Terabit Burst Switching Progress Report (9/98–12/98)

Jonathan S. Turner
jst@cs.wustl.edu

WUCS-98-31

March 28, 1999

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

This report summarizes progress on Washington University's *Terabit Burst Switching Project*, supported by DARPA and Rome Air Force Laboratory. This project seeks to demonstrate the feasibility of *Burst Switching*, a new data communication service which can more effectively exploit the large bandwidths becoming available in WDM transmission systems, than conventional communication technologies like ATM and IP-based packet switching. Burst switching systems dynamically assign data bursts to channels in optical data links, using routing information carried in parallel control channels. The project will lead to the construction of a demonstration switch with throughput exceeding 200 Gb/s and scalable to over 10 Tb/s.

[0]This work is supported by the Advanced Research Projects Agency and Rome Laboratory (contract F30602-97-1-0273).

Terabit Burst Switching
Progress Report (9/98–12/98)

Jonathan S. Turner
jst@cs.wustl.edu

This report summarizes progress on the Terabit Burst Switching Project at Washington University for the period from September 15, 1998 through December 15, 1998.

1. Improved Link Scheduling

One of the key issues in the design of a burst switch is scheduling the transmission of bursts on outgoing channels. Previously, we have concentrated on a technique called *horizon scheduling*. In horizon scheduling, the link scheduler maintains a time horizon for each of the channels of an outgoing link. The horizon is defined as the earliest time after which there is no planned use of the channel. The horizon scheduler assigns arriving bursts to the channel with the latest horizon that is earlier than the arrival time of the burst, if there is such a channel. If there is no such channel, the behavior depends on whether the system supports burst storage or not. In a system that does not support burst storage, the scheduler simply discards the burst. In a system that does support storage, the burst is assigned to the channel with the smallest horizon and is delayed until that channel becomes available. Horizon scheduling is straightforward to implement in hardware, but because it does not keep track of time periods before a channel's horizon when the channel is unused, it cannot insert bursts into these open spaces.

Figure 1 illustrates the operation of a horizon scheduler. The diagram shows an arriving burst that needs to be assigned to an outgoing channel. The burst header cell arrives at time t with the start of the burst arriving at $t + \Delta_1$ and the end of the burst arriving at $t + \Delta_2$. The right hand part of the figure shows bursts that arrived earlier that were assigned to channels by the horizon scheduler. The thick lines indicate time periods during which the various channels are scheduled to be in use and the shaded region shows those time periods that are currently unavailable to new bursts. The box in the center highlights the time period during which the arriving burst will need an outgoing channel and the check marks at the right indicate those channels that it could be assigned to. Since the horizon scheduler prefers the viable channel with the latest horizon, it will select the bottom channel.

In some situations, the horizon scheduler can provide good performance. Let $b_1, \ldots, b_n$ be a sequence of bursts, where b_i is characterized by a triple (r_i, t_i, ℓ_i) and r_i is the time at which the link scheduler receives the burst header cell informing it of the imminent arrival of the burst, t_i is the

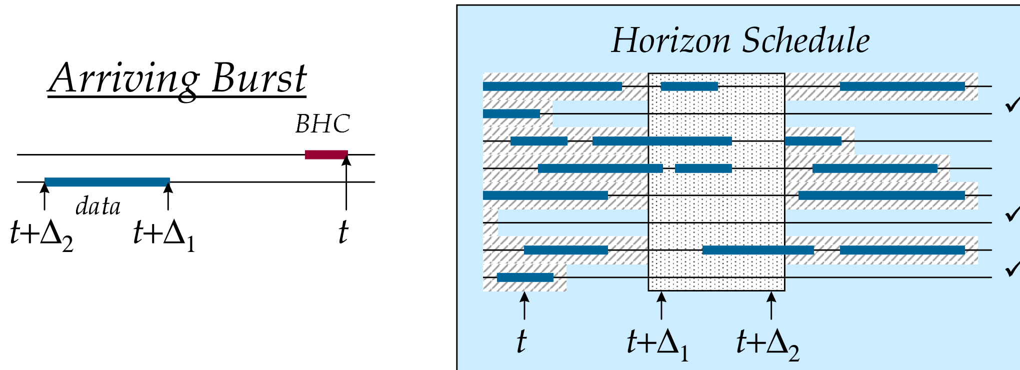

Figure 1: Horizon Link Scheduling

arrival time of the burst and ℓ_i is the length (time duration) of the burst. For convenience, assume that for $i < j, r_i \le r_j$; that is, the bursts are listed in the order in which the burst header cells arrive. Define the *width* $W(B)$ of a burst sequence B, to be the size of the largest subset of bursts which all overlap in time with one another (that is, the earliest burst ending time in the set is later than the latest burst starting time in the set). A sequence of bursts B can be scheduled without delaying any burst if and only if the number of channels on the link is at least equal to $W(B)$. We would like to have a link scheduling algorithm that would schedule a sequence of bursts without delaying any burst, so long as $W(B)$ is no larger than the number of available channels.

A horizon scheduler can schedule a burst sequence $B = \{b_1 = (r_1, t_1, \ell_1), \ldots, b_n = (r_n, t_n, \ell_n)\}$ using no more than $W(B)$ channels if the bursts arrive in the same order as the burst header cells. However, we can allow some misordering of bursts and still achieve this level of performance. In particular, the horizon scheduler uses at most $W(B)$ channels if for all $i < j$, $t_i < t_j + \ell_j$. That is, we get good performance if no burst b_j precedes another burst b_i with $i < j$ by more than the length of b_j. In a burst switching system that does not provide any burst storage, this condition can usually be satisfied, making horizon scheduling a good approach for such systems. On the other hand, the condition can be violated in systems where bursts are stored, in order to avoid discarding.

These observations suggest an alternative that can perform better than horizon scheduling in more general situations. Rather than process bursts as soon as their burst header cells arrive, one can delay the scheduling of bursts, and then process them in the order of expected burst arrival, rather than the order in which the burst header cells arrive. Essentially, as the burst header cells arrive, we insert them into a resequencing buffer, in the order in which the bursts are to arrive. A horizon scheduler then processes requests from the resequencing buffer. The processing of a request is delayed until shortly before the burst is to arrive, reducing the probability that we later receive a burst header cell for a burst that will arrive before any of the bursts that have already been scheduled. More precisely, we define a parameter Δ and schedule a burst with arrival time t_i at time $t_i - \Delta$. Resequencing buffers developed for ATM switching can be used for this purpose. See, for example, references [1, 2]. Figure 2 illustrates this approach.

By processing the requests out-of-order, we can get good performance, for a much wider class

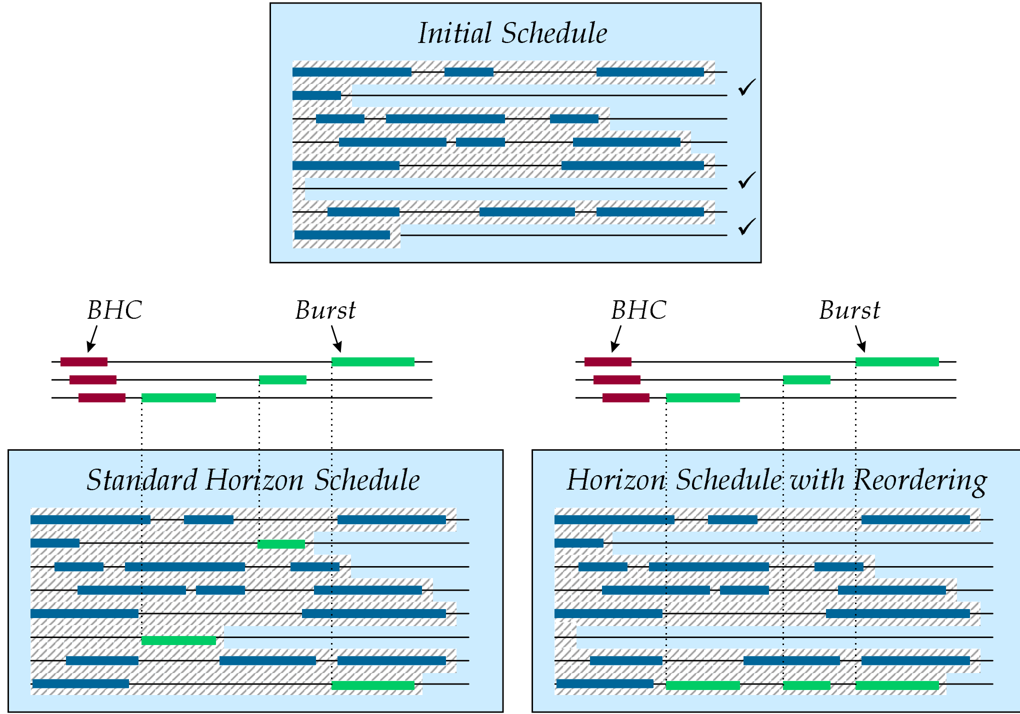

Figure 2: Horizon Scheduling with Reordering

of burst sequences than we can with an ordinary horizon scheduler. In particular, we can schedule a sequence $B = \{b_1 = (r_1, t_1, \ell_1), \ldots, b_n = (r_n, t_n, \ell_n)\}$ using no more than $W(B)$ channels so long as for all $i < j$, either $t_i < t_j + \ell_j$ or $s_j \leq t_i - \Delta$; that is, the request for burst b_j arrives before burst b_i is scheduled.

While out-of-order scheduling of bursts allows us to improve on the performance of horizon scheduling, it has an unfortunate side-effect. Since the scheduler does not assign bursts to channels until shortly before the bursts are to go out, we cannot know if a burst will be accepted or not until long after the burst header cell's arrival. In some situations, this information is needed in order to make other resource allocation decisions. What is needed, in these situations, is a way of deciding ahead of time, *when* a burst can be transmitted, while delaying the actual assignment of the burst to a particular channel.

This leads to a *split processing* algorithm that separates *burst scheduling* from *channel assignment*. First, the algorithm decides when an arriving burst should go out. It then makes an entry for that burst in a reordering buffer which is processed in the order in which the bursts are to be sent to the output link. Entries in the reordering buffer are processed shortly before bursts are to go out, using horizon scheduling to assign bursts to specific channels. This approach allows us to schedule bursts using a minimum number of channels, but also enables other resource allocation decisions to be made at the time a burst header cell arrives, rather than delaying them until the burst is about to be forwarded. This is critical in multistage networks, where it is important to forward burst scheduling

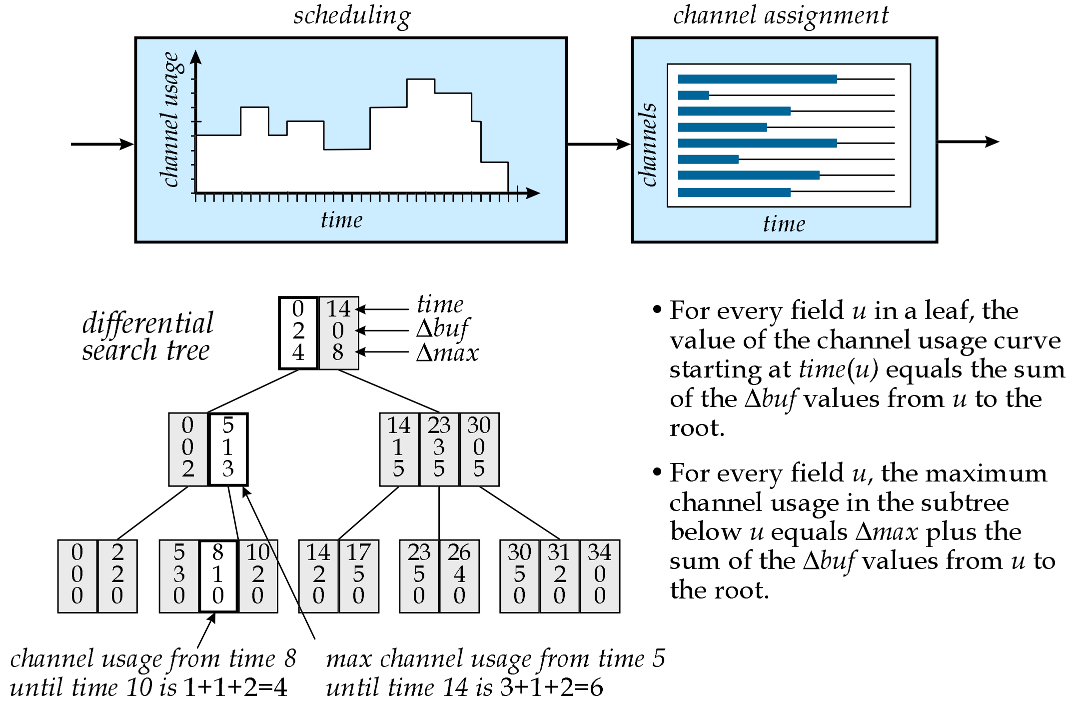

Figure 3: Split Processing Algorithm for Link Scheduling

requests from stage to stage when the request first comes in, so that the downstream burst switch elements can allocate the resources they need to accommodate the burst.

To implement the burst scheduling step, we use a data structure that represents the *link usage curve*, which specifies the number of channels that are scheduled to be in use at every future time. When a burst header cell arrives, the algorithm consults the link usage curve to determine if the link is completely occupied at any time between the arrival of the burst and its completion. If not, the burst can be forwarded directly to the output, as soon as it arrives. Otherwise it will have to be delayed. The link usage curve is represented with a *differential search tree*, as illustrated in Figure 3. (This data structure is a slightly simplified version of the data structure that we plan to use to manage memory for burst storage. This was described in our previous progress report [6].)

Even with split processing, we still need a mechanism to decide when to schedule the transmission of a burst that must be delayed. Ideally, if an arriving burst must be delayed, we would like to insert it into the schedule at the earliest possible time. That is, we need to find the earliest time interval that is at least as long as the arriving burst duration and during which the buffer usage curve is always less than the number of channels on the link. It is possible to find this earliest time interval using the differential search tree, but it is difficult to do it quickly. Indeed, in the worst-case one may have to examine the entire search tree in order to find the earliest time at which the burst will fit. Moreover, there does not appear to be an alternative data structure that would allow an arriving burst to be quickly mapped to the earliest schedule gap that could accommodate it.

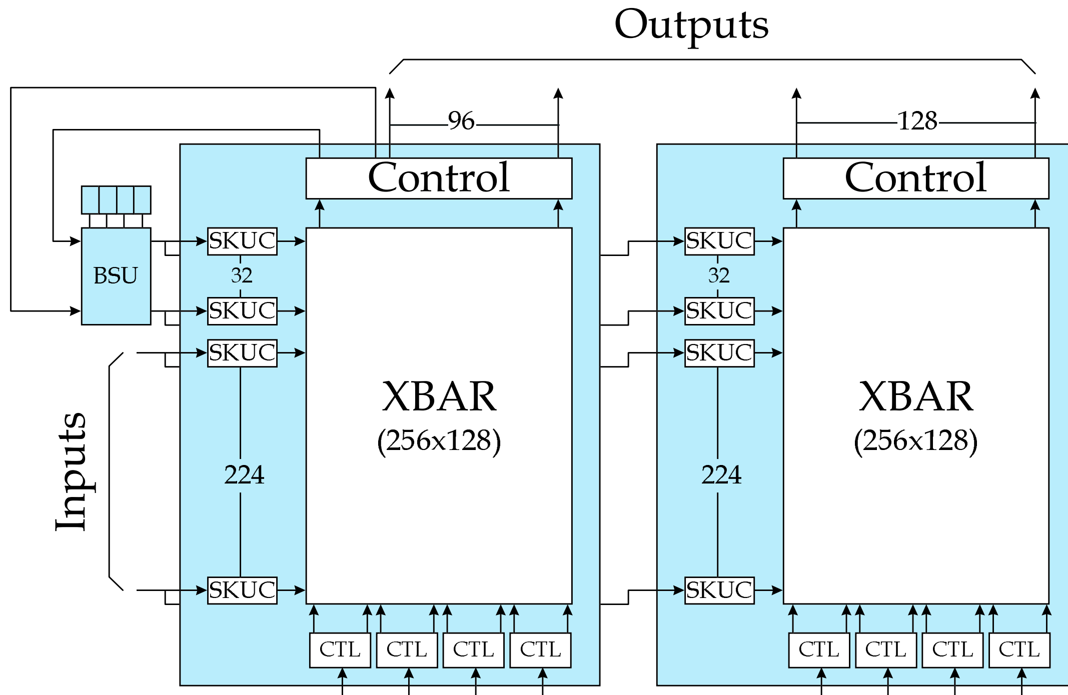

Figure 4: Crossbar for Burst Switch Data Path

A simple alternative is to maintain a single horizon variable for the entire outgoing link. The value of this variable is the *latest* time when the channel usage curve is equal to the number of channels on the link. When a burst arrives, we first use the differential search tree to determine if it can be scheduled for transmission without delay. If it cannot, we schedule it for transmission starting at the time given by the horizon variable. Since we know that the link is never completely used following the horizon value, it will always be possible to schedule the burst starting at that time. We then update the differential search tree and the horizon value.

This approach to scheduling bursts for transmission can produce suboptimal results if there are multiple time intervals where all the channels are in use, separated by time intervals when not all the channels are in use. However, it appears likely that for typical traffic this approach will yield results that are almost as good as can be obtained with an ideal scheduling algorithm. We plan to test this hypothesis using simulation. If the results do show a significant difference between these two alternatives, we will also explore intermediate options in which the scheduling algorithm keeps a separate record of some small constant number of maximal time intervals during which the link is not fully occupied. This will allow scheduling of arriving bursts in these *non-full periods*. So long as the number of such periods that must be tracked is kept small, a hardware scheduler can do the necessary matching operations in parallel, allowing a fast and practical implementation.

2. Crossbar Design

The burst switch element in the prototype burst switch supports seven inputs and outputs, with 32 channels each, where the channels operate at 1 Gb/s. It also provides the ability to route up to 32 bursts at a time into and out of its shared burst storage unit. This requires a 256×256 crossbar.

To provide the requisite switching capability, we are currently designing a crossbar component with 256 inputs and 128 outputs. A pair of these chips can be used together to form a 256×256 crossbar. To achieve the required 1 Gb/s link rate, we are using nine such pairs operating in parallel at a clock rate of 150 MHz. The chips are to be fabricated in a .25 micron CMOS process using a laser programmable gate array fabrication technique.

The crossbar components incorporate some specific features, not found in commercial crossbar components but required for the burst switch prototype application. The first of these features is a per-input skew compensation circuit that automatically compensates for data and clock skew between different components in the system. The approach used is very robust, and simple enough that 256 copies of the skew compensation circuitry can be incorporated in a single chip without consuming an excessive amount of the chip area. A transmitting circuit formats the data to be sent as a sequence of 32 bit words, preceded by a two bit start pattern (10) and followed by one or more 0 bits. When the receiving skew compensation circuit detects the start pattern for a word, it passes the arriving data through one of three alternative delay paths in order to align the data with the local clock to ensure that the arriving data bits are sampled when the values are stable. An unusual feature of the skew compensation circuit is that it chooses one delay path for use following rising data transitions and another for use following falling data transitions. By explicitly accounting for the difference in rise and fall times typical of CMOS circuits, it can select the best sampling point for each, allowing significantly better performance than if the same sampling point is used for both rising and falling transitions.

The second unusual feature of the crossbar design is that it divides the control circuitry used to configure the crossbar, to enable the different Burst Processors in the control section of the Burst Switch Element to operate independently of one another. Specifically, for each group of 32 outputs of a crossbar chip, there is a separate control circuit. Using this control circuit, a Burst Processor can connect any of the 256 crossbar inputs to any of the outputs in this set of 32.

Figure 4 shows the pair of crossbar chips used to implement the required 256×256 crossbar, along with the *Burst Storage Unit* and associated memory chips at the upper left of the diagram.

References

[1] Henrion, Michel A. R. “Resequencing System for a Switching Node,” U.S. Patent #5,127,000, August 1990.

[2] Turner, Jonathan S. “Data Packet Resequencer for a High Speed Data Switch,” U.S. Patent #5,339,311, August 1994 and U.S. Patent #5,260,935, November 1993.

[3] Turner, Jonathan S. “Terabit Burst Switching,” Washington University Technical Report, WUCS-98-17, 1998.

[4] Turner, Jonathan S. "Terabit Burst Switching Progress Report (12/97-3/98)," Washington University Technical Report, WUCS-98-16, 1998.

[5] Turner, Jonathan S. "Terabit Burst Switching Progress Report (3/98-6/98)," Washington University Technical Report, WUCS-98-22, 1998.

[6] Turner, Jonathan S. "Terabit Burst Switching Progress Report (6/98-9/98)," Washington University Technical Report, WUCS-98-30, 1998.