Congestion-free Routing of Streaming Multimedia Content in BMIN-based Parallel Systems

Harish Sethu

Department of Electrical and Computer Engineering
Drexel University
Philadelphia, PA 19104, USA
sethu@ece.drexel.edu

Abstract. Multimedia servers are increasingly employing parallel systems for the retrieval, scheduling and delivery of streaming multimedia content. However, given non-zero blocking probabilities in interconnection networks of most real parallel systems, jitter-free constant-rate delivery of streaming data cannot often be guaranteed. Such a guarantee can be best accomplished through the elimination of all congestion in the network. In this paper, we focus on folded Benes BMIN networks and achieve a fast convergence to congestion-free transmissions using an approach that combines flow-based adaptive routing with a distributed version of Opferman's looping algorithm. This combined approach significantly reduces the buffering requirements at the receiving stations. In addition, it also achieves a low start-up delay, important for interactive web-based multimedia applications.

1 Introduction

A popular trend in the architecture of multimedia servers and other networked multimedia systems is the use of commercial parallel computing systems such as the IBM RS/6000 SP as distribution engines, co-ordinating the retrieval, scheduling and delivery of streaming multimedia content. New, popular and commercially promising applications such as web-based interactive communications have further fueled this trend. This is driven in part by the fact that parallel system interconnects and the associated hardware interfaces and communication subsystem software services offer high-bandwidth network-level access to system I/O at very low latencies. In many networked multimedia systems, aggregation and/or striping is achieved before the data reaches the network, and the primary issue is the guaranteed-rate jitter-free delivery of continuous data streams between sets of source and destination end-points of the network, with no source paired with more than one destination or vice versa. Guaranteed-rate jitter-free delivery is best accomplished through elimination of all congestion in the network. This provides our motivation to consider the problem of congestion-free routing of multiple streaming multimedia traffic across packet switched multistage interconnection networks popular in parallel systems.

In this paper, we focus on a class of Bidirectional Multistage Interconnection Networks (BMINs) used in several offerings of commercial parallel systems [1,2]. Within this class of topologies, we specifically consider those topologies which possess the rearrangeable non-blocking property. A rearrangeable non-blocking topology offers the potential for congestion-free routing; that is, given the knowledge of all exclusive source-destination pairs in the network, one can always use a set of rules by which a congestion-free set of paths can be established between each source-destination pair. The looping algorithm by Opferman and Tsao-Wu for Benes networks is a good example of such a rule [3]. Centralized control routines can easily implement such algorithms in software. However, large delays associated with software involvement are not suitable for interactive web-based multimedia servers. This provides a motivation for a technique that allows the start of transmissions immediately upon request, and which subsequently seeks a fast convergence of the routing to a congestion-free state. Such a technique serves an additional goal of minimizing the disruptions and the associated jitter and/or buffer overflows that occur when there are changes in the pairings between traffic sources and destinations. In this work, we assume that the transient congestions that may occur due to control packets are inconsequential.

In this paper, we present an approach that can be implemented in hardware in the switching elements of parallel systems and which achieves congestion-free, and thus, jitter-free transmission within a short period of time after the start of transmissions. Our method combines flow-based adaptive routing (as opposed to adaptively routing each individual packet) and a distributed implementation of Opferman's looping algorithm modified for the class of networks under consideration. This technique enables a convergence to congestion-free routing in less time than it takes to even complete the execution of the modified looping algorithm. This is because, as the algorithm establishes routes for source-destination pairs, the flow-based adaptive routing begins to achieve greater stability. The two algorithms in tandem, achieve fast convergence to congestion-free routing, thus reducing the buffering requirements at the receiving portals of the multimedia streams, while also achieving a very small delay in starting up the transmissions.

In Section 2, we describe the topology of folded Benes networks. Sections 3 and 4 describe flow-based adaptive routing and the distributed version of the looping algorithm, respectively. Section 5 briefly discusses simulation results, and Section 6 concludes the paper.

2 Folded Benes networks

BMINs may be thought of as Benes networks or symmetrically extended Banyan networks which are folded in the middle to create bidirectional links between switches. Some of IBM's RS/6000 SP system topologies [1,5] and the CM-5 data router [2], are examples of commercial realizations of this family of network topologies. Recognizing that many real BMINs are *folded Benes* networks, we refer to them as such and limit discussion, in this paper, to this class of BMINs.

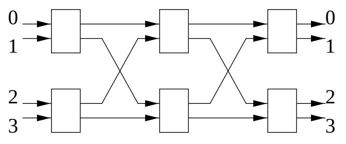

Figure 1(a). A 4-by-4 Benes network

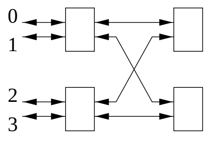

Figure 1(b). An FB(4, 2) network

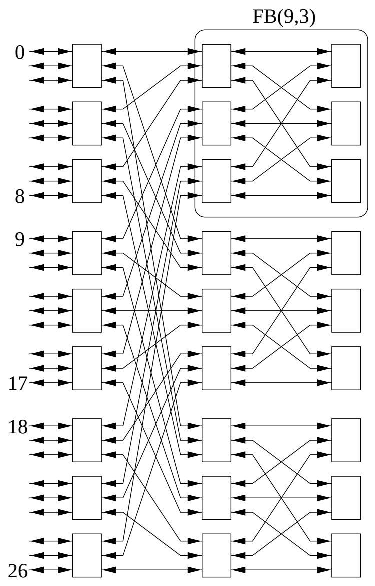

Figure 2. An FB(27, 3) network

Denote by $FB(N, b)$, an $N \times N$ folded Benes network using $2b \times 2b$ switches, where N is a power of b. Figure 1(a) shows a 4×4 Benes network using 2×2 switches, which when folded yields an $FB(4, 2)$ network shown in Figure 1(b), i.e., a 4×4 folded Benes network using 4×4 switches. This is shown only for the sake of illustration—it may be noted that an $FB(4, 2)$ network does not quite make sense since one may just as well use a single 4×4 switch. Figure 2 shows an $FB(27, 3)$ network.

3 Flow-based adaptive routing

In this work, we define a flow as the string of packets that originate from the same source and are headed to the same destination. This concept of flows is similar to that used in some implementations of IP over ATM to achieve connection-oriented behavior in a connectionless protocol [4].

The technique of flow-based adaptive routing merges the two concepts of flow switching and adaptive routing. In our implementation of this technique, the switches maintain a table of active flows, henceforth referred to as the *flow table*. Each entry in this table associates an output port with a flow. Packets belonging to a given flow are all queued for transmission at the output port specified by the entry in this table. As far as possible, the switch attempts to ensure that no output port is associated with more than one flow in the flow table. This is in contrast to traditional packet switching in which each packet is independently routed. Unlike in the popular use of the concept of flow switching in the Internet, our technique allows adaptivity in the path taken by the flow. For example, when a given flow is experiencing congestion, the switch that observes this congestion changes the entries in the flow table seeking to avoid further congestion. This triggers corresponding changes in the tables for other switches downstream as well.

Whenever a switch detects that one of its output queues corresponding to a certain flow, f, has more than a certain threshold number of packets in it, it seeks to change the output port entry corresponding to this flow. In folded Benes networks, the first half of the hops in the path of a flow can be without congestion, since there are as many valid output ports as there are flows seeking to access these output ports [6]. On the return path, i.e., during the second half of the hops in the path of a flow, destination-based contention causes congestion. In this return path, often, two flows have to share a link to reach their respective destinations. Changing the flow table entries in such a way that f's output port is now associated with another flow that is headed to a destination maximally distant from that of f, has the potential to reduce downstream destination-based contention. This is based on the premise that paths to distant nodes will likely diverge in subsequent hops of the packets, and therefore will not cause congestion. The probability of downstream congestion, therefore, can be reduced by switching the output port entry of the flow associated with the congested output port with the output port entry of a flow with a destination as distant as possible from that of the flow experiencing the congestion.

The following algorithm captures the core of our implementation of flow-based adaptive routing. This algorithm determines the destination output port of each new packet that arrives at the switch. In the following, F is defined as the set of all flows that pass through the switch executing the algorithm. T is the threshold number of packets necessary in an output queue, in order for the algorithm to consider switching the output ports in the flow table entries.

```
1. $f =$ flow the packet belongs to;
2. if flow table entry of $f$ is empty, then,
     Set a random idle output port as the output port entry of $f$;
3. if output queue of output port entry for $f$ has more than $T$ packets, then,
     Find $g \in F$ such that $|destination(f)$ - $destination(g)|$ is maximum;
     Pick a random number, $r$, between 0 and 1;
     If $r < \alpha$, exchange the output port numbers in entries for $f$ and $g$;
4. Output port of new packet = output port in flow table entry of $f$;
```

An important aspect of adaptive routing is the potential for oscillations in the paths taken without achieving the desired convergence to a congestion-free set of paths. This phenomenon requires that we use a dampening factor, $\alpha \leq 1$, as in Step 3 of the above algorithm, so that flow table entries are not switched during each cycle that the corresponding queue length is above the threshold number. Instead, a switch in the entries is made only with a probability α when the threshold is reached.

The above routing technique can guarantee a very high rate but not the 100% of the input rate necessary for jitter-free delivery of streaming data. Besides, the routing using this algorithm is not stable since it does not necessarily converge to a congestion-free set of paths for any given permutation of source-destination pairs. The distributed algorithm described in the next section, in conjunction with flow-based adaptive routing achieves the desired convergence.

4 The distributed modified looping algorithm

The looping algorithm, as first proposed by Opferman [3], was for Benes networks with 2×2 switches. In this paper, we present a generalization of the algorithm to $b \times b$ switches, modify it to work in conjunction with the flow-based adaptive routing algorithm, and apply it to folded Benes networks. The algorithm presented here assumes that flow-based adaptive routing is in effect during the time that the looping algorithm is executing. This is a crucial aspect of the algorithm, which helps to simplify the distributed algorithm while also allowing a low start-up delay. To simplify the presentation of the looping algorithm, we assume the worst-case scenario where each node has multimedia streams to receive as well as to send; recall that, in a folded Benes topology, each end-point of the network can be both a receiving as well as a sending portal.

We define an entry in the flow table in a switch as a *forward* entry if the source node of the flow is lesser number of hops away from the switch than the destination node of the flow. Similarly, define an entry in the flow table in a switch as a *backward* entry if the destination node of the flow is lesser number of hops away from the switch than the source node of the flow. Thus, given a flow with a 5-hop path from node 0 to node 26, the entries corresponding to this flow in the first two switches in the path are forward entries while the entries in the last two switches in the path are backward entries. It is possible that an entry is neither a forward nor a backward entry. In a folded Benes network, the same switch can be both a first switch in the path of some flow, and the last switch in the path of some other flow. Note also that a first-stage switch merely means a switch directly connected to the nodes. A second-stage switch is one that has at least one other switch between itself and a node. In general, a stage-s switch has $s - 1$ switches between itself and a node.

As in the original looping algorithm, congestion-free paths are established in a recursive fashion. In the first application of the algorithm, entries are locked only in the first-stage switches of the network. The algorithm is next applied within each of the *FB* subnetworks interconnecting the first-stage switches; this results in the locking of entries in the second stage switches of the overall network. This recursive application of the algorithm continues on smaller and smaller subnetworks, until the entries in all stages are locked. The three steps enumerated below describe the algorithm executed to lock the entries in the first stage of the network or subnetwork under consideration.

1. Select a first-stage switch that already has a forward locked entry in the flow table, and with at least one unlocked forward entry. If no such switch is

found, select a first-stage switch with no locked forward entries. If no such switch is found either, the algorithm ends.

2. Lock the only unlocked forward entry, or a randomly chosen unlocked forward entry in the selected switch. Lock the corresponding backward entry of the flow in the last switch in the path of the flow. Now, select an unlocked backward entry in this switch, and go to Step 3. If there are no unlocked backward entries in this switch, go to Step 1.
3. Lock the selected unlocked backward entry in the switch. Now, in the first switch in the path of the corresponding flow, lock the forward entry. If there is an unlocked forward entry in this switch, select this switch and go to Step 2; otherwise, go to Step 1.

As shown in Figure 2, an $FB(N, b)$ network may be thought of as being divided into a first stage of $2b \times 2b$ switches, and a second (or last) stage of b $FB(N/b, b)$ subnetworks. In the first application of the algorithm, only the entries in the first-stage switches are locked. Consider a flow that has its corresponding first-stage switch entries locked in this first recursion. This does not lock the path of the flow within the $FB(N/b, b)$ subnetworks. However, from the nature of FB networks, the entry and exit points of the flow within the $FB(N/b, b)$ network are locked and thus the algorithm can now be applied within the b $FB(N/b, b)$ subnetworks. In the subsequent applications of the algorithm within the $FB(N/b, b)$ subnetworks, the entries in the first stage of the $N/b \times N/b$ subnetworks are then locked. When an entry is locked, no flow other than the one in the locked entry may use the output port associated with the locked entry.

We now describe the core aspects of the distributed modified looping algorithm. Assume the model of the $FB(N, b)$ network as shown in Figure 2, with the first stage of switches connected to the sources and destinations, and interconnected via b separate $FB(N/b, b)$ subnetworks. The algorithm begins in the first stage switches, which use short control packets to exchange information. The distributed algorithm begins in a designated switch in the first stage which generates a control packet; randomly chooses a flow that has a destination in another switch; and sends the control packet to this destination switch which is also a first-stage switch. Let the source of this flow be i, and the destination be $d(i)$. A lock is placed in the first-stage switches on the path from i to its destination, $d(i)$. Placing a lock on a path and associating it with a flow is equivalent to establishing a deterministic non-adaptive path for the flow through the corresponding switch. The switch connected to $d(i)$, upon receiving the control packet, responds with a new control packet headed to the switch connected to the source of packets arriving at a destination within this switch connected to $d(i)$. This control packet uses an $FB(N/b, b)$ subnetwork different from the one used by the previous control packet. This happens automatically because no more than one flow can use the output port in a locked entry, and by the nature of FB networks, a switch in the first stage is connected to each of the subnetworks by exactly one cable. Once again, the first stage switches in the path of the control packet establish locks associating the path with the flow. This pattern of choosing different subnetworks to reach different destinations connected to the same switch, is continued until all inputs and outputs are connected via paths that are locked. As far as possible, an attempt is made to merely lock entries, rather than rewrite entries as well as lock them.

Exceptions occur when, for example, in Step 1, a first-stage switch receives a control packet instructing it to lock an entry, which is the last unlocked entry in the switch. The algorithm now needs to continue at a new first-stage switch, as described in Step 1 of the recursive algorithm. We ensure this through using a bit-map of sources, within each control packet, to indicate which flows have already had entries locked in the first stage. For example, if a flow with source address S already has its entries locked, bit S in the length-N bit-map would be set to a 1. A switch can now determine which flow entries are not yet locked, even though all of its own flow entries are locked. An informational control packet can now be sent to the first-stage switch with an unlocked entry. When locks are being placed in the second-stage switches, i.e., within the first-stage switches of the $FB(N/b, b)$ subnetworks, bit $\lfloor S/b \rfloor$ in the length-(N/b) bit-map is set to a 1.

The three-step algorithm described above is a constructive proof that non-overlapping paths can be established in the first stage switches of an FB network. Now, note that an $FB(N, b)$ network is constructed out of a first stage of switches connected to b $FB(N/b, b)$ subnetworks. For example, Figure 2 shows that an $FB(27, 3)$ network has a first stage of switches and its subsequent stages are a set of 3 $FB(9, 3)$ subnetworks. Thus, after non-overlapping paths are established in the first stage switches of a $FB(N, b)$ network, non-overlapping paths can be similarly established in the first stage switches of the $FB(N/b, b)$ subnetworks. These first stage switches of the subnetworks are really the second stage switches of the overall network. Recursively applying the algorithm and thus locking flow table entries in each of the stages of the network leads to congestion-free routing of any given permutation of exclusive source-destination pairs.

5 Simulation results

Our simulation environment consists of output-queued switches, with an architecture that allows one switch hop in a minimum of 4 cycles in the absence of blocking. All input and output queues are set to a size of 4 packets. All links between switches can be traversed in exactly one cycle.

Our simulations indicate a fast convergence to congestion-free routing in the vast majority of randomly chosen permutations, even before all the flow entries are locked. However, the worst-case convergence time is what is relevant in the design of the system, especially with regard to buffering requirements. In a 2-stage $FB(16, 4)$ network, for example, the worst-case convergence time is equal to (i) the time taken to lock 15 paths (the last path is automatically determined by elimination) plus (ii) the time taken for 3 control packets to be sent to other first-stage switches when the sending switch has all its forward entries locked (part of Step 1 of the looping algorithm). This time depends on the input rate since the number of cycles it takes to traverse a path depends on the probability of blocking, which depends on the input rate. However, even at a 100% input load, our simulation of flow-based adaptive routing indicates that the time taken to communicate between two first-stage switches is never more than 12 cycles. Assuming a 4 nanosecond cycle time, this translates to an observed maximum convergence time of less than 1 μs. Note that, during this time, packets are being delivered to the destinations although at a slightly slower rate than the rate at which the cables in the network can send or receive data. The effective output rate varies with the chosen permutation; however, at a 100% input load, an effective output rate of at least 95% was achieved for all the permutations attempted in our simulations. Assuming that the data absorption rate is the same as the rate at which data is being sent from the senders, it should be expected that the receivers should buffer some data before beginning the absorption so as to achieve jitter-free reception. At 1 GB/s, using the earlier example of an $FB(16,4)$ network, this implies a buffering requirement of just 50 bytes. On the other hand, if one does not attempt convergence to congestion-free routing, the buffering required is unbounded since flow-based adaptive routing alone cannot guarantee stable congestion-free routing. The self-stable nature of this technique can similarly minimize the buffering requirements when disruptions occur due to the introduction of a new set of data streams replacing an older set.

6 Conclusion

In this paper, we have presented a mechanism that combines the concept of flow-based routing, which has shown to have worked very well in the Internet, with Opferman's looping algorithm, originally designed for circuit-switched telecommunication networks. We have added adaptivity in the model of flow-based routing, and used a distributed version of the looping algorithm modified for folded Benes networks and for use in conjunction with flow-based adaptive routing. This combined approach allows a fast convergence to congestion-free routing and therefore, jitter-free constant-rate delivery of multimedia streams, while reducing buffering requirement as well as the start-up delay.

References

1. Sethu, H., Stunkel, C. B., Stucke, R. F.: IBM RS/6000 SP Large-System Interconnection Network Topologies, *Proceedings of the 27th Int'l. Conf. Parallel Processing*, Minneapolis, August 1998.
2. Heller, S.: Congestion-Free Routing on the CM-5 Data Router, *Lecture Notes in Computer Science* **853** (1994) 176–184.
3. Opferman, D. C., Tsao-Wu, N. T.: On a class of rearrangeable switching networks—Part I: Control algorithms, *Bell Syst. Tech. J.* **50** (1982) 1579–1618.
4. Newman, P., Minshall, G., Lyon, T. L.: IP Switching—ATM under IP, *IEEE/ACM Transactions on Networking*, **6**(2) (1998) 117–129.
5. C. B. Stunkel, *et al.*:, "The SP2 High-Performance Switch," *IBM Systems Journal*, **34**(2) (1995) 185–204.
6. Y. Aydogan, *et al.*:, "Adaptive Source Routing in Multicomputer Interconnection Networks," *Proceedings of the 10th Int'l Parallel Processing Symp.*, April 1996.