time packets: *P1*, *P2*, *P3*, *P4* and *P5*. Note that if at the beginning of a time frame the outgoing link is transmitting asynchronous data, then this transmission will be preempted by the real-time traffic in a non-destructive manner, as was shown in Figure 4.

We note that the packet position within a time frame is arbitrary, since timing information is not used for routing - unlike circuit switching with time division multiplexing (TDM). The real-time packets are routed in the same fashion as other non real-time data packets. The “RISC-like” forwarding principle can be viewed as a *compromise* between “pure” asynchronous packet switching and circuit switching.

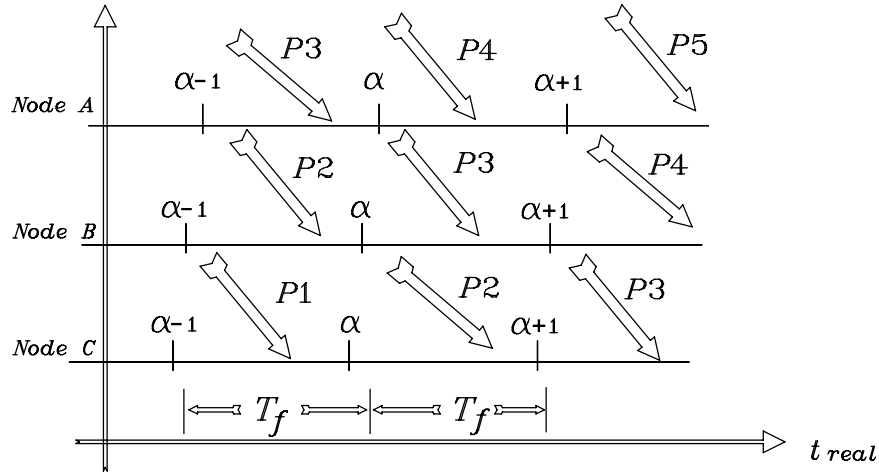


Figure 5: “RISC-like” Packet Forwarding

2.3.2 Generalized timing structure

The time is divided into frames of duration $T_f = T_{max}$. In each time frame one or more packets can be transmitted, depending on the serial transmission rate over the link. k time frames are grouped into a time cycle, T_c , such that, $T_c = k \cdot T_f$. The time frames in a cycle are numbered from 1 to k .

The transmission rate of a connection is fixed and is determined by the number of packets that can be sent in every time cycle. Thus, the smallest connection rate possible is sending one packet every time cycle. The packets of the same connection, from the source, are always sent in the same time frame within each time cycle.

2.3.3 Generalized forwarding principle

The basic “RISC-like” forwarding is generalized by the following two conditions:

Condition 1: All packets that should be sent in time frame i by a node are in the output port of that node before the beginning of time frame i .

Condition 2: The delay between an output port of one node and the output port of the next down-stream node is a **constant integer** multiple of T_f ⁵. The generalized forwarding principle is depicted in Figure 6.

The generalized forwarding principle is important for *protocol processing in software* in heterogeneous internetworking environment. In this case, a predefined but fixed number of time frames will be added to some intermediate switches. As a result, there will be an increase in the end-to-end delay, but the end-to-end delay jitter will remain constant.

⁵The time frame duration can be much smaller than the propagation delay between two nodes. A typical time frame value is $125\mu\text{sec}$ or 8000 time frames per second.

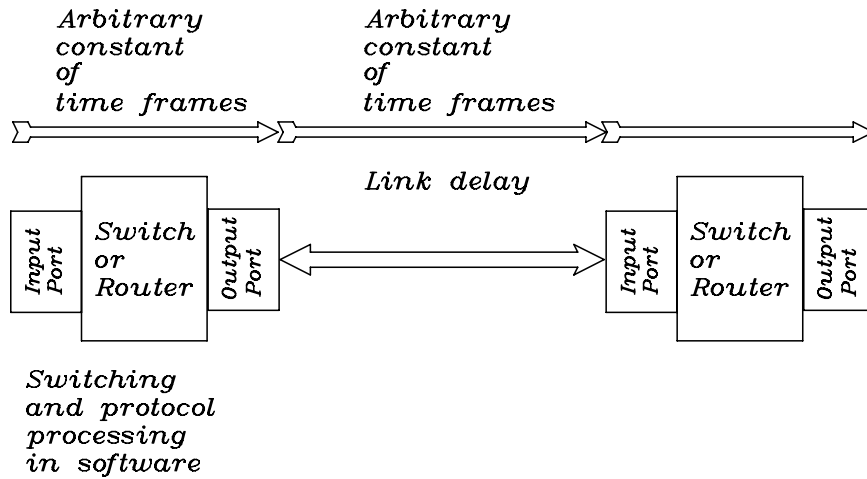


Figure 6: Generalized "RISC-like" Forwarding

2.3.4 Buffering requirement

The "RISC-like" forwarding operates in the same manner on the packets of real-time connections with different rates. In other words, all real-time packets are forwarded in the same fashion once they are inside the network. Therefore, the buffer requirement inside the network depends only on the time frame duration and the link bandwidth. For example, if $T_f = 125\mu\text{sec}$ and link bandwidth of 100Mb/s, then the total output buffer size per link for real-time traffic is about 1.6 Kbytes, and for link bandwidth of 1Gb/s the output buffer size per link is about 16 Kbytes.

2.3.5 Real-time multicast

The "RISC-like" forwarding can be used to support point-to-multipoint real-time multicast. When a multicast packet reaches an input port, it is duplicated to all the output ports used by the multicast tree on that node. Then each packet is forwarded to the next node as in the unicast case.

3 Quality of Service

Next we discuss what kind of quality of service (QoS) parameters can be provided for real-time applications. Intuitively, we can view the quality of service problem as an attempt to match between two sets of parameters: (i) the *network parameters*, such as: maximum packet size, switching policy, switch architecture, buffer size inside the network, time frame and cycle time intervals for the "RISC-like" forwarding, and (ii) the *application QoS requirements*, such as: loss, delay, jitter. The objective, of course, is that the network parameters will be able to satisfy the application QoS requirements (under a route assignment handled by resource manager, which we do not discuss here).

Note that if the network parameters do not satisfy the application QoS parameters, then it can result in adverse consequences, such as, (a) loss due to congestion inside the network, and (b) large jitter, which may be proportional to the network size (i.e., the jitter may grow with the network size). We will show that time-driven priority improves the assured quality of service parameters that can be provided by the network to the applications.

3.1 Deterministic Real-time Services

3.1.1 Bandwidth guarantees

Using the parameters defined in Section 2.3, the time is divided into *time frames*, T_f (in seconds), and k time frames are grouped into a time cycle, T_c (in seconds), such that, $T_c = k \cdot T_f$. The time frames in a cycle are numbered from 1 to k .

If for a real-time connection a packet with payload of b bytes is sent every time cycle its rate will be $8 \cdot b/T_c$. If the b bytes packet is sent every time frame its rate will be $8 \cdot b \cdot k/T_c$. For example, let $k = 64$ and $T_f = 125\mu\text{sec}$ ($T_c = 8\text{msec}$), then a connection with packets of size of 256 bytes which are sent at every time cycle will have a rate of 256 Kb/sec. If the 256 bytes packets are sent every time frame the connection rate will be 16.384 Mb/sec.

If the packet size should match the ATM cell size, 48 bytes, then we can select $k = 48$ and $T_f = 125\mu\text{sec}$. If packets of this size are sent every time cycle the connection rate will be 64Kb/sec. If we want to increase the rate we can send either *larger packets* or *multiple packets*.

In summary, we have a method for assigning bandwidth to each connection, as shown in the following simple formula:

$$Connection(\text{bandwidth}) = \frac{8 \cdot b}{k \cdot T_f} \quad (1)$$

3.1.2 Delay bound

The delay bound under the “RISC-like” forwarding scheme depends on the number of hops from the source s_1 to the destination s_h and the time frame, T_f . First, let’s assume that the propagation delay between two neighboring nodes and the delay from the input to the output of the switch is less than one time frame (the switching delay is discussed in Section 4.3.) Thus, the delay from source to destination (path length $h - 1$) is:

$$Connection(\text{delay}) = 2 \cdot (h - 1) \cdot T_f \quad (2)$$

However, in an existing *heterogeneous internetworking* environment, which consists of IP routers, MAC bridges, Hubs, Switched LANs etc., some of the switch or router functions may require more than one time frame. Typical operation that may need to be supported in software are address decoding and protocol conversion. Since the real-time traffic has a predictable pattern it is possible to compute, in advance during connection setup, the number of additional time frames that are needed for these operations, on every switch along the route. Let d_i be the *known constant of additional number of time frames* needed for the software operations on switch s_i . Thus, the heterogeneous internetworking delay (HI-delay) from source to destination (path length $h - 1$) will be:

$$Connection(HI - delay) = T_f(2 \cdot h - 2 + \sum_{i=1}^h d_i) \quad (3)$$

Note that this constant increase in delay does not change the “RISC-like” forwarding principle (as stated in the traffic pacing - condition 2, in Section 2.3) and will not affect the jitter, which we discuss next.

3.1.3 Jitter bound

Under the “RISC-like” forwarding scheme all packets of the same connection are forwarded in the same manner along the path from source to destination. If we assume that all real-time packets of a given connection have the same size, then the delay uncertainty or jitter is bounded by:

$$Connection(jitter) < 2 \cdot T_f, \quad (4)$$

independent of the number of hops and the network size.

3.2 Statistical Multiplexing of VBR

In this section we show how variable bit rate (VBR) sources with statistically varying rates, *around some average*, can be multiplexed in the network. Thus, the inter-switch synchronization not only improves the deterministic guarantees for constant bit rate services but also facilitates statistical guarantees of VBR sources. Moreover, the deterministic guarantees will not be adversely affected by the VBR traffic.

In this environment the packet rate of a connection over link L is not constant. We assume that during the connection’s/session’s setup phase the bandwidth of a session is allocated according to some measure, e.g., effective bandwidth, estimated by the user. The result of all setup procedures over link L is that in a given time frame, T_f^i , reservations from n sources were made for some capacities, $c_1^i, c_2^i, \dots, c_n^i$ (the sum of those capacities is smaller than link L capacity).

3.2.1 VBR with immediate discard

During run-time the packets which exceed the reserved average rate of its connection over link L in time frame T_f^i will be marked and identified as having lower priority. If not all incoming packets can be forwarded in the next time frame, as required by the “RISC-like” forwarding, the lower priority real-time packets will be discarded⁶. In the next subsection we show some strategies to smooth the VBR traffic by delaying the packet discard operation. “RISC-like” forwarding with the immediate discard operation ensures that at each point along the route the traffic of each connection has its *original source characteristics*. The only difference can be that some of the packets that were marked as low priority have been discarded.

Example: loss probability

The loss probability of the marked or low priority packets which are multiplexed in the network is a subject of further research. However, we would like to present the following perhaps intriguing example.

- Let connections c_A^i and c_B^i interact over link L at time frame T_f^i .
- Let the statistical characteristics of the two traffic sources be as follow: average rate is 1 packet in each time frame and the actual rate of a connection in time each time frame is 0, 1 or 2 packets with equal probability of $\frac{1}{3}$.
- Let’s compute how many packets actually arrive to link L at time frame T_f^i from these two connections:
 - 0 packet with probability $\frac{1}{9}$.
 - 1 packet with probability $\frac{2}{9}$.
 - 2 packets with probability $\frac{3}{9}$.
 - 3 packets with probability $\frac{2}{9}$.
 - 4 packets with probability $\frac{1}{9}$.
 - 5, 6, ... packets with probability 0.

The outcome is that the probability for receiving more than 4 packets is 0. Thus, we can control the accumulation of low priority packets inside the network. This is in contrast with packet switch network with asynchronous forwarding (with no internal traffic shaping) where there is non-zero probability to receive 5, 6, 7, and so on, packets in one time frame. Moreover, the probability increases as the network gets larger.

⁶This mechanism can be used to discard high frequency spectral components in *subband-coded* video.

3.2.2 VBR with non-immediate discard

As we discussed in the previous subsection, packets from variable bit rate sources which exceed the average guaranteed rate may be discarded inside the network. However, in some traffic scenarios the immediate packet discard may be too conservative. In the following we describe two methods for delaying the packet discard operation at the expense of larger delay and jitter. It is an open question, how the two methods affect the loss probability of the low priority packets.

Non-immediate with additional buffers:

The packet can be delayed in the switch by using additional buffers. On every switch a low priority packet can be delayed up to some constant number of time frames. If this number is, say, one time frame, then the delay and jitter bounds are increased by the time frame interval times the number of hops.

Non-immediate with a delay counter:

If we want to bound the jitter by a constant number of time frames, then we can include, in the header of each packet, a *count field*, which will be incremented each time the packet is delayed. Only when this *count field* reaches some predefined maximum value the packet will be discarded. In this case, this maximum value will determine the increase of the delay and jitter bounds.

4 Complexity Study

In this section, we investigate the complexity issues related to the (i) inter-switch synchronization, (ii) switch buffer complexity, and (iii) switch buffer access bandwidth. We show that by adding relatively little complexity for the inter-switch synchronization, it is possible to reduce the overall complexity of our switch design in comparison with some alternative internal traffic shaping approaches.

4.1 Inter-switch Synchronization

The inter-switch synchronization process is depicted in Figure 7. Inside a switching node there are usually three clock domains: (1) Receiver: each receiver is synchronized to the transmitter on the other end of the link with a phase-locked loop. (2) System: the entire switching system is synchronized to one high-speed local clock. (3) Transmitter: each transmitter is driven by a clock which may or may not be derived from the high-speed local clock.

The process of forwarding the TICK signals is done as follows. The TICK signals between each pair of network nodes can be embedded in the data transmission. On the receiving side (input port) the TICK signals are extracted from the physical layer and sent to the inter-switch synchronization controller, see

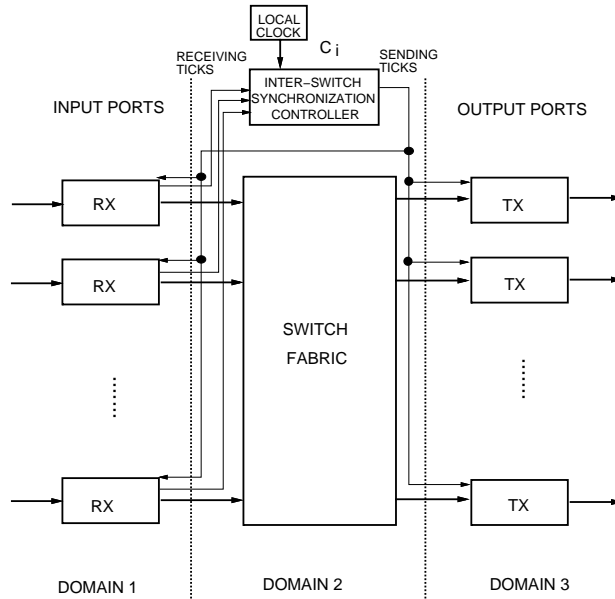


Figure 7: Division of Clock Domains

Figure 7. This controller executes the synchronization algorithm and generates the TICK signals, as was discussed in Section 2.1. The TICK signals are sent to neighboring nodes by combining them with the data transmission in the physical layer. The locally generated TICK signals are also used for the “RISC-like” forwarding inside the switch, from the input ports to the output ports, and from the output ports to the neighboring nodes, as was shown in Sections 2.2 and 2.3.

4.2 Switch Buffer Complexity

In this section we show that the “RISC-like” forwarding has complexity which is similar to the simple switching with a single FIFO buffer. In such a scheme the incoming packets from different connections are stored in a single queue and in their *arrival order*. We compare the single FIFO buffer with internal traffic shaping schemes which separate the incoming packets of different connections into multiple queues.

The complexity is measured in terms of the silicon area required to implement the flow control function in a *shared memory* switch architecture. In a shared memory, the single FIFO switching can be realized by having a single FIFO data structure for each output port, as shown in Figure 8(a), which can be maintained by a linked list data structure. In the FIFO approach, each new arrival is always added to the end of the list and only the head of queue can be removed from the list for delivery to the output port.

When flow control and shaping are performed inside the network without inter-switch synchronization, it can be done with or without deadline scheduling, see for example: [7, 21, 5, 6, 18, 3, 17, 9]. This is

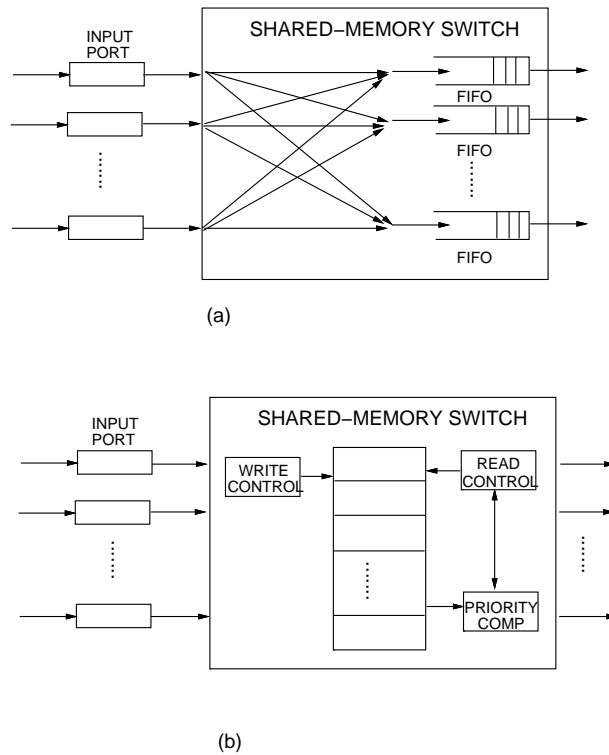


Figure 8: Comparison of Two Switch Architecture

a rather complex operation since scheduling and policing has to be performed per connection and per packet. A typical implementation employs a priority queue. For such a queue, implemented either by software pointers or with hardware assistance, it is usually necessary to ensure that packets are stored in the queue according to their connection rates, i.e., a newly arrived packet may not be put at the end of the queue. A shared memory switch which incorporates priority insertion mechanism is shown in Figure 8(b). In this switch each insertion due to a new packet involves significantly more operations than in the case of a single FIFO queue switch. As a result, this sort of implementation requires longer processing time and more hardware. The priority queue operation involves the following steps:

- (1) Extract connection information from each packet header.
- (2) Consult the scheduling table, which contains the scheduling information of the connections stored in the switch.
- (3) Determine the priority of the new packet (this can involve the recalculation of the priority of all the packets in the queue, and thus may cause a substantial overhead).
- (4) Determine the entry point given this packet priority. This is a searching operation which has logarithmic time complexity - measured by the size of the priority queue. This time complexity can have a prohibitive consequence in high-speed networks.
- (5) Insert the new packet into the appropriate position in the priority queue.

Since the dynamic calculation of priorities is in the critical data path, hardware assistance is usually

required in priority queue implementations. An implementation of a hardware-assisted virtual clock scheduler has been investigated in [3]. Other hardware-assisted VLSI implementation of priority queue has been reported, for example, in [19]. A rigorous comparison of the chip area required by a conventional single FIFO and a priority queue is reported in [19]. In this study, the priority queue is shown to require 66% more area than a conventional FIFO using 1.2 μm CMOS technology.

The “RISC-like” forwarding scheme does not require priority queue, and can be realized, for example, with one FIFO at each input port, and one FIFO at each output port. The input queue holds all arrivals until it receives the TICK signal from the inter-switch synchronization controller, as was shown in Figure 7, after which the packets are forwarded to the output queues. This operation for a *non-shared-memory switch* will be further discussed in the next subsection.

4.3 Switch Buffer Access Bandwidth

Another important measure of complexity in high-speed switch design is the bound on the access bandwidth to the switch output buffers (i.e., the *memory access bandwidth*). If we use shared memory switches, as depicted in Figure 8, the access bandwidth to the shared buffer is $2 \cdot P \cdot L_{BW}$, where P is the number of input/output ports and L_{BW} is the link bandwidth. Since the shared memory should be accessed simultaneously by the input and output ports, a factor 2 is included.

“RISC-like” forwarding, on the other hand, has a predictable pattern in time. Thus, the switch design can be simplified so that the switch buffer access bandwidth is bounded by a constant - $2 \cdot L_{BW}$. This gain becomes more important as the internetworking link bandwidth increases to the range of multi-gigabit per second. Next we explain why the constant bound of 2 is sufficient.

If the real-time switch operates according to the pacing and forwarding conditions (as was described in Section 2.3), then we have the following: **"RISC-like" forwarding switching property:** *during every time frame up to k real-time data units arrive at each input buffer, and during every time frame up to k real-time data units should be switched to each output port.*

If the real-time switch can transfer $2k$ real-time data units, in every time frame, from each input and to each output. Then using *Clos Theorem*, in the time domain (see [8] page 65), it is possible to ensure that in every time frame: (i) k real-time data units are switched from each input buffer, and (ii) k real-time data units are switched into each output buffer. This satisfies the "RISC-like" forwarding switching property. Thus, by having a staging buffer, with capacity of one time frame, at each input port, and by using the TICK signal to synchronize the forwarding from the input to output, it is possible to bound the input/output buffer access bandwidth to/from the switch by $2 \cdot L_{BW}$.

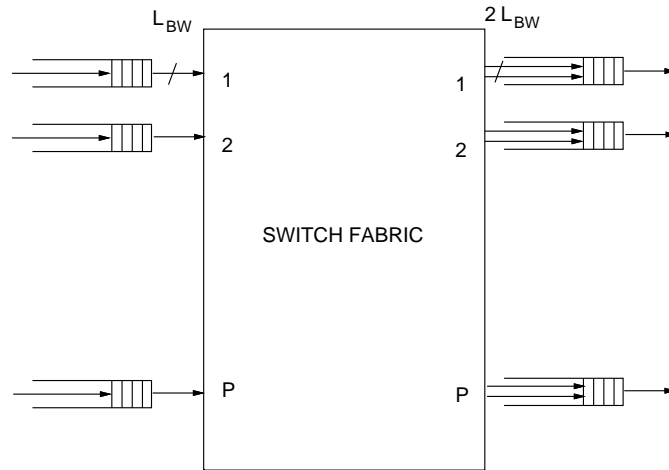


Figure 9: “RISC-like” Switch Architecture

5 Consequences

In this section we will show that in addition to reduced complexity of switching, there are unique consequences in using time-driven priority. The consequences are to the services provided to by the network to users and applications: (1) *bandwidth independent QoS*, (2) *securing the network QoS*, and (3) *transfer of large files with short latencies*.

5.1 Bandwidth Independent QoS

From the point of view of the network user and application scalability, it is natural that bandwidth and QoS requirements (delay, jitter, and loss) should be independent. In the following we first compare time-driven priority with traffic shaping that use only local timing (i.e., the time in each node is independent of the time on its neighboring nodes), and then compare it with circuit switching in which all nodes are synchronized.

5.1.1 Time-driven priority vs. shaping with local timing

There are two types of traffic shaping with local timing: without deadline scheduling, for example [7, 21, 5, 17], and with deadline scheduling, for example [6, 9]. In traffic shaping without deadline scheduling in order to obtain high utilization with no loss *the delay and jitter are inversely proportional to the connection bandwidth*, which means that low rate connections may experience large delay and jitter inside the network. In traffic shaping with deadline scheduling *the delay and jitter are controlled at the expense of possible congestion and loss*.

Time-driven priority, as was shown by Equations 1, 2, 3, and 4 in Section 3.1, provides loss-free (due

to congestion) packet delivery with delay and jitter, which are independent of the connection bandwidth and the link utilization. This is similar to circuit switching where the delay is independent of the number of channels the user requests, and the jitter is bounded by a constant. We next discuss the key differences between our approach and circuit switching.

5.1.2 Time-driven priority vs. circuit switching

In circuit switching, for each data unit (e.g., a byte) at the time it has been transmitted from its source, it is possible to predict deterministically the future times it will be transmitted from any switch along its route. The time resolution of this advanced knowledge is *much shorter than the data unit transmission time*.

On the other hand, in time-driven priority, for each data unit (e.g., a cell) at the time it has been transmitted from its source, we know the future time frames that this data unit will be forwarded along its route. However, the time frame, which constitute the accuracy of this advance timing knowledge, is *much larger than one data unit transmission time*. For example, the transmission time of an ATM cell (53 bytes) over a gigabit per second link is 424 nanoseconds, which are 294 times smaller than a typical time frame of $125\mu\text{sec}$ - used for time-driven priority.

There are several consequences which further differ our approach from circuit switching:

- In time-driven priority the synchronization requirements are independent of the physical link transmission speed, while in circuit switching the synchronization becomes more and more difficult as the link speed increases.
- In time-driven priority timing information is not used for routing, and therefore, in the Internet, for example, the routing is done using IP routing.
- In time-driven priority "*best effort*" strategy is possible. If a time slot within a time frame has not been fully utilized, then non real-time traffic can use this capacity.

5.2 Securing the Network QoS

In the following discussion we explain how the added timing information can enable traffic policing for securing the network QoS parameters. Intuitively, in circuit switching a source data is restricted and served in a very predictable form. In packet switching, on the other hand, as the timing uncertainty or jitter grows, the “global state” of the network is less defined. The main consequence is that it becomes more difficult to determine whether or not the system deviates from its expected operation mode. Moreover,

causal correlation of events cannot be done, since association of events to time is losing its meaning.

In time-driven priority, we can still assure authenticity of data source and measure source compliance with its requested service. We employ cryptographic tags computed at the source and attached to the data unit at a given time frame. Switches match the tags against tags precomputed at the switches. A match assures the source of a data unit and assures the well-behavior of sources. Fast cryptographic methods (stream ciphers) can be used for fast precomputations and computations of the authenticity tags (each time unit the next block of bits is the tag - only the source and the switches can compute the tag based on a shared key). Using the time framework it is easy to associate a data tag and a source with a precomputed tag at a previous (well determined) time frame (which is impossible if timing information is not available).

We can add various security mechanisms that rely on strict timing framework, examples include: (1) **Simple rate checking:** randomly monitor the number of packets and their sizes per time frame. Since the real-time traffic pattern is predefined the monitor can easily detect connections which exceed their specified rates. (2) **Simple delay checking:** randomly a pair of switches S_0, S_1 on the network agree that one will mark the packet and the other will check that the actual delay and the predefined delay are matched. This can be further employed to prevent long existence of *stray packet* and for *loop detection*.

5.3 Transfer of Large Files

There are increasingly growing number of applications which involve transfer of large files (tens and hundreds of megabytes). For example, transfer of images and video files over the *world wide web* (WWW). Performing such file transfers with existing internetworking “*best effort*” data services, may introduce large periods of congestion inside the network and may also, under some circumstances, overflow the receiver buffer at the host computer.

These two adverse phenomena can be solved by setting up a CBR connection and using time-driven priority. Such a connection will have a bandwidth which can be satisfied by the server, network and host receiver. Note that the connection setup may take tens of milliseconds while the actual transfer may take tens of seconds. Furthermore, it is possible reduce the amount of buffering needed at the host receiver while transferring a video file if we start playing the video before the whole file has been transferred.

References

- [1] IEEE 802.9a Editor. Integrated service (is): IEEE 802.9a isochronous services with CSMA/CD

- MAC service. *IEEE Draft*, March 1995.
- [2] E. W. Butterline, A. E. Abate, and G. P. Zampetti. Timing a national telecommunication network using GPS. *Proceedings of the 6th European Frequency and Time Forum*, pages 493–496, March 1992.
- [3] H. J. Chao. A novel architecture for queue management in the ATM network. *IEEE J. on Selected Areas in Comm.*, SAC-9(7):1110–1118, September 1991.
- [4] P. H. Dana. Global positioning system overview. <http://www.utexas.edu/depts/grg/gcraft/notes/gps/gps.html>, 1997.
- [5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM Computer Communication Review (SIGCOMM'89)*, pages 3–12, 1989.
- [6] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE J. on Selected Areas in Comm.*, SAC-8(4):368–379, April 1990.
- [7] S. J. Golestani. Congestion-free communication in high-speed packet networks. *IEEE Trans. on Comm.*, COM-39(12):1802–1812, December 1991.
- [8] J. Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1990.
- [9] D. D. Kandlur, K. G. Shin, and D. Ferrari. Real-time communication in multi-hop networks. *IEEE Trans. on Parallel and Distributed Systems*, 5(10):1044–1056, 1994.
- [10] C-S. Li and Y. Ofek. Distributed source-destination synchronization in ATM using inband clock distribution. *IEEE J. on Selected Areas in Comm.*, 1996.
- [11] C-S. Li, Y. Ofek, A. Segall, and K. Sohraby. Pseudo-isochronous cell switching in ATM networks. *IEEE INFOCOM'94*, pages 428–437, 1994.
- [12] Y. Ofek. The topology, algorithms and analysis of a synchronous optical hypergraph architecture. *Ph.D. Dissertation Electrical Engineering Department, University of Illinois at Urbana, Report No. UIUCDCS-R-87-1343*, May 1987.
- [13] Y. Ofek. Integration of voice communication on a synchronous optical hypergraph. *INFOCOM'88*, 1988.
- [14] Y. Ofek. Generating a fault tolerant global clock using high-speed control signals for the MetaNet architecture. *IEEE T. on Communications*, pages 2179–2188, May 1994.

- [15] Y. Ofek and M. Faiman. Distributed global event synchronization in a fiber optic hypergraph network. *The 7th Int. Conf. on Distributed Computing Systems*, pages 307–314, 1987.
- [16] Y. Ofek and M. Yung. The integrated MetaNet architecture: A switch-based multimedia LAN for parallel computing and real-time traffic. *IEEE INFOCOM'94*, pages 802–811, 1994.
- [17] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control - the multiple node case. *IEEE/ACM T. on Networking*, 2(2):137–150, 1994.
- [18] J. M. Peha and F. A. Tobagi. Evaluating scheduling algorithms for traffic with heterogenous performance objective. *GLOBECOM'90*, pages 21–27, 1990.
- [19] D. Picker and R. D. Fellman. A VLSI priority packet queue with inheritance and overwrite. *IEEE Trans. on Very Large Scale Integration Systems*, 3:245–253, 1995.
- [20] T. Piotrowski. Synchronization of telecommunication network using a global positioning satellite. *IEEE PLANS'92*, March 1992.
- [21] L. Zhang. VirtualClock: A new traffic control algorithm for packet-switch networks. *ACM Trans. on Computer Systems*, 9(2):101–124, May 1991.

Appendix A

Time-driven priority vs. stop-and-go queueing

1. Time frame definition.

Time-driven priority: The time frame interval is a global parameter which is completely independent of the connection rate (the only possible, but not necessary, restriction is that it should be able to contain at least one packet).

Stop-and-go: The time frame is **inversely proportional to the rate**, i.e., the time frame cannot be smaller than the inter-packet delay. This is stated in Golestani's paper [7] on page 1807 (Section 5 on Impact of Frame Size) 2nd column "*However, a small [time] frame size T can be chosen only at the cost of reduced flexibility in the bandwidth allocation, i.e., allocation of transmission rate to connections, which can be done in incremental steps of one packet per frame*". This is also an immediate outcome of the $(r, T) - smooth$ definition (Definition 2 on page 1803, [7]).

In time-driven priority there is no such restriction, as clearly stated in our paper see, for example, Eq. 1 in this paper. In this case, increasing k will decrease the connection rate. This is based on the *Time Cycle* concept, see Sections 3.1.1 and 2.3.2, in our paper. *In stop-and-go there is no time cycle.*

2. End-to-end delay and jitter.

One of the consequences of the capability to determine a very short time frame, say $10\mu\text{sec}$, is that it is possible to have a queueing delay per hop and end-to-end jitter of only $20\mu\text{sec}$! Furthermore, it is *independent of the connection rate and packet size*.

However, in stop-and-go if the payload in every (IP) packet, for example, is 320 bytes, and the connection rate is 64Kbit/sec, than the minimum delay per hop will be 80 msec - which is 4,000 times larger! A factor of 4,000 is a fundamental difference, indeed.

3. The variable bit discussion in time-driven priority is new with no comparable discussion in stop-and-go queueing.
4. The complexity discussion in time-driven priority is completely different in content and results than what is discussed in stop-and-go queueing.
5. Stop-and-go queueing does not utilize *preemptive priority* as was introduced in time-driven priority.
6. Another major difference with stop-and-go is what we called in an earlier paper non-immediate forwarding [11]. This provide more flexibility in scheduling connections and reduces, **from exponential to polynomial**, the **blocking probability** of new connections. This key result was described in [11].
7. Furthermore, Stop-and-go queueing does not even discuss the issue of scheduling and blocking probability, see [11].

These are indeed classical circuit switched issues. So the next question can be how time-driven priority differs from circuit switching, which was already discussed in the conclusions.