A Self-Timed Real-Time Sorting Network*
(in Proceedings of International Conference on Computer Design, October 1998)

Kenneth Y. Yun† Supratik Chakraborty‡ Kevin W. James†
Robert Fairlie-Cuninghame† Rene L. Cruz†

† ECE Dept., University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093
‡ EE Dept., Stanford University, Stanford, CA 94305

E-mail: `kyy@ucsd.edu`

Abstract

High speed networks are expected to carry traffic classes with diverse Quality of Service (QoS) guarantees. For efficient utilization of resources, sophisticated scheduling protocols are needed; however, these must be implemented without sacrificing the maximum possible bandwidth. This paper presents the architecture and implementation of a self-timed real-time sorting network *to be used in packet switches that support a diverse mix of traffic. The sorting network receives packets with appropriately assigned priorities, and schedules the packets for departure in a highest-priority-first manner. The circuit implementation uses zero-overhead, self-timed, self-precharging domino logic and timed-roadblock techniques to minimize the circuit latency. Experimental chips being built using the techniques described in this paper support 10 Gb/s links with ATM size packets.*

1 Introduction

The explosive growth in the network utilization in the past few years has triggered the development of a global high bandwidth network infrastructure. A challenging problem in the design of today's high speed networks is allowing prioritization of network access so that diverse Quality of Service (QoS) guarantees can be provided to users with different requirements. Central to this concept are the issues of: (i) assigning priorities to information packets, and (ii) sorting packets by their priorities at intermediate switches to determine the allocation of network resources. While the priority assignment problem has been addressed elsewhere [4, 2], there remains interesting new research in that direction. This paper addresses the second issue.

We describe a novel architecture for a high-throughput, low-latency 2×2 switch with *priority-based* conflict resolution and distributed buffering. The switch sorts conflicting packets, those destined for the same output port, according to priority levels. The packet with the highest priority is transmitted, and lower priority packets are stored for future transmission. Sorting is done in real-time, so that the network can be used in real-time networked applications, such as video-conferencing.

*This research was supported in part by a NSF CAREER Award MIP-9625034 and by a gift from Intel Corp.

Related work: Sorting networks using fast optical switches have been implemented in [5, 8]. However, these networks do not support priority-based sorting for QoS. In addition, viability of the optical switching techniques for immediate volume production is unclear. Priority-based hardware sorting algorithms have been presented elsewhere [7, 12]. These may be viewed as degenerate cases of our algorithm. Most of the related work comes from the ATM switching community. The work of [9] describes an ATM switch that supports sorting for QoS, but at a relatively low speed (500 Kb/s). A very fast electronic switch is discussed in [1], but priority-based sorting is not supported in it. [11, 13] describe different ATM switch architectures with prioritized servicing. Unfortunately, existing solutions do not support both high speed operation (e.g., 1 to 10 Gb/s) and sorting for QoS.

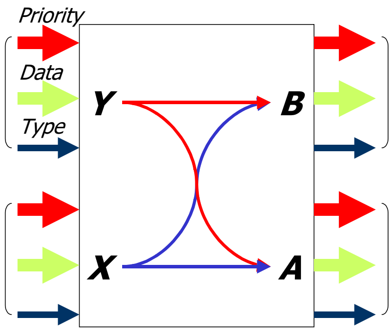
This work addresses key issues in practical packet switched networks: high throughput with low latency, priority-based sorting, and extensible architecture yielding large storage capacity. We describe the architecture in detail and explain how the buffer capacity of the network can be extended, without incurring excessive latency, by stacking up multiple layers of the network. We suggest how slower (and presumably cheaper) sorting networks/systems can be used to extend the buffer capacity of our network without sacrificing its functionality, since large storage capacity is critical for tolerating bursts of conflicting packets. We also present a general technique called *timed-roadblocking* for allocating more stages in each layer of the network than circuit speed would normally allow. We then describe our implementation of the sorting network using zero-overhead, self-timed, self-precharging domino logic. Our implementation technique enables extremely fast operation of the network with zero control overhead. Finally, we provide a brief overview of an ATM chip that we are currently designing using the techniques described in this paper.

2 Architecture

This section describes the architecture of a 2×2 real-time packet sorting network. The sorting network receives up to two packets in every packet slot and transmits up to two packets at the end of each packet slot. We assume that there are two packet types, A and B, and two associated output ports, A and B. Type A packets are destined for output port A, and type B packets for output port B. Contention arises if two input packets of the same type arrive at the same time, in which case the sorting network stores one of the packets in a buffer. The criterion used to determine which one is to be buffered and which one is to be transmitted is based on the *priority* stamped on each input packet. In each packet slot, our sorting network outputs on port A the type A packet with the highest priority of all type A packets, both incoming and stored. Similarly, the type B packet with the highest priority is outputted on port B.

2.1 Basic Switching Element

The basic switching element, henceforth called 2×2 *crossbar*, used in our sorting network is illustrated in Figure 1. It has two input ports (X and Y) and two output ports (A and B). If a single packet arrives, then it leaves the crossbar on its destined output port. If two packets of different types arrive in the same slot, then each packet is routed to its destined port. However, if two packets of the same type arrive in the same slot, the crossbar routes the packet with higher priority to its destined port and the packet with lower priority to the other port.



Cross only if		
Port X	Port Y	Priority Level
type B	type A	–
type B	none	–
none	type A	–
type A	type A	port X < port Y
type B	type B	port X > port Y

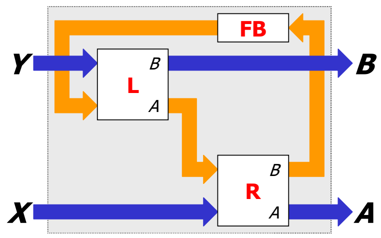
Figure 1. Basic switching element: 2×2 **crossbar.**

2.2 TC Stage

A TC ("Track Changer") stage is constructed from two 2×2 crossbars and a latch as depicted in Figure 2. The TC stage functions similarly to the crossbar but with an additional buffering capability. *Assuming that incoming packets have been sorted by a crossbar*, the TC stage routes the packets with the highest priorities to their destined ports and stores the remainder in the buffer, unless all packets, both incoming packets and the stored one, are of the same type.

For example, if the packet stored in the buffer is type A and the incoming packets are of different type (type A on input port X and type B on input port Y), then crossbar L routes the type B packet to output port B and the type A packet stored in the buffer to crossbar R. Crossbar R compares the priority levels of the two type A packets, routes the one with higher priority to output port A, and routes the one with lower priority to the buffer.

However, if all three packets are of type A, then crossbar L has no choice but to route a type A packet to output port B. Because the incoming packets are assumed to have been sorted by a crossbar, the type A packet entering input port Y must have lower priority than the one entering input port X. The priority of this packet is compared to that of the type A packet stored in the buffer. Since the packet with the lower of the two priorities is routed to port B, the one leaving on port B must have the lowest priority of all three. In contrast, the one with the highest priority leaves on its destined port, port A, and the remainder (with the medium priority) is stored in the buffer.



Packets	Port A	Port B	Buffer
$3A$	A_{high}	A_{low}	A_{medium}
$3B$	B_{low}	B_{high}	B_{medium}
$2A, B$	A_{high}	B	A_{low}
$A, 2B$	A	B_{high}	B_{low}
A, B	A	B	–

Figure 2. TC stage and its function. We assume that packets have been sorted by a crossbar before entering the TC stage. $3A$ means that both the incoming packets and the packet stored in the buffer are all type A; $2A, B$ means that two are type A and one is type B. A_{high}, A_{medium}, and A_{low} are type A packets with *high*, *medium*, and *low* priority levels respectively.

2.3 TC Chain

A TC chain of length N is constructed by cascading a 2×2 crossbar and N TC stages in series. The TC chain implemented as a cascaded optical delay line *with no priority based sorting* was first introduced in [5]. In this paper, we extend that work to an electronic implementation with priority based sorting.

A two-stage TC chain is shown in Figure 3. The crossbar labeled 0 sorts the incoming packets and routes them to stage 1 of the TC chain. When an incoming packet arrives, crossbar L_1 sorts it against the packet stored in FB_1 and routes a packet each to R_1 and L_2. R_1 and L_2, in turn, sort packets and route them to the next stage.

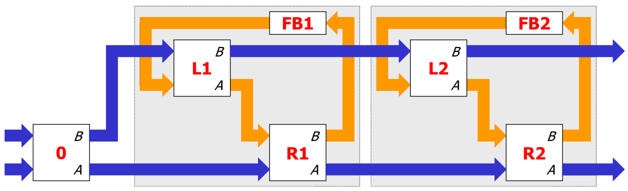


Figure 3. 2-stage TC chain.

It is not difficult to verify that a TC chain constructed this way has the following properties.

- Type A (B) packet leaving on output port A (B) has higher priority than all other type A (B) packets stored in the chain.
- Type A (B) packet leaves on output port B (A) if and only if
  - both incoming packets are of type A (B) and
  - the TC chain is filled only with type A (B) packets with higher priorities.

In other words, the packet leaving on port A is either the type A packet with the highest priority or the type B packet with the lowest priority.

A drawback of this architecture is that the latency of a TC chain is proportional to the number of stages, N, in the chain. That is, type and/or priority comparison must be made in every stage serially before determining which packets are to leave the chain in a given packet slot. Therefore, increasing the buffer capacity also increases the sorting latency.

2.4 TC Stack

In general, we need large storage capacity without incurring long sorting latency. To increase the buffer capacity without increasing the sorting latency, we employ the architecture depicted in Figure 4. Incoming packets enter the bottom TC layer. Overflow packets from layer L are "pushed" onto layer $L + 1$. In a given packet slot, packets "popped" from layer $L + 1$ (descending from $L + 1$ to L) have the highest priority of the incoming packets to layer $L + 1$ and all the packets stored in layers $L + 1, L + 2, \ldots$. We cap the sorting latency at the delay through a single TC layer by insuring that the bottom layer contains a packet of each type with higher priority than the packet of the same type descending from overflow layers.

Each TC layer is implemented as shown in Figure 5. *Mixed* buffer is the main sorting network implemented as an N-stage TC chain. A/B-buffers are shorter M-stage TC chains ($N \geq M$). Because A-buffer and B-buffer are completely symmetrical as shown in Figure 5, without loss of generality, we limit our discussion to A-buffer only. If it is assumed that packets entering overflow input port X (see Figure 5) are only of type A, then A-buffer

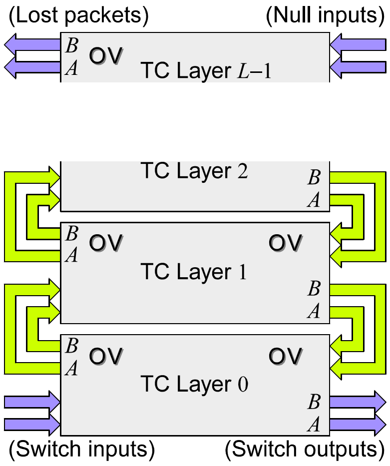


Figure 4. TC stack.

never receives two type B packets in a packet slot. Therefore, if a type B packet enters input port X of A-buffer, it leaves A-buffer via overflow port B in the same packet slot. This means that no type B packet can be stored in A-buffer, which is the reason it is named as such. It is also easy to verify that a type A packet leaves via overflow port B (see Figure 5) only if A-buffer is filled with M type A packets with higher priorities and both the packets entering A-buffer are of type A.

Suppose a packet $\mathcal{P}$ of type A overflows from layer L and requires a minimum of K packet slot delays to descend back to L. We define the minimum *sojourn time* of $\mathcal{P}$ to be $K \times$ (packet slot delay). In order to insure that no new type A packet with lower priority than $\mathcal{P}$'s "leaks out" from layer L via output port A before $\mathcal{P}$ returns to this layer, there must be at least K type A packets with higher priority stored in layer L. We enforce this by choosing $M \geq K$ and $N \geq K$ in the TC stack. To understand why this works, consider the scenarios that can lead to a type A packet leaving layer L through an overflow port:

- If it left via overflow port A, then it must have entered B-buffer from *Mixed* buffer. *Mixed* buffer must therefore have been filled with N type A packets with higher priorities. As long as $N \geq K$, there is no "leakage" problem.
- If it left via overflow port B, then it must have departed from A-buffer and A-buffer must have been filled with M type A packets with higher priorities. Thus there is no "leakage" problem.

Observation: If the minimum sojourn time is large, i.e., overflow layers are slow, then the size of A/B-buffers can be adjusted to compensate. In fact, we just need to increase M to match the speed of the slow overflow layers. Thus it is conceivable to use a *heterogeneous storage architecture*, in which overflow layers are implemented with a completely different architecture, e.g., an SRAM-based architecture.

When building sorting networks with large buffer capacity, economic considerations might actually require us to use such heterogeneous architectures, with the overflow layers implemented in

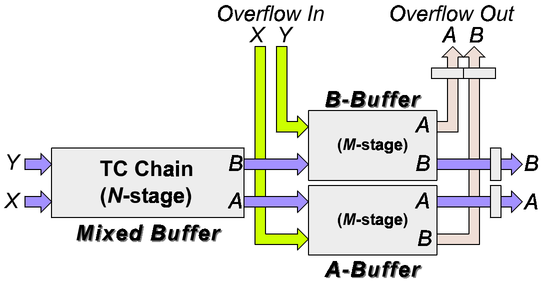


Figure 5. TC layer.

cheaper and slower technology. Since the speed of the overflow layer sets a lower bound on M and N, we might be forced to implement a TC layer with a large number of stages. The next section describes a technique for achieving this efficiently.

3 Architectural analysis

Ideally, if there are N stages with $T_{\max}$ as the maximum delay through each TC stage, we would like the worst-case latency of the chain to be $NT_{\max}$. Unfortunately, conventional implementation styles, such as pipelining, can cause the worst-case latency to be significantly higher than $NT_{\max}$ due to the overhead associated with pipeline latches. Therefore, more sophisticated techniques are needed to increase the number of stages.

3.1 A Simple Pipelined Implementation

For comparison, we first consider a pipelined implementation of an N-stage TC chain. Each TC stage is treated as a pipeline stage with latches inserted between consecutive stages. Assuming that the processing delay of each TC stage varies between $T_{\min}$ and $T_{\max}$ and a pipeline latch has a setup time of t_{su} and an input-to-output delay of T_{pd}, the worst-case latency of the pipeline with N stages (hence $N-1$ latches) is:

$$\mathcal{T} = NT_{\max} + (N-1)(t_{\text{su}} + T_{\text{pd}}). \tag{1}$$

Therefore, the difference between the actual latency and the desired latency ($NT_{\max}$) grows as N increases.

3.2 A Cascaded Implementation

We can cascade TC stages without intervening latches, which *guarantees* that the worst-case latency of the TC chain is $NT_{\max}$. Unfortunately, there is a fundamental limit on the length, N, of the chain that can be connected this way, as shown below.

For clarity of exposition in the following analysis, we treat the flow of packets in the chain as though a single packet propagated all the way through N stages in every packet slot. In reality, however, any packet may interact with a stored packet in a buffer and "bump" the stored packet down the chain in its place.

Let T_c represent the time separation between two consecutive packets entering the chain. Assume that the first packet entering the chain at $t = 0$ takes the worst-case delay, $T_{\max}$, through every stage. Thus, the first packet leaves stage k ($k = 1, \ldots, N$) at $t = kT_{\max}$. If the second packet entering the chain at $t = T_c$ *arrives at* stage k before the first packet *leaves* stage k, then a "collision" occurs, which may corrupt the packets. The earliest time that the second packet can arrive at stage k is $T_c + (k-1)T_{\min}$. Consequently, the condition for collision is: $kT_{\max} \geq T_c + (k-1)T_{\min}$.

If a minimum settling time Δ is required between the departure of the first packet and the arrival of the second packet in a TC stage (this corresponds to the precharging time in a self-precharging logic), the condition for collision is modified to: $kT_{\max} + \Delta \geq T_c + (k-1)T_{\min}$. Therefore, to avoid collision in a chain of length N, we must have:

$$N < N_0, \quad \text{where } N_0 = \frac{T_c - (\Delta + T_{\min})}{T_{\max} - T_{\min}}. \tag{2}$$

This is a fundamental limit on N for correct operation of the chain. Note that this analysis is similar to those used in [3, 10] to determine clocking frequencies in wavepipelined circuits.

3.3 Timed-Roadblocking

In order to allocate more stages than N_0, we must prevent later packets from overtaking earlier packets. However, in doing so, we must be careful not to further slow down the propagation of the slowest packets, i.e., we must prevent the worst-case latency of the chain from exceeding $NT_{\max}$. One way to accomplish this is to place "timed-roadblocks" along the TC chain in such a way that fast packets are prevented from moving down the chain too fast, while the propagation of slow packets is never blocked.

We now describe a method for implementing long TC chains using timed-roadblocking. Let N be the total number of stages in the TC chain. We divide N stages into R segments and insert timed-roadblocks between consecutive segments. Each segment, except possibly the last one, consists of $n = \lceil \frac{N}{R} \rceil$ stages. For simplicity of analysis, we will assume the following timing of roadblocks. Each roadblock is enabled by a signal with period T_c. If packets enter the chain at $t = kT_c,\ k \in \{0, 1, \ldots\}$, the first roadblock, $\mathcal{R}_1$, is opened (allowing packets to flow through) at $t = kT_c + \delta,\ k \in \{0, 1, \ldots\}$. Once $\mathcal{R}_1$ is opened, it remains open for τ, where $\delta + \tau \leq T_c$. Each subsequent roadblock, $\mathcal{R}_i$, opens δ after $\mathcal{R}_{i-1}$ opens, and remains open for τ. This behavior is depicted in Figure 6. In this example, 3 roadblocks are used, $\delta = \frac{T_c}{4}$, and $\tau = \frac{T_c}{2}$.

Note that a similar technique was used in [6] to tame the clock-skew problem in synchronous domino circuits. The present work generalizes the technique of [6] and applies it to solve a fairly general complex design problem.

Timing Constraints: Below, we derive the constraints for the chain to function correctly with worst-case latency $= NT_{\max}$.

Assume that packet $\mathcal{P}$ enters the chain at $t = 0$ and takes the worst-case delay, $T_{\max}$, to propagate through each TC stage. Thus $\mathcal{P}$ arrives at the first roadblock, $\mathcal{R}_1$, between stages n and $n+1$, at $t = nT_{\max}$. Since $\mathcal{R}_1$ remains open between $t = \delta$ and $t = \delta + \tau$, $\mathcal{P}$'s propagation is *guaranteed* not to be impeded if (i) $nT_{\max} \geq \delta$, and (ii) $(n+1)T_{\max} \leq \delta + \tau$. The first condition guarantees

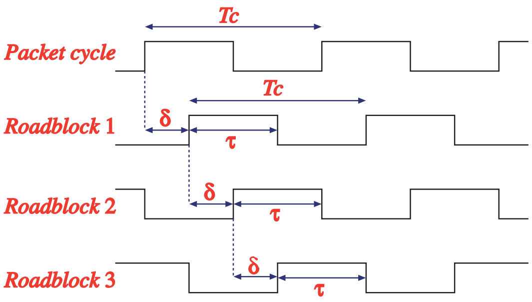


Figure 6. Timed-roadblocks: packets arrive at the beginning of each packet cycle; road-block i is open when it is high.

that roadblock $\mathcal{R}_1$ opens before $\mathcal{P}$ arrives. The second condition insures that $\mathcal{P}$ leaves before $\mathcal{R}_1$ closes. In general:

$$k\delta \leq knT_{\max} \quad \text{or} \ \delta \leq nT_{\max} \tag{3}$$

$$(kn+1)\,T_{\max} \leq k\delta + \tau \tag{4}$$

for all $k \in \{1, \ldots, R-1\}$. Note that an implementation satisfying Inequalities 3 and 4 *guarantees* that packet $\mathcal{P}$ is never slowed down by any roadblock; consequently, $\mathcal{P}$ incurs a latency of $NT_{\max}$.

The purpose of roadblocks is to slow down fast packets so that fast packets do not overtake slow packets. To slow down fast packets, roadblocks must be closed when the fast packets arrive; i.e., $\delta > nT_{\min}$. In other words, δ must be greater than the minimum packet propagation delay between any two consecutive roadblocks. Combining this with Inequality 3, we obtain the range for δ as:

$$nT_{\min} < \delta \leq nT_{\max} \tag{5}$$

We now derive conditions for "collision" of packets in a TC chain with timed-roadblocks. As before, we assume that the first packet, $\mathcal{P}_1$, enters the chain at $t = 0$ and takes the worst-case delay, T_{max}, through each stage. Since roadblocking does not slow down the slowest packets, $\mathcal{P}_1$ leaves the last stage in the kth segment (stage nk) at $t = nkT_{\max}$. The next packet, $\mathcal{P}_2$, enters the chain at $t = T_c$. Inequality 5 now implies that the earliest time $\mathcal{P}_2$ reaches stage nk is $T_c + (k-1)\delta + (n-1)T_{\min}$. Therefore, to avoid collision of packets, we must have:

$$T_c + (k-1)\delta + (n-1)T_{\min} > knT_{\max} + \Delta, \tag{6}$$

for all $k \in \{1, \ldots, R\}$, where Δ is the settling time required between the departure of a packet and the arrival of the next packet in a TC stage.

Inequalities 4 through 6, along with $n = \lceil \frac{N}{R} \rceil$ and $\delta + \tau \leq T_c$, spell out the constraints that must be satisfied for the TC chain with timed-roadblocks. We assume that T_c, $T_{\min}$, $T_{\max}$ and Δ are system parameters. The design variables are N, R, δ and τ. Depending on the application, we might, for example, want to determine the maximum number of stages, N, that can be allocated using a certain number of roadblocks, R. Alternatively, we might want to determine R, δ and τ for a desired number of stages, N. In the following example, we determine the maximum number of stages, N, that can be allocated using one roadblock.

Example: Assume that practical considerations impose the following restrictions: $\tau = \frac{T_c}{2}$ and $\delta = \frac{T_c}{R}$, where $R - 1 =$ number of roadblocks. Given these constraints, we want to determine the maximum number of stages, N, that can be allocated while capping the latency at $NT_{\max}$. We choose $N = nR$, where n is an integer. Inequality 5 now becomes:

$$\begin{aligned} \frac{N}{R}T_{\min} \ &< \ \frac{T_c}{R} \ \leq \ \frac{N}{R}T_{\max} \\ \Rightarrow \frac{T_c}{T_{\max}} \ \leq \ N \ &< \ \frac{T_c}{T_{\min}} = \nu_1 \end{aligned}$$

The constraint $\delta + \tau \leq T_c$ implies that:

$$\frac{T_c}{R} + \frac{T_c}{2} \ \leq \ T_c \ \ \Rightarrow \ \ R \ \geq \ 2 \tag{7}$$

Inequality 4, applied to this example, implies that for all $k \in \{1, \ldots, R-1\}$,

$$\begin{aligned} \left(\frac{kN}{R} + 1\right) T_{\max} \ &\leq \ \frac{kT_c}{R} + \frac{T_c}{2} \\ \Rightarrow N \ &\leq \ \frac{T_c}{T_{\max}} + \frac{R}{k}\left(\frac{T_c}{2T_{\max}} - 1\right) \end{aligned} \tag{8}$$

The right hand side of Inequality 8 is minimized at $k = R - 1$, yielding

$$N \ \leq \ \frac{T_c}{T_{\max}}\left(1 + \frac{1}{2}\frac{R}{R-1}\right) - \frac{R}{R-1} = \nu_2 \tag{9}$$

Finally, Inequality 6 implies that for all $k \in \{1, \ldots, R\}$,

$$\begin{aligned} T_c + (k-1)\frac{T_c}{R} + \left(\frac{N}{R} - 1\right)T_{\min} \ &> \ k\frac{N}{R}T_{\max} + \Delta \\ \Rightarrow N \ &< \ \frac{(\frac{k}{R} + \frac{R-1}{R})T_c - (T_{\min} + \Delta)}{\frac{1}{R}(kT_{\max} - T_{\min})} \end{aligned} \tag{10}$$

The right hand side of Inequality 10 is minimized at $k = R$. The corresponding upper bound on N is:

$$N \ < \ \frac{(1 + \frac{R-1}{R})T_c - (T_{\min} + \Delta)}{T_{\max} - \frac{T_{\min}}{R}} = \nu_3 \tag{11}$$

Thus, the number of stages, N, using R roadblocks, is given by:

$$\frac{T_c}{T_{\max}} \leq N < \min(\nu_1, \nu_2, \nu_3)$$

With $T_c = 42, T_{\min} = 1, T_{\max} = 3$, $\Delta = 12$, and we obtain $\frac{T_c}{T_{\max}} = 14$. The maximum number of stages possible without roadblocks is: $N_0 = 14$. With $R = 2$, we obtain $\nu_1 = 42$, $\nu_2 = 26$ and $\nu_3 = 20$. Therefore, $14 \leq N < \min(42, 26, 20) = 20$; i.e., we can allocate up to 19 stages using only one roadblock. In fact, it can be shown that, with our choice of T_c, $T_{\max}$, $T_{\min}$, δ and τ, $N \leq 21$ for all $R > 2$. $R = 2$, therefore, is a reasonably good design point, since it allows nearly as many stages as the maximum achievable while using only one roadblock.

4 Implementation

Our implementation of the TC chain is a self-timed, self-precharging domino logic design. Self-timed, self-precharging design avoids the need for a precise clock distribution network and allows "zero-overhead" evaluation [16]. Control logic is simple and efficient as well.

Each crossbar starts in evaluation mode and operates immediately on arriving packets. Its outputs trigger its successors to evaluate immediately. It then prepares itself for the next pair of packets based on feedback from its predecessors and successors. In other words, it precharges itself when its outputs have been absorbed by its successors (latches and crossbars) and puts itself back in evaluation mode when all of its predecessors are precharged. Note that evaluation may be *enabled* well in advance of the next packet's arrival — the only constraint is that all predecessors be precharged, so that new inputs can be distinguished from old ones. As such, *no control overhead* delay is incurred upon arrival of a new packet. This technique substantially accelerates the flow of packets through the TC chain by hiding not only the controller latency, but also the delays to enable evaluation transistors. Furthermore, packets flow through the chain with minimal synchronization delay — synchronization is needed only at R crossbars.

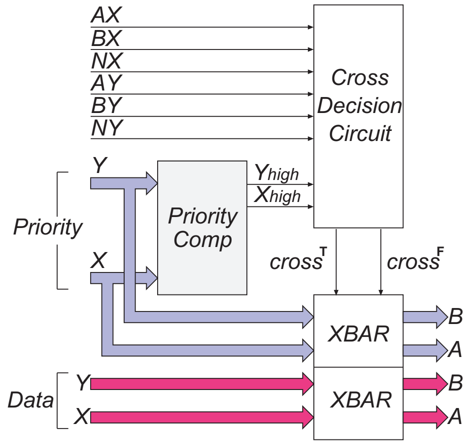
Roadblocks are implemented by a single static gate inserted in the evaluation enable control. They introduce overhead only for fast packets. For slower packets, the added gate delay is hidden; i.e., the roadblock is open before the slow packet arrives. This achieves the goal of blocking overly fast packets without delaying slow packets.

4.1 Datapath

The datapath consists of three TC chains (*Mixed* buffer, A/B-buffers). Each TC stage consists of two crossbars and a feedback latch. The crossbars are further divided into decision units and data-routing "slices". Each decision unit consists of a dual-rail domino borrow-skip subtractor and a dual-rail cross decision circuit, as shown in Figure 7. Decision to cross or not to cross is based on packet types and priority comparison. When packets are of the same type, priorities are compared to determine routing. When packets are of different types, each packet is immediately routed to its destined port. 16-bit unsigned priorities are used; therefore, 32 signals are required for dual-rail signaling. Packet types are one-hot encoded: 3 signals indicate type A, B or *no packet*. Each data slice routes 8 bits. Eight data slices are used in conjunction with each crossbar, totaling 64 bits of packet data. In contrast to the priority and type bits, no logical operations are performed on data bits; thus less costly single-rail domino logic suffices for the data crossbars.

4.2 Control

Datapath control consists of extended burst-mode controllers [17], implemented as generalized-C elements [18]. The routing decision is made at the crossbar as shown in Figure 7 and propagated from one data slice to the next so that a broadcast to all



AX	type A packet on X port
BX	type B packet on X port
NX	no packet on X port
AY	type A packet on Y port
BY	type B packet on Y port
NY	no packet on Y port

Figure 7. Crossbar block diagram.

constituent data slices is not necessary. Control flow from crossbar to crossbar across the TC chain is considerably more complex. An example detailing the process is presented below.

4.2.1 Control Flow Example

Evaluation of datapath elements is done in a true *domino* fashion — the arrival of each new packet starts a domino effect, unencumbered by control. Control is only used for setting up the datapath elements for the next cycle, such as precharging crossbars and latching data in the feedback buffers. Each datapath element (except feedback latches) operates as follows:

- Start in evaluation mode;
- Evaluate as soon as the data from its predecessors arrive;
- Precharge only after all successors have finished evaluation;
- Set up for the next evaluation only after all its predecessors have finished precharging.

Consider stage 1 of a two-stage TC chain, as shown in Figure 8. When L_1 receives a packet (data, priority, and type information), it starts evaluation immediately. When done evaluating, it presents inputs to L_2 and R_1, which in turn start evaluating immediately. L_1, once done evaluating, no longer requires its inputs to remain asserted. FB_1 therefore precharges in preparation for the next cycle. L_1 cannot precharge until its successors, L_2 and R_1, have completed evaluation (otherwise their inputs would vanish). Once both L_2 and R_1 have finished evaluating, L_1 precharges. However, R_1 is unable to complete its cycle until it stores its data in FB_1. After FB_1 evaluates, R_1 precharges. At this point, stage 1 has completed an evaluation/precharge cycle. Once the inputs from crossbar 0 have become low, L_1 and R_1 enter evaluation mode, and are ready to receive new packets.

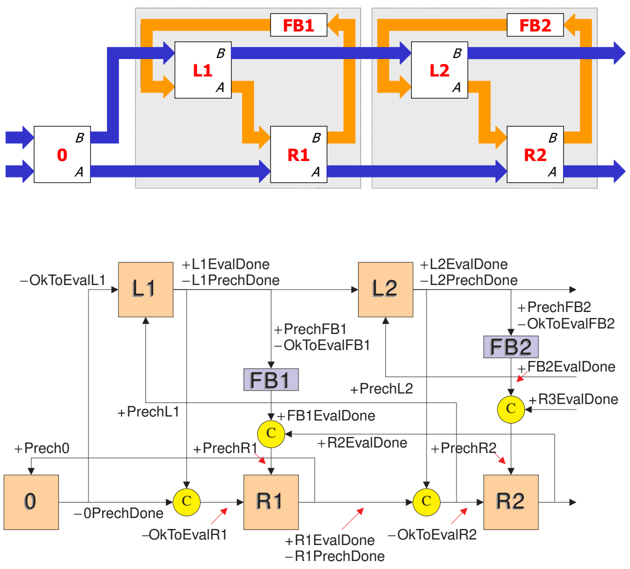


Figure 8. A two-stage TC chain and its control: $+$L1EvalDone denotes that a rising transition on the wire that it labels indicates that L_1 has finished evaluating. $-$OkToEvalL1 denotes that a falling transition on the wire that it labels indicates that it is "okay" to put L_1 in evaluation mode.

4.2.2 Extended Burst-Mode Controllers

The controllers are implemented as extended burst-mode asynchronous finite state machines [17]. A state transition $S_i \rightarrow S_j$, from state i to state j is *enabled* by an *input burst* and is accompanied by an *output burst* (also enabled by the same input burst). An input burst consists of one or more *terminating edges*, denoted by $Signal-$ ($Signal$ falling), or $Signal+$ ($Signal$ rising). For example, in Figure 9(a), $S_0 \rightarrow S_1$ is enabled by $(SuccDone-, PredDone+, Done+)$. Terminating input edges in an input burst may arrive in any order, but the corresponding output burst can occur only after all have arrived. As drawn in Figure 9, signals that change during the output burst are specified after '/'. The output burst enabled during $S_0 \rightarrow S_1$ is $(Eval-)$.

$Signal*$ specifies a *directed don't-care*. A sequence of state transitions marked with $Signal*$ followed by a transition marked with $Signal-$ means that $Signal$ falls *exactly once* somewhere during the sequence of state transitions.

4.2.3 Throughput Constraints

Recall that, during evaluation, the TC chain behaves like combinational logic. Precharging of crossbars and storing new data in the feedback buffers are done in the background, which poses no problem as long as the next packet arrives after a sufficient time. Here we consider the problem of how soon the next packet can arrive; i.e., the constraints on the throughput imposed by the time required for processing background tasks.

The crossbar must be in evaluation mode before new packets from its predecessors arrive, in order to prevent slowing down the input packets. This constraint is shown as a dashed arrow in Figure 9(a): i.e., $Eval+$ must precede $PredDone+$. This is a constraint on maximum throughput.

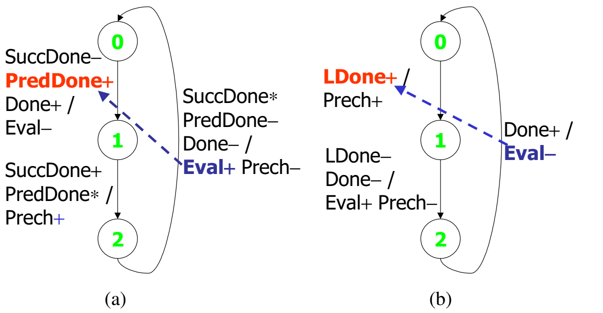


Figure 9. (a) Crossbar control; (b) feedback latch control.

Similarly, the feedback latch must be ready for precharging, i.e., must have disabled evaluation, before its successor (L crossbar) completes its evaluation. The reason for this is that L crossbar prompts the feedback latch to begin precharging when it finishes evaluating. This constraint is also shown as a dashed arrow in Figure 9(b), i.e., $Eval-$ must precede $LDone+$. This is another constraint on maximum throughput.

5 Practical Example: ATM Packet Switch

One application of the sorting network is an ATM packet switch supporting *deadline-based quality of service* (QoS). A chip design, compatible with details described in the previous sections, is nearly completed. Output packet sequencing is prioritized based on the deadline provided with the packet data and type. ATM packets (cells) contain 53 bytes (424 bits). Each chip supports 64 bits of data for cost reasons: our implementation can support the entire 424 bits on one chip, but the test chip fabrication cost would be prohibitive. The switch is designed to support 10 Gb/s per port. This rate yields a bit time of 0.1ns, and packet time of 42.4ns. Each 53-byte packet is sliced to eight 53-bit slices as shown in Figure 10. Each slice is split among 7 chips. It takes 5.3ns to load each slice into the switch and 42.4ns per packet. Deadline and type are broadcast to all 7 chips along with the first slice of each packet.

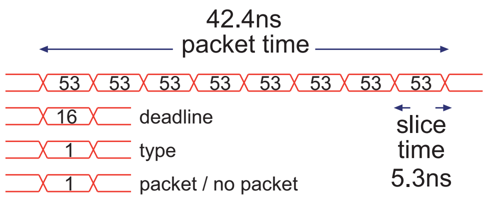


Figure 10. Slicing packets.

Our self-timed design has zero-overhead during evaluation. Without roadblocking, approximately 13 TC stages can be cascaded, each with worst-case delay of about 3ns. Addition of a single roadblock allows 20 stages, requiring approximately 1.5 clock cycle latency.

The sorting network, of course, forms only a part of the complete 2×2 switch function. Surrounding logic includes serial-to-parallel input conversion and parallel-to-serial output conversion. Routing and deadline assignments are arguably the most significant external functions. The packet type assignment is the equivalent of a routing decision and is handled outside the sorting chip. Most importantly, the sorting network assumes that a deadline is stamped on each packet. Deadlines can be assigned based on a packet priority combined with a constantly advancing (and cyclic) baseline: $Deadline = t_s + t_\delta$, where t_s is the baseline, and t_δ is the allowable delay based on priority. The baseline portion insures that deadlines increase as time elapses, so that not all similar priority packets are assigned similar deadlines over time. For example, old high-priority packets should have precedence over new high-priority packets. Assignment of a suitable priority to each packet is crucial in the pursuit of meaningful QoS guarantees. The deadline assignment can be derived from "service curves" [4].

Packet Pointer Architecture: Rather than storing the entire packet in the sorting network, we could store packets in an external memory, and instead sort pointers to fixed-size packet "slots" as shown in Figure 11. Slots are suitable for ATM cells or for fragments of larger variable length packets. One possible implementation described below uses a circular list of pointers to the packet slots. Slots are assigned to new packets from the head of the list, and freed slots are appended to the tail end of the list following packet transmission.

Clearly, both allocating and returning a slot are $\mathcal{O}(1)$ operations. Note that slots in use are not linked in any way: the order in which slots are freed (packets transmitted or lost) is determined entirely by the order in which slot pointers exit the switch. In any cycle, there may be up to two writes to, and two reads from, the packet memory. This implies an additional two reads and *four* writes for the list maintenance — it is possible to have up to two packets overflowing from the switch, and the slots for these packets must also be reclaimed; hence two additional writes are needed. Because of the large number of memory accesses in each cycle, it is prudent to separate the pointer memory from the packet memory. The packet memory would require using, as a minimum, dual-ported 20ns SRAM's to support four accesses in one 42.4ns packet cycle, assuming 53-byte ports. With an upper limit of several million pointers stored inside the switch (assuming a heterogeneous architecture described in Section 2.4), an implementation consisting entirely of off-the-shelf SRAM's and a simple VLSI memory controller is definitely feasible.

Finally, we can use 2×2 packet switches to form larger switches, e.g., 8×8 switch [15, 14]. Among these are several multi-stage distributed-buffer networks; e.g., Omega, Banyan, Flip, and Baseline [15].

6 Conclusion

This paper addressed key issues in practical packet switched networks: high throughput with low latency, priority-based sorting and extensible architecture yielding large storage capacity. We described the basic architecture based on TC stages and suggested how slower cheaper sorting networks can be used to extend the buffer capacity of our network, without sacrificing the sorting latency. We also presented the *timed-roadblocking* technique for allocating more stages in a TC chain than circuit speed would normally allow. We believe that our analysis is applicable to other systems of cascaded stages as well. More importantly, we used our analysis to quantitatively justify our choice of parameters in the actual sorting network design. We described our implementation of the sorting network used in an ATM chip under development. We showed that our zero-overhead, self-timed, self-precharging domino logic design with timed-roadblocking enables us to achieve zero control overhead. Finally, we are investigating a hierarchical storage architecture to extend our current design.

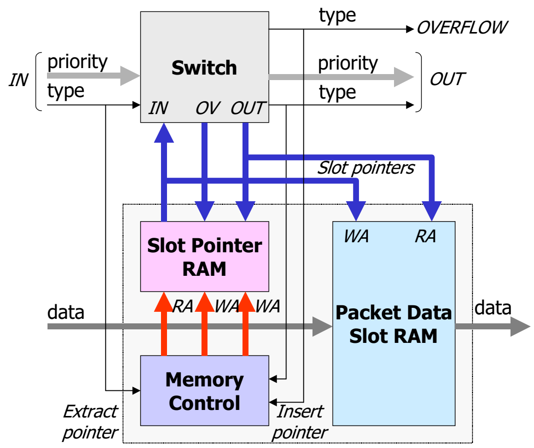


Figure 11. Packet pointer switching system architecture.

Acknowledgment

We would like to thank Jason Hernandez of IBM, Austin, TX for his comments on the use of pointers to increase buffer capacity.

References

[1] S. E. Butner and R. Chivukula. On the limits of ATM switching. *IEEE Network Magazine*, 10(6):26–31, November/December 1996.

[2] G. Chen and I. Stavrakakis. ATM traffic management with diversified loss and delay requirements. In *Proceedings of IEEE INFOCOM*, pages 1037–1044, 1996.

[3] L. Cotten. Maximum rate pipelined systems. In *AFIPS Proceedings of Spring Joint Computer Conference*, pages 581–586, 1969.

[4] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[5] R. L. Cruz and J. T. Tsai. COD: Alternative architectures for high speed packet switching. *IEEE/ACM Transactions on Networking*, 4(1):11–21, February 1996.

[6] D. Harris and M. A. Horowitz. Skew-tolerant domino circuits. *IEEE Journal of Solid-State Circuits*, 32(11):1702–1711, November 1997.

[7] M. R. Hashemi and A. Leon-Garcia. The single-queue switch: a building block for switches with programmable scheduling. *IEEE Journal on Selected Areas in Communications*, 15(5):785–794, June 1997.

[8] R. Kannan, K. Y. Lee, and H. F. Jordan. A high-speed hardware efficient optical sorting network. In *Proceedings of Asia-Pacific Conference on Communications*, pages 873–877, 1995.

[9] A. R. Krappel. CAN (Controller Area Network) controller for real-time use. *Elektronik*, 46(10):88–92, May 1997.

[10] W. K. C. Lam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Valid clock frequencies and their computation in wavepipelined circuits. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems*, 15(7):791–807, July 1996.

[11] K. L. E. Law and A. Leon-Garcia. A large scalable ATM multicast switch. *IEEE Journal on Selected Areas in Communication*, 15(5):844–854, June 1997.

[12] S. W. Moore and B. T. Graham. Tagged up/down sorter – A hardware priority queue. *The Computer Journal*, 38(9):695–703, September 1995.

[13] R. Palmer. An experimental ATM switch for BISDN studies. *International Journal of Digital and Analog Communication*, 3(4):341–349, 1990.

[14] J. Turner and N. Yamanaka. Architectural choices in large scale ATM switches. Report WUCS 97-21, Washington University, Department of Computer Science, May 1997.

[15] J. Walrand and P. Varaiya. *High Performance Communication Networks*. Morgan Kaufmann Publishers, Inc., 1996.

[16] T. E. Williams. Latency and throughput tradeoffs in self-timed asynchronous pipelines and rings. Technical Report CSL-TR-90-431, Stanford University, August 1990.

[17] K. Y. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. PhD thesis, Stanford University, August 1994.

[18] K. Y. Yun. Automatic synthesis of extended burst-mode circuits using generalized C-elements. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 290–295, September 1996.