Terabit Burst Switching
Progress Report (3/98–6/98)

Jonathan S. Turner
jst@cs.wustl.edu

WUCS-98-22

December 29, 1998

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 (3/98–6/98)

Jonathan S. Turner
jst@cs.wustl.edu

This report summarizes progress on the Terabit Burst Switching Project at Washington University for the period from March 15, 1998 through June 15, 1998. Efforts during this period have concentrated on developing a detailed understanding of the control issues associated with the burst handling process. We have also continued the engineering efforts on the ATM switch and begun to address the interconnection of ATM and burst switches.

1. Prototype Burst Switch Plans

Figure 1 shows the planned configuration for the burst switch prototype. The system will include six I/O modules each capable of terminating a burst data link with 32 channels. For budgetary reasons, the different channels will be carried on separate fibers rather than on WDM channels in a single fiber. However, the prototype will treat the collection of fibers constituting a single "link" exactly as it would if they were carried on a single fiber. Each I/O module will contain the optics and transmission electronics for terminating 32 channels. We have tentatively selected 12 channel VCSEL-based components from Siemens for the optics, and a four channel transmission component from Vitesse (compatible with Fibre Channel) for the transmission line coding and clock recovery functions. In addition to these components, the I/O module will contain a four channel synchronization chip whose principal function is to delay data received from the link for a fixed time period before forwarding it to the interconnection network. The control section of the I/O module includes a *Time Stamp Component*, an ATM *Input Port Processor*, an ATM *Output Port Processor* and a *Burst Processor*.

The interconnection network for the prototype will contain a single *Burst Switch Element* (BSE) with seven input and output ports. The BSE is being designed to support multistage configurations, allowing for systems with total capacities of tens of Tb/s. The data path portion of the BSE contains a 256×256 crossbar, which is bit-sliced in order to provide the required aggregate throughput of 256 Gb/s. The data path also includes a *Burst Storage Unit* which provides shared storage space for bursts that cannot be immediately switched through to the proper output channel. The BSU will provide 16 MB of aggregate storage capacity. The control portion of the BSE contains an *ATM Switch Element* (ASE), a set of seven *Burst Processors* (BP) and a *Burst Storage Manager* (BSM). These components collectively control the switching of bursts, including diverting bursts to the BSU as necessary.

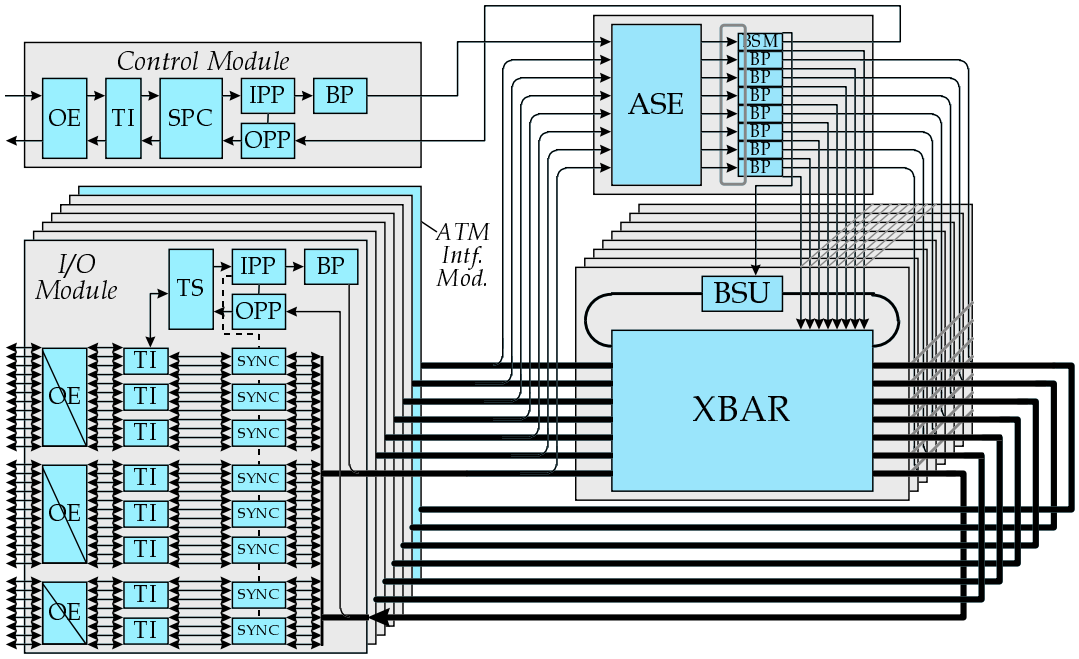


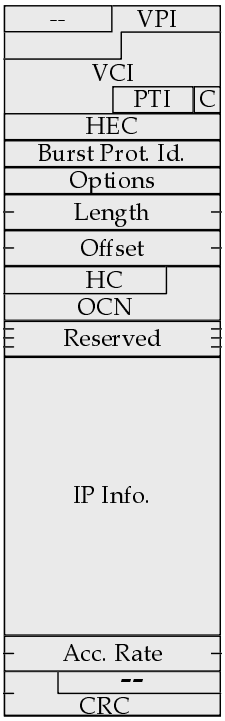
Figure 1: Prototype Burst Switch Architecture

The prototype will also include an ATM interface card that will allow data from an ATM network to be propagated through the burst switch. The interface will form cells from different virtual circuits into bursts for transmission through a burst network, and will convert the data back to an ATM cell stream on the output side.

To provide high level control of the burst switch, a control module containing a general purpose processor will also be provided. The software provided for this processor will provide a simple network management interface that can be used to configure the system.

2. Burst Header Cell Format

Figure 2 shows the planned format of the *Burst Header Cell* (BHC) that will be used in the prototype burst switch. To enable use of existing ATM components in the control section of the switch, a standard ATM cell format will be used. Burst switching control cells will all have an ATM Payload Type of 110_2 (the resource management cell type) and the first byte of the payload will be used to distinguish burst protocol cells from other ATM resource management cells. A one byte options field will be used to distinguish different routing options, including virtual circuit routing, IPv4



Notes

- Standard ATM cell header with PTI=110_2 for burst control cells
- Burst protocol identifier - identifies burst control cells
- Options - burst handling options (virtual circuit, IPv4, IPv6)
- Length - burst length in 16 byte words (1 MB max)
- Offset - number of ns from first bit of BHC to first bit of burst
- Hop Count (HC) - allowed number of hops remaining to destination
- Optical Channel Number (OCN) - channel used by data burst
- Reserved field used by burst switches for local control
- Access rate (AR) gives rate at which burst was received at entry to burst network; used to pace burst when forwarded to access net.
- IP information field
  - » for IPv4 includes source and destination addresses, port numbers and protocol field
  - » for IPv6 includes source and destination addresses
- User field - available for end-to-end information (8 B)

Figure 2: Burst Header Cell Format

routing and IPv6 routing. At this time, we are not planning to implement the IP routing options, but are making provisions in the cell format to facilitate later addition of these options.

The *Length* field gives the length of the burst in multiples of 16 bytes and allows for a maximum burst length of 1 MB. The *Offset* field specifies the number of nanoseconds between the first bit of the BHC and the first bit of its associated burst. The offset field is adjusted by various components within burst switches to reflect variations in control delay in different components. In general, a component updates the offset field to give its best estimate of the actual offset as it forwards a BHC to the next component either within a switch or in a subsequent switch. Each component in a burst network is responsible for making adjustments necessitated by differences in the delay of the control and data channels leading from an upstream component, as well as any local differences in the delay.

3. Scheduling Future Operations

To handle short bursts with acceptable efficiency, the control mechanisms in burst switches must be capable of assigning channels to bursts for periods as short as a few microseconds. Since queueing

delays within the control portion of a burst switch may be as large as 10 μs, burst switches must have the ability to schedule the allocation of resources at future times. These resources include both channels within communication links and memory used to store bursts that can't be forwarded immediately.

This type of resource allocation is very different from the usual resource allocation done within computer systems and networks. In typical systems today, the resource allocation mechanisms only keep track of what resources are currently in use or currently idle, and assign resources when a resource request is made. This is completely satisfactory in situations where resources are used as soon as they are allocated, or where the time period between allocation and use is much smaller the period during which the resource is used. In burst switching however, these conditions do not hold. The time period over which a resource is used may be comparable or even smaller than the time between the arrival of a burst header cell (when an allocation decision must be made) and the arrival of the burst. In situations like this, the resource allocation mechanisms must project resource usage into the future, so that they can determine if resources will be available at the time a burst actually arrives. In burst switches with 2.4 Gb/s channels, scheduling decisions must be made within the time it takes to receive a single cell (roughly 170 ns). This requires invention of new algorithms and mechanisms for resource scheduling.

3.1. Scheduling Channels in Burst Switch Data Links

The assignment of bursts to channels within a burst switching system turns out to be the most straightforward of the scheduling operations that must be performed in a burst switch. A Burst Processor (BP) must maintain a schedule for each of its output channels, that describes those time periods when the channel will be busy and those when it will be free. When a Burst Header Cell (BHC) comes in, the BP uses the offset information stored in the cell to determine when, relative to the current time, the burst will arrive. It also uses the burst length field to calculate when the end of the burst will arrive. It then consults the schedules for all of its channels to determine if any of them will be free during the entire time period that the burst needs a channel. If it finds that a channel is available, it assigns the burst to that channel and modifies the channel data structure appropriately to reflect the allocation of resources.

While the general scheduling operation is conceptually straightforward, it is difficult to see how to make it fast enough to handle high performance burst switching. To support fast scheduling, it appears necessary to trade off scheduling accuracy for speed. This can be done by reducing the amount of schedule information that is maintained and making scheduling decisions in relation to this approximate schedule. Instead of keeping track of every single busy and idle interval for every channel, we can keep track of only a subset of the idle intervals for each channel, and assume that in all other time periods, the channel is busy.

The simplest version of this approach is called *horizon scheduling*. Here, the schedule for channel i consists of a single number τ_i which denotes the latest time in the future at which the channel is busy. An arriving burst can be assigned to channel i, if and only if the burst will not arrive until after τ_i. If the burst is assigned to channel i, then τ_i is changed to the time at which the burst will finish. This algorithm is simple enough for a fast hardware implementation. To minimize wasted channel bandwidth, the controller can select the last channel to become available after the burst arrival time.

Because horizon scheduling does not keep track of idle periods before the horizon, it can waste channel bandwidth unnecessarily. We can improve its performance by adding back more scheduling information. In *single gap* scheduling, the schedule keeps track of both the horizon and one idle period before the horizon. This keeps the total scheduling information small enough to make fast hardware implementations feasible, and may produce significant performance benefits. In single gap scheduling, when a burst is scheduled to a channel, the controller must decide which new idle period to keep track of, selecting larger idle periods over smaller ones, but also giving preference to later idle periods over earlier ones (since later idle periods are more likely to be used than earlier ones). Clearly, single gap scheduling can be extended to two or more gaps. Simulation studies are needed to fully determine the cost/benefit trade-offs associated with large numbers of gaps.

There is another approach to approximate scheduling that is worthy of consideration. In this approach, the scheduler maintains a data structure containing triples of the form $[i, t_1, t_2]$. Each such triple means that channel i is idle from t_1 to t_2. To schedule an arriving burst, we need to be able to quickly determine if the time period during which the burst needs a channel will fit within any of the available idle periods. One way to do this is to first partition the triples according to their length. Within each subset, the triples can be organized into a search tree data structure that is sorted according to the start time of the idle period. When a BHC is received, we search all of the search trees corresponding to time intervals that are at least as long as the time interval needed for the arriving burst. The worst-case time required for the search is $O((\#ofsets)log(\#ofidleperiods))$. To reduce the running time, the intervals can be partitioned into just a small number of sets. A good partitioning strategy is to partition the intervals into time periods of exponentially increasing length. The implementation of this approach requires a small random access memory. For a system with OC-48 channels, there will be time to perform only about 20-30 memory accesses per scheduling operation. This may be sufficient, but more detailed study is needed to determine if this approach is practical and if its complexity is justified by the performance benefits.

3.2. Burst Storage Scheduling

In addition to the link resources, a burst switch must also manage the allocation of burst storage resources. This is somewhat more complex because there are two dimensions to consider, time and storage space. The upper left portion of Figure 3 shows a plot of buffer usage vs. time for a burst storage unit that is receiving and forwarding bursts. As new bursts come in, they consume additional storage space releasing the storage later as they are forwarded to the output. The plot shows how a new burst arrival modifies the buffer usage curve. One way to schedule a burst storage unit is to maintain a data structure that effectively describes the buffer usage curve for the burst store. For high performance, we need a data structure that supports fast incremental updating in response to new burst arrivals.

This can be done by extending a search tree data structure such as a 2-3 tree (2-3 trees are a special case of B-trees [2]). The upper right hand portion of Figure 3 shows a 2-3 tree that represents the buffer usage curve at the top left. The leaves in this search tree contain pairs of numbers that define the left endpoints of the horizontal segments of the buffer usage curve. Each interior node of the search tree contains a time value equal to the largest time value appearing in any of its descendant leaves. These time values allow one to determine the range of times represented by a given subtree without explicitly examining all the nodes in the subtree.

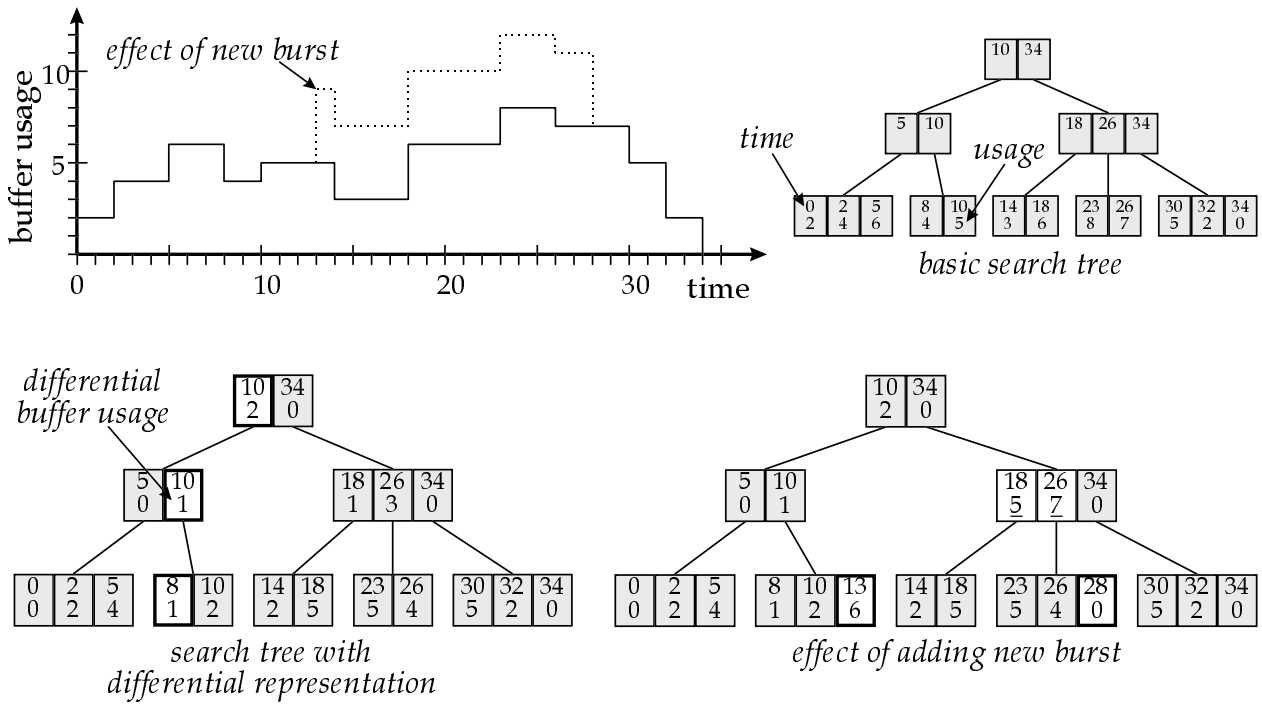


Figure 3: Burst Store Occupancy vs. Time

To allow fast incremental updating of the 2-3 tree when adding a new burst, the basic structure needs to be extended by using a *differential representation* for the buffer usage values. This is illustrated at the bottom left of Figure 3 which shows a search tree that represents the same buffer usage curve. In this search tree, buffer usage must be computed by summing the values along the path from the root of the tree to a value-pair at a leaf. Consider for example, the three value-pairs that are highlighted in the figure. The sum of the buffer usage values along this path is equal to 4, which equals the buffer usage of the corresponding value-pair at the "bottom" of the original search tree.

The differential representation is useful because it allows rapid incremental updating in response to a new burst arrival. This is illustrated at the bottom right of Figure 3. Here, two new value-pairs (highlighted) have been added to reflect the addition of the burst indicated in the top left portion of the figure. Note that two buffer usage values in the righthand child of the tree root have also been modified (these values are underlined) to reflect the addition of the new burst. Using this sort of differential representation, the data structure can be modified to reflect an additional burst in $O(\log n)$ time, as opposed to $O(n)$ time with the original representation.

Unfortunately, to test if a newly arriving burst can be stored without exceeding the available storage space, we may still need examine a large portion of the search tree, resulting in unacceptably large processing times. However, this extra processing time can be avoided if we are willing to accept some loss in the efficiency with which the buffer is used. Let t_1 be the minimum time

between the arrival of a BHC and its corresponding burst and let t_2 be the maximum time between a BHC and its corresponding burst. For scheduling purposes, we can ignore the portion of the buffer usage curve that precedes the current time plus t_1. More significantly, the buffer usage curve is monotonically non-increasing for all values of t greater than the current time plus t_2. If $t_1 = t_2$ then we can completely ignore the non-monotonic part of the buffer usage curve and focus only on the monotonic part (so in the example curve in Figure 3, we can ignore the portion before time 28 if the minimum and maximum offsets are equal).

These observations suggest a scheduling algorithm that maintains a monotonic approximation to the buffer usage curve and uses this approximation to make scheduling decisions. The use of this approximation will occasionally cause bursts to be discarded that could safely be stored. However, so long as the gap between the minimum and maximum offsets is small compared to the typical "dwell time" of stored bursts, these occurrences should be rare.

With a monotonic buffer usage curve it is easy to determine if an arriving burst can be accommodated. If there will be enough storage space to handle the "head" of the burst then we are guaranteed that there will be enough to handle the remainder (since the approximate buffer usage curve never increases). Thus, we only need to perform a single comparison to check if an arriving burst can be handled. Once we've determined that a burst can be safely handled, the incremental updating can occur in $O(\log n)$ time, as described above.

To achieve gigabit processing rates for large numbers of stored bursts, 2-3 tree may be too slow. Substantial additional speedup can be obtained using a general B-tree. In a B tree [2], each node can have between B and $2B-1$ children, allowing a substantial reduction in the depth of the search tree. In a Burst Storage Manager using a B-tree to manage the storage space, hardware mechanisms can be used to enable parallel processing of individual B-tree nodes.

4. Skew Management for High Efficiency Burst Switching

To operate a burst-switched network efficiently, it's important to maintain tight control over the skew between a Burst Header Cell and its associated burst. Uncertainty in the relative timing of the BHC and the burst translates directly into overhead, since switches must make take the timing uncertainty into account when making connections, so as to avoid clipping user data from the start or end of a burst.

When a user terminal has a burst to send, it first sends a BHC on the control channel of its access link and shortly afterwards, it sends the burst. The BHC includes an *offset* field that specifies the time between the transmission of the first bit of the BHC and the first bit of the burst. When BHCs are processed in burst switches, they are subject to variable processing delays, due to contention within the control subsystem of the burst switch. To compensate for the delays experienced by control cells, the data bursts must also be delayed. A fixed delay can be inserted into the data path by simply routing the incoming data channels through an extra length of fiber. To maintain a precise timing relationship between control and data, BHCs are *time-stamped* within burst switches. The timestamps are used as the burst progresses through the switch to help keep track of variable delays. The offset field is updated at various points to reflect these variations in delay.

As a burst progresses through a burst network, there is some inevitable loss of precision in the offset information. Timing uncertainty arises from two principal sources. First, signals using different wavelengths of a WDM link travel at different speeds down the fiber. While the absolute magnitudes of these differences are not large (under 1 μs for links of 1000 km), they are significant enough to be a concern. Fortunately, they can be compensated at the receiving end of a link, if the approximate length of the fiber is known. Given the length information, the difference in the delay between the control channel and each of the data channels can be calculated and the receiving control circuitry can adjust the offset associated with an arriving burst, accordingly. Note, that this does not require inserting optical delays in the data channel. The control subsystem, simply adjusts the offset value in the BHC to reflect the differences in delay across channels. While this compensation is not perfect, it can be made accurate enough to keep timing uncertainties due to this cause under 100 ns on an end-to-end basis in a wide area network.

Another source of timing uncertainty is clock synchronization at burst switches along the path in a network. When data is received on the control channel at a burst switch, the arriving bit stream must be synchronized to the local timing reference of the burst switch. There is some inherent timing uncertainty in this synchronization process, but the magnitude of the uncertainty can be reduced to very small values using synchronizers with closely-spaced taps. Even with fairly simple synchronizer designs, it can be limited to well under 10 ns per switch. In an end-to-end connection with ten switches, this results in an end-to-end timing uncertainty of 100 ns.

Within switches, it's possible to avoid significant loss of timing precision, since all control operations within a switch can be synchronized to a common clock, and since the data path delays in a switch are fixed and can be determined with high precision. Thus, the end-to-end timing uncertainty is essentially the sum of the uncertainty due to uncompensated channel-to-channel delay variation on links and the uncertainty due to synchronization. If the end-to-end uncertainty is limited to 200 ns then bursts with durations of 2 μs can be handled efficiently. This corresponds to 600 bytes at 2.4 Gb/s.

It's possible to avoid the necessity for tight end-to-end timing control if the switches interpret the information sent on the data channels so that they can explicitly identify the first and last bit of a burst. While this is reasonable to do in electronic implementations, it is more difficult in systems with optical data paths. In addition, it constrains the format of data that can be sent on the data paths, at least to some degree.

References

[1] Chaney, Tom, J. Andrew Fingerhut, Margaret Flucke and Jonathan Turner. "Design of a Gigabit ATM Switch," *Proceedings of Infocom*, April 1997.

[2] Cormen, Thomas, Charles Leiserson, Ron Rivest. *Introduction to Algorithms,* MIT Press, 1990.

[3] Siemens Semiconductor Group. "Parallel Optical Link (PAROLI) Family," http://w2.siemens.de /semiconductor/products/37/3767.htm, 1998.

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