Virtual Networks: Implementation and Analysis

by

Keith H. Randall

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author . . . .
Department of Electrical Engineering and Computer Science
October 19, 1994

Certified by . . . .
Charles E. Leiserson
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
F. R. Morgenthaler
Chair, Department Committee on Graduate Students

Virtual Networks: Implementation and Analysis

by

Keith H. Randall

Submitted to the Department of Electrical Engineering and Computer Science
on October 19, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Programming conventional parallel computers can be a difficult task because intricate measures are required to avoid deadlock. This problem exists because the network does not make any guarantees about the order that packets get delivered. Non-critical packets can delay critical packets by filling up all available queue space. When writing programs for these computers it is often advantageous to assume that there are several *virtual* networks connecting the processors instead of only one physical network. This strategy makes writing deadlock-free algorithms easy because different types of packets cannot block each other. It is not feasible to build a different physical network for each of these virtual networks, however. In this thesis, we build what we call a *virtual router*, a routing algorithm that can simulate many virtual networks on one physical network.

The virtual router is a similar concept to virtual memory. Virtual memory simulates many virtual address spaces on one physical address space. In the same manner, the virtual router simulates many virtual networks on one physical network. The virtual router guarantees that packets in one virtual network cannot adversely affect packets in another virtual network.

The virtual routing scheme we have developed uses Ranade's routing algorithm on a butterfly with N inputs along with a distributed algorithm to determine packet priorities. This distributed algorithm assigns priorities to packets in a time-dependent fashion and guarantees that each virtual network gets its fair share of network bandwidth. In particular, if V virtual networks are active, we show that each virtual network is guaranteed an $\Omega(1/V)$ fraction of the network bandwidth by proving that in each window of $\Theta(V \log N)$ time each virtual network gets exclusive use of the physical network for $\Theta(\log N)$ time. Furthermore, we show that the virtual router delivers the whole set of virtual networks in asymptotically optimal time.

Thesis Supervisor: Charles E. Leiserson

Title: Professor of Computer Science and Engineering

Contents

Chapter 1

Introduction

1.1 Networks

Networks for parallel computers have been improving steadily in recent years. Designers have been successful in improving the bandwidth and reducing the latency of networks to satisfy application demands. These designs give few if any guarantees about the order in which packets get delivered to their destinations, however. The only constraint commonly imposed is FIFO (First In, First Out) ordering between any source/destination pair. This lack of constraint allows network designers the largest possible leeway in deciding how to deliver packets so that they may deliver them quickly.

There is a problem with allowing any possible ordering of the packets, however. The problem that arises is that deadlock is possible if the wrong packets are chosen to get delivered. To illustrate this phenomenon, consider the following example. There are two types of packets. One is a *request* packet, and one is a *reply* packet. In a typical application, the request might be a request for the value of a memory location, and the reply might be the value stored in that location. In our example, requests are generated spontaneously by the processors in the network. When a processor receives a request, it looks up the appropriate value in its memory and sends a reply back to the requester. When the requester receives the reply, the transaction is complete.

At any point in time, the network may have several request packets and several reply packets pending. Suppose that the network decides to deliver only the request packets to their destinations. Then for each request packet that gets delivered, a reply packet gets generated. As long as no reply packet gets delivered, the number of packets in the network can only increase. Eventually, assuming requests continue to be generated, the network's queueing capacity fills up, and it cannot accept any more packets. Consequently, the processors must start queueing request packets at their destinations because they cannot be serviced. Eventually this queue space fills up as well, and processors can no longer accept packets from the network. In the end, the network is waiting for the processors to accept packets and the processors are waiting for the network to accept packets. No progress is made, and the machine is said to *deadlock*.

The fundamental problem in the above scheme is that the network was not required to deliver any of the reply packets as long as there was a request packet to deliver instead. In effect, the network is acting as an adversary and forcing the receiver to process packets in an order different from the order it would like to process them. If the receiver could choose which packets it would like to process, then deadlock is easy to avoid.

1.2 Avoiding Deadlock

There have been several proposed solutions to the problem of deadlock. Unfortunately, none of these solutions adequately address the fundamental problem discussed above. I will outline two of these solutions.

- Make the network and/or processor queues infinite. This solves the deadlock problem by "brute force", making sure that there is always room to send one more message. This scheme is used in many commercial parallel machines. Unfortunately, this approach has two problems. The first is that one can't build an infinite queue, only a very large one. Designers must implement mechanisms to store large amounts of pending packets so that critical packets (ones not generating further messages) can get through. The second problem is that because the queues are not infinite, there is no guarantee of deadlock freedom, it is just very unlikely. These two problems show that this solution is not desirable because it is very memory intensive and not guaranteed to work.

- Build more than one network in the machine. This solves the problem by allowing the receiving processor to select which network it wants to service. For instance, if we put request packets in one network and reply packets in another, the receiver can always service the reply packets first. This solution works well but it requires a large hardware investment and is not expandable to the situation where more types of packets need to be sent.

1.3 Previous Work

A great deal of work has been done in making routing algorithms deadlock-free within the routing network itself. Some schemes use simple rules to constrain routing choices so that the routing is deadlock-free, for example dimension order packet routing on meshes. Modifications for wormhole routing have been suggested by Dally and Seitz using virtual channels [2]. More modern schemes modify the virtual channel idea to allow adaptive routing, as in Linder and Harden's scheme [7] that builds an acyclic virtual network out of virtual channels, Chien and Kim's planar routing scheme [1] that restricts adaptivity to bound queue requirements, and Su and Shin's $3P$ scheme [8] that is fully adaptive and uses small queues. All of these schemes, however, assume that destination processors are infinite sinks, i.e. each processor has an effectively infinite queue. Although these algorithms are very good for preventing deadlock *within* the routing network, they do not address the problem of deadlock *between* the routing network and the processors.

Thinking Machine's CM5 [6] has solved the deadlock problem for a limited case. In this machine there are two networks. One is called the request network and one is called the reply network. Request packets are sent on the request network and reply packets are sent on the reply network. Because replies generate no additional message traffic, they can be processed with no additional resource requirements and therefore the reply network always makes progress. Therefore, the reply network eventually empties and requests can continue to be processed, thereby emptying out the request network. This strategy enables the CM5 to avoid deadlock.

There are two problems with this scheme. First, dedicating each network to a specific type of traffic can have the effect of underutilizing the available resources. For instance, if the request packets are large and the reply packets are small, the request network is saturated but the reply network is underutilized. If the difference in sizes is large, the combined networks will only be 50 percent utilized. The second problem is that this scheme only allows a two-way protocol. Three-way or more complicated protocols can't be done this way because you need one network per packet type to insure deadlock freedom.

A related problem has been studied in relation to wide area networks. The problem in WANs is that the network wants to give a fair amount of bandwidth to each conversation passing over a link. Many algorithms for routing traffic fairly have been proposed for use in these wide area networks [3, 4]. These algorithms are able to support an arbitrary number of virtual networks (called conversations), but they require queues at each switch whose size is linear in the number of virtual networks. This bound is feasible for wide-area networks because conversations are typically persistent, point-to-point, circuit-switched paths, as opposed to transient, packet-switched broadcasts or many-1 routings, so buffer space is only needed on the switches that the conversation crosses. Furthermore, it is more feasible to implement large queues in wide area networks because the latency and compactness requirements for a switch are not as stringent.

1.4 Virtual Networks

The solution we propose for the deadlock problem is to allow the processors to act as if they have an infinite (or at least a very large) number of independent networks at their disposal. We will call this set of networks the *virtual* networks of the machine. Instead of actually building these networks in hardware, we will simulate all of them on a single physical network (see Figure 1-1). This idea is very similar to the idea of virtual memory where many virtual address spaces are simulated on a single physical address space.

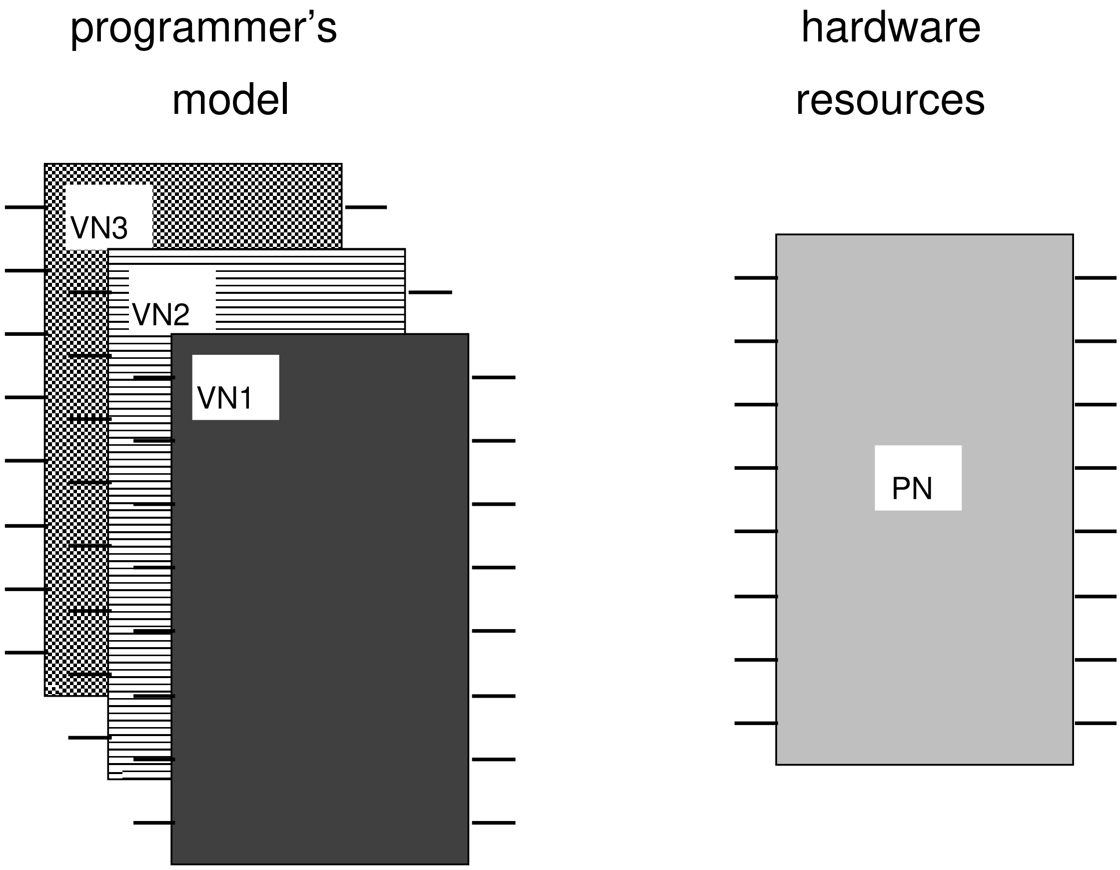

Figure 1-1: We will show how to simulate several virtual networks (VNs) on one physical network (PN)

The rest of my thesis describes how to implement this simulation and proves some properties about it. The algorithm we will describe is called the *virtual router*, since it routes the virtual networks over a physical network.

In order to facilitate the description of our new virtual router we need to decide on the topology of the physical network. In order to separate the issues of deadlock within the network and deadlock between the network and the processors, we will choose a deadlock-free network. In particular, we will use a network called the *butterfly* [5] (see figure 1-2). The butterfly network is popular because it has a large bandwidth (N packets per step) and low latency ($\log N$ steps). The results obtained in this paper naturally generalize to any levelled network of depth d. Also, additional virtual channels allow these results to be extended to non-levelled networks such as meshes and tori.

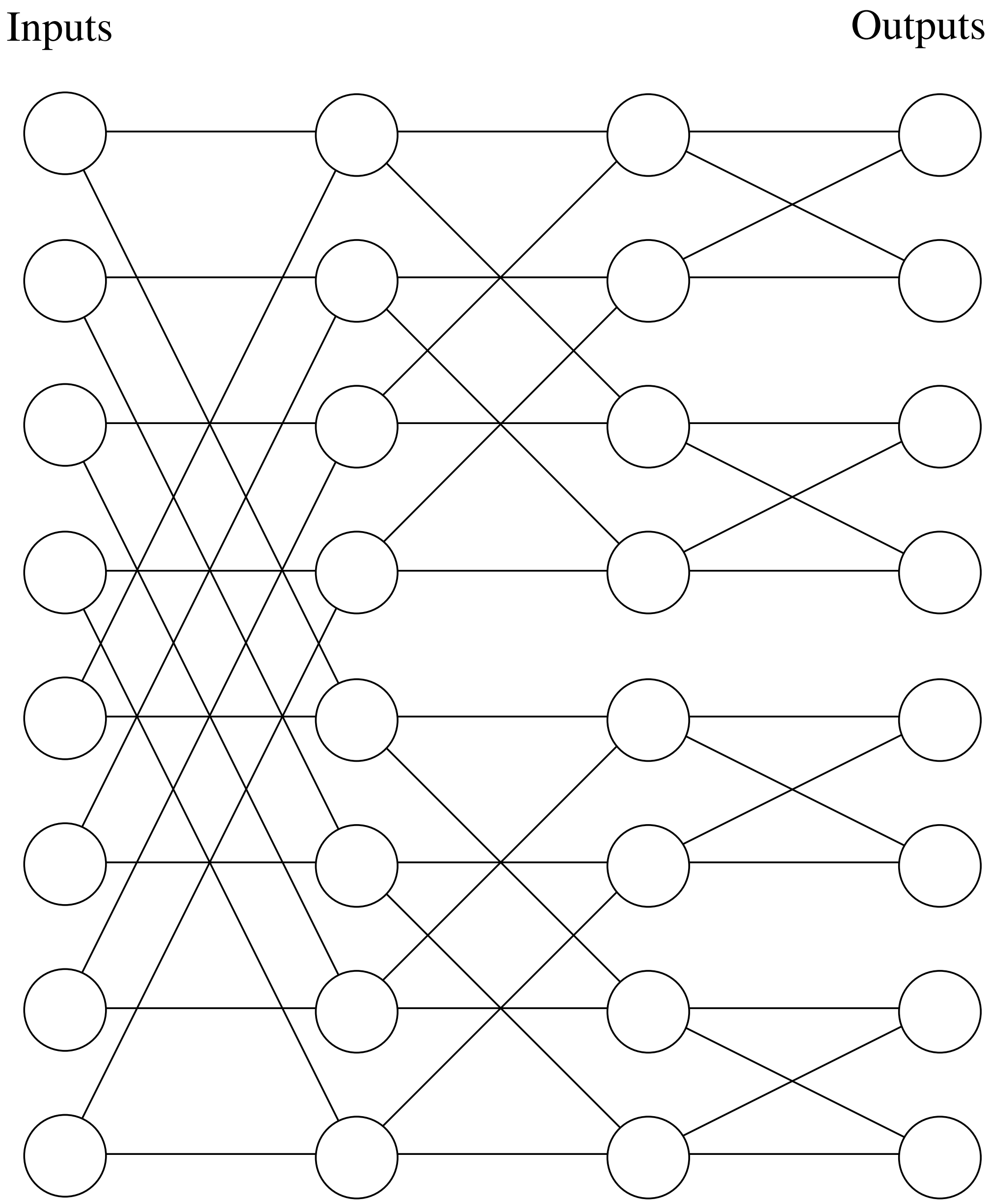

Figure 1-2: An $N = 8$ butterfly

In order to make the virtual router achieve its goal of avoiding deadlock in a machine, we must prove the following claim.

Claim 1 *(Progress) If a virtual network has at least one packet to deliver, and all* *of that virtual network's packet's recipients are able to accept those packets, then eventually the virtual router delivers one of those packets.*

Note that the progress of one virtual network can't depend on the status of any other virtual network. In other words, progress in one virtual network must be *independent* of progress in any other virtual network.

We will actually prove some stronger claims than this claim (see Claim 2), but Claim 1 is sufficient to allow a programmer to write trivially deadlock-free algorithms. All the programmer need do is to allocate one virtual network for each type of packet that he/she wants to send and make sure that there are no cyclical dependencies among these types of packets.

In addition to giving each virtual network a guarantee of progress, we would also like to guarantee that each virtual network gets a fair share of the total network bandwidth. Even though deadlock freedom is guaranteed by the Progress claim, a machine can slow down dramatically when the network is congested if one packet type is only given a small portion of the network bandwidth.

Claim 2 *(Fairness) If there are currently V active virtual networks, then the virtual router gives each network an $\Omega(1/V)$ fraction of the total network bandwidth.*

Finally, we would like to guarantee that our scheme does not impose too much of a performance penalty over using the raw network under normal network operation. This claim shows that the virtual router is practical and does not compromise on the performance of other deadlock avoidance schemes.

Claim 3 *(Performance) The virtual router routes a set of packets within a constant factor of the time it takes on the raw physical network.*

In order to show that a particular algorithm is acceptable, we must prove the above three claims for it. In the next chapter, we develop the algorithm for the virtual router, and in chapter 3 we prove the above three claims.

Chapter 2

Implementation Algorithms

This chapter first describes several implementation algorithms for virtual networks and analyzes each of them. Flaws found in early algorithms will lead to our final virtual router algorithm in section 2.3. We assume for simplicity that our physical network is an N-input, $\log N$-depth butterfly.

2.1 Simple Algorithms

We present here two simple algorithms for simulating V virtual networks on one physical network. These algorithms are based on time sharing the physical network at various time slice quanta.

1. Time sharing the whole network: This algorithm gives each virtual network a time slice of $\Theta(\log N)$ steps during which it has exclusive access to the network. Each virtual network is given its time slice in round-robin fashion.

   This algorithm works well when all virtual networks are heavily loaded. When this assumption is not satisfied, however, it has many drawbacks.

   - If some virtual network has no packets to send, the network is idle for $\Theta(\log N)$ steps. This problem could be solved by detecting an empty network, but this is costly because it involves a global accumulation.

  - If a virtual network has only a few packets to send, the network is mostly unused during its time slice. Even with an end-of-route detector most edges are unused, especially at the tail of the route when only a few packets remain to be delivered.
  - The latency of a packet transmission is large because if a packet just missed its scheduled time slice, it has to wait $O(V \log N)$ time steps before it gets another chance.

2. Time sharing the network cycle-by-cycle. Each network gets one cycle of routing every V cycles. This solves the problem of underutilized wires because we can always send another virtual network's packet on a wire if the currently active virtual network can't use that wire. It may turn out that when a virtual network gets its turn to route, however, all the queues into which it wants to put a packet are full, and it cannot make any progress. This phenomenon keeps us from satisfying the Progress claim, and therefore this algorithm is not acceptable.

2.2 The Virtual Channel Algorithm

We would like to design an algorithm that has the good properties of both of the algorithms 1 and 2 above. In particular, we want to have the guaranteed deadlock freedom of the first combined with the efficient utilization of the second. A solution to this problem is to use the second algorithm but reserve some queue space in each switch for each virtual network. For instance, we could reserve one queue entry for each virtual network and have some additional unreserved queue entries. Therefore, when a packet wants to move from switch to switch it can always go to its reserved queue spot. The only packets that can block its movement are other packets from its own virtual network, but this is acceptable because some packet from that virtual network will make progress if all output nodes have an empty queue for that virtual network. This is enough of a guarantee to satisfy the Progress claim. The extra unreserved queue entries can be used by any virtual network and are useful for maintaining full utilization of the wires when only a few virtual networks are active.

This algorithm is called the virtual channel algorithm because it resembles Dally and Seitz's virtual channel router in [2]. The only limitation of this algorithm is that the number of virtual networks is limited by the queue space at a switch. In order to support V virtual networks each switch must have a $\Omega(V)$ size queue. This scheme has much the same drawback as building a separate physical network for each virtual network because the number of virtual networks is fixed when the machine is built. In the next section we will develop an algorithm that requires only $O(1)$ queue space per switch to simulate an arbitrary number of virtual networks. This will allow a programmer to allocate virtual networks as his communication needs warrant without worrying about a hard limit on the number of networks that can be allocated.

2.3 The Virtual Router Algorithm

We now describe the algorithm for our virtual router. This algorithm must satisfy the three claims given in the introduction — Progress, Fairness, and Performance — and use constant queues to do so. In order to satisfy these requirements, our algorithm must have certain properties. The first is that we must allow for packets to be *dropped*. We require this property because if a queue is filled and a packet that wants to enter that queue which is from a different virtual network than all the queued packets, we must make room for the new packet to assure progress for that virtual network. Therefore, we must drop one of the packets currently queued at the node.

Second, we must use a Randade-like routing scheme [5]. Ranade's algorithm allows us to get good bounds on the routing time when constant-sized queues are used. This property will allow us to prove some statements about both the performance of individual virtual networks and the performance of the system as a whole.

The approach of our virtual router algorithm is to combine two networks into one. Our first network, called N_1, is a constant-queue, Ranade-routing network. This network routes packets using Ranade's algorithm and randomly selected keys. At any one time, most of the virtual networks will be routing in N_1.

Unfortunately, this physical network gives no performance guarantees to any virtual network. To remedy this, we add another network N_2 to our system that routes only one virtual network at a time. Like N_1, N_2 is also a constant-queue, Ranade-routing network. Each virtual network gets to use N_2 for a block of $\Theta(\log N)$ routing steps every $\Theta(V \log N)$ steps. The virtual network that is currently being routed on N_2 is called the *active* virtual network.

Both N_1 and N_2 have their own queues at each network switch and share wires. Priority for use of a wire is given to the network that has used that wire least recently. This rule guarantees that each network is allowed to use a wire on every other cycle. An exception to this rule is that ghost packets from both networks must be allowed to cross a wire every cycle. In any case, the worst traffic burden that a wire has to carry in any cycle is one real packet and one ghost packet.

Routing is performed in *rounds*. In each round, the following actions are performed.

1. An active virtual network is selected. See the next section for details.

2. Each packet is given a random key.

3. The active virtual network is routed on N_2 and all other virtual networks are routed on N_1. Routing proceeds for $\Theta(\log N)$ cycles.

4. Undelivered packets are dropped and acknowledgements, both positive and negative, are returned to the packet senders.

Note that step 4 of one round can be easily overlapped with step 1 of the next round, because they are transmitting packets in opposite directions.

2.4 Activating Virtual Networks

A fundamental problem in any virtual network scheme is that there may be many more virtual networks allocated than are actually in use at any one time. To highlight this distinction, let $\mathcal{V}$ be the set of allocated virtual networks, and let V be the set of virtual networks that actually have packets to route. In our scheme we would like to allow $\mathcal{V}$ to be large, say $|\mathcal{V}| = 2^{32}$, while V may contain only a few of the elements of $\mathcal{V}$. In the final analysis we want to guarantee a $\Omega(1/V)$ fraction of the physical network bandwidth to each virtual network.

When we cycle through the set of virtual networks we only want to activate those virtual networks that are in V. Giving a $\Theta(\log N)$ time slice to a virtual network in $\mathcal{V} - V$ would only waste N_2 for that time slice because that virtual network would have no packets in it. Unfortunately, there is no easy way for a particular processor to know what V is because some distant processor might have started routing some virtual network that it doesn't know about. Therefore, we need to have a distributed algorithm for deciding which virtual networks should be given a time slice of N_2.

Our algorithm to solve this problem routes *tokens* through the network to "claim" parts of the network for a particular virtual network. Each of these tokens is labeled by an integer equal to the number of the virtual network that it represents. Tokens are routed in circuit-switched fashion, but they also use the following routing rules:

1. Contention among tokens is resolved by sending the next token which is labeled in round-robin order after the currently active virtual network. i.e, if x is the currently active virtual network, then token $(x+1) \bmod |\mathcal{V}|$ has highest priority in the token round, $(x+2) \bmod |\mathcal{V}|$ has the next highest priority, and so on.

2. Whenever a token is sent from a switch, it is sent on both outputs.

3. Tokens are only given one routing wave every round.

Every time a new active network needs to be chosen, processors send tokens representing the next virtual network for which they have an outstanding packet. Processors with no outstanding packets send a dummy token that has lower priority than all other tokens. Tokens then route through the network according to the above rules. After $\log N$ steps, exactly one token exits the network at each output. Because we send tokens on both outputs at each switch, all tokens that exit the network at some round identify the *same* virtual network. This virtual network then becomes the active virtual network and gets to route exclusively on N_2 for $\Theta(\log N)$ cycles before another token round is performed.

2.5 Acknowledging Packets

In order for an input to know whether its packets need to be resent or not, it must receive an acknowledgment that its packets were either delivered correctly or dropped. If the longest packet lifetime is T, then there is a simple scheme to get all acknowledgments (both positive and negative) back in time $2T$. This scheme is to simply route the packet acknowledgements back through the network in the order opposite to which they traversed the network. In other words, a packet that traversed wire w at time t traverses that same wire at time $2T - t$ in the opposite direction. Therefore, the latest return time of any packet is at most $2T$. Furthermore, acknowledgments are often smaller than their corresponding packets, so we can run the acknowledgement routing at a higher speed (less time per routing cycle).

Chapter 3

Analysis of the Virtual Router

In this chapter we will prove that the virtual router scheme achieves the goals set out in section 1.4. These goals are to guarantee progress and fairness to each of the virtual networks, and to guarantee the overall performance of the system.

3.1 Progress Guarantee

The progress guarantee is fairly trivial to prove.

Theorem 1 *If a virtual network has at least one packet to deliver, and all of that virtual network's packet's recipients are able to accept those packets, then eventually the virtual router will deliver one of those packets.*

Proof: Suppose that at some time t a virtual network has some packets to deliver. Assume for contradiction that none of these packets ever get delivered. Because all of the virtual network's recipients can accept a packet, it must be the case that no packet ever makes it to an output. Since no packet ever makes it to an output, the virtual network cannot ever win a token round, because if it did then some packet would make it to an output $\log N$ time steps after the virtual network won the token round. But the virtual network must win a token round eventually because it always has an outstanding packet to send and its token must eventually become highest priority. Therefore, we have a contradiction and the theorem is proved. ■

3.2 Fairness Guarantee

The next theorem we would like to prove is that the virtual router is fair to each virtual network. In particular, we would like to prove that the virtual router guarantees that each virtual network gets an $\Omega(1/V)$ fraction of the network bandwidth.

In order to prove this statement, we must be precise about what we mean by V. Since this is a distributed algorithm, defining V as the set of virtual networks active at any one instant is of little use because no switch can be sure of what V is at any one instant. Therefore, we will define V as the set of virtual networks that are active somewhere within a window W of time that is sufficiently large.

Also, we must be precise about what we mean by "getting a $\Omega(1/V)$ fraction of the network bandwidth". The raw bandwidth of the butterfly is N packets per time step, so an $\Omega(1/V)$ fraction of that is $\Omega(N/V)$ packets per time step. Unfortunately, many routing problems can't achieve $\Theta(N)$ packets per time step on a dedicated network, so we certainly can't guarantee $\Omega(N/V)$ packets regardless of the routing pattern. Instead, we shall guarantee that each virtual network can route at least as many packets as it can route on a dedicated network in $\Theta(\log N)$ time in the virtual router in $\Theta(V \log N)$ time. Therefore, the bandwidth reduction from a dedicated physical network is $\Omega(1/V)$.

Theorem 2 *Let the time between rounds be $A \log N$ where A is a constant. Consider a time window W. Let S be the set of virtual networks that have outstanding packets at the beginning of W, and let V be the number of virtual networks that have outstanding packets during any part of W. For any W such that $|W| \geq (V+1)A \log N$, each virtual network in S gets at least as many of its packets delivered as it could deliver on a dedicated physical network in $(A/4) \log N$ time.*

Proof: We show that each virtual network in S either gets all of its packets through on network N_1, or it gets a time slice of $(A/2) \log N$ in W during which it has exclusive access to the network N_2.

Suppose that at the start of W, a virtual network v belonging to S wants to send a routing pattern R_v. Then one of the following three things may happen:

1. By the time virtual network v gets a chance to become active in W, all of the packets in R_v have been delivered.

2. Virtual network v still has packets to route when it gets a chance to become active in W. Then, it wins the token round and gets one round of routing on N_2. One round of routing corresponds to $(A/2)\log N$ cycles of routing because half of each round is spent routing forward. Furthermore, N_2 is guaranteed to get at least one of every two cycles on each wire, so it can simulate at least $(A/4)\log N$ rounds of routing R_v on a dedicated physical network.

3. Virtual network v never gets a chance to become active in W, but still has packets in R_v to deliver at the end of W. Fortunately, this situation cannot happen because a new token round is started every $A\log N$ rounds, and there are only V virtual networks competing for the use of N_2. Since priority is given in round-robin fashion, v must win a token round within $VA\log N$ time of the start of W. Therefore, if $|W| \geq (V+1)A\log N$, then v becomes active and completes its route before W ends.

Thus, in every window W of $(V+1)A\log N$ time, each virtual network in S can get at least as many packets delivered as it could deliver on a dedicated physical network in $(A/4)\log N$ time. ■

This theorem shows that each virtual network receives at least an $\Omega(1/V)$ fraction of the network bandwidth.

3.3 Performance Guarantee

Our last task is to show that the virtual router has a good performance guarantee. In other words, we would like to show that the virtual router does as well as any other routing algorithm on the overall routing problem.

Theorem 3 *The virtual router routes a set of packets within a constant factor of the optimal time, with high probability.*

Proof: Let R be the set of packets that the network has to route, and let C be the congestion of this route. We will prove that the total routing time is $O(C+\log N)$ with high probability. Clearly $C+\log N$ is a lower bound on the routing time, so this will prove our theorem.

We start by showing that the congestion is reduced by $\Omega(\log N)$ on every round of the routing. Let the keys for the packets be chosen randomly from the interval $[1, K]$, where 1 is high priority and K is low priority. Define a *small* packet as one whose key is smaller than $(aK\log N)/C$, where a is a constant to be determined. We expect that since the small packets have high priority, they are likely to get through the network, as the following lemma shows.

Lemma 1 *All small packets in a round get delivered to their destinations, with high probability.*

Proof: Let R_s be the subset of the routing problem R that contains all of the small packets. The congestion C_s of R_s can be bounded as follows. The probability that at least C_s small packets pass through a particular switch is bounded by

$$\begin{aligned}\binom{C}{C_s}\left(\frac{a\log N}{C}\right)^{C_s} &\leq \left(\frac{Ce}{C_s}\right)^{C_s}\left(\frac{a\log N}{C}\right)^{C_s}\\ &\leq \left(\frac{ae\log N}{C_s}\right)^{C_s}\end{aligned}$$

By choosing $C_s = 2ae\log N$ and $a \geq 1$, we know that the probability of at least $2ae\log N$ small packets passing through a switch is at most

$$\begin{aligned}\left(\frac{1}{2}\right)^{2ae\log N} &\leq N^{-2e}\\ &\leq N^{-5}\end{aligned}$$

Therefore, over all switches, the probability that C_s is greater than $2ae\log N$ is at most N^{-3}.

Thus, the congestion of the small packets is at most $2ae\log N$, with high probability. In particular, the congestion of the small packets in both N_1 and N_2 is at most $2ae \log N$. Using a delay sequence argument, we can prove that a congestion $O(\log N)$ route can be finished in $O(\log N)$ time with high probability using Ranade's algorithm [5]. Therefore, if we choose the length of the round to be $\Theta(\log N)$, the small packet routing completes by the end of the round, with high probability. ■

Next, we need to show that the congestion of the routing problem will go down by $\Omega(\log N)$ on every round. There are two cases:

- $C \leq a \log N$. Because the congestion is this small, all packets are small and therefore they all get routed in one round.

- $C > a \log N$. Consider a switch with a congestion of at least $C/2$. We show that at least $b \log N$ of these $C/2$ packets are small, where b is a constant to be determined. Let X be a random variable that denotes the number of small packets that go through a switch with a congestion of at least $C/2$. Then X is a binomial random variable with $p = (a \log N)/C$ being the probability of a packet being small, and $n \geq C/2$ being the number of packets. From probability theory we have that

  $$Pr\{Y \leq -a\} \leq e^{-a^2/2pn}$$

  for any zero mean binomial random variable Y with probability of success p and number of trials n. Therefore, we have

  $$\begin{aligned} Pr\{X \leq b \log N\} &= Pr\{X - pn \leq b \log N - pn\} \\ &\leq e^{-(pn - b \log N)^2/2pn} \\ &\leq e^{-(a/2-b)^2 \log N/a} \end{aligned}$$

  Choosing $b = a/4$, we have

  $$Pr\{X \leq b \log N\} \leq e^{-a \log N/16}$$

and choosing $a = 32$ we have

$$\begin{aligned} Pr\{X \leq b \log N\} &\leq N^{-2\log_2 e} \\ &\leq N^{-2.8} \end{aligned}$$

Therefore, the probability that some switch in the network with at least $C/2$ congestion does not deliver at least $8 \log N$ packets is less than $1/N$.

Therefore, the congestion after one round of routing is either 0 or it has gone down by at least $8 \log N$. As a result, we have shown that a route R with congestion C can lower its congestion by at least $8 \log N$ in $O(\log N)$ time, so all of R can be routed in $O(C + \log N)$ total time using the virtual router.

■

This theorem proves that the virtual router can deliver a routing problem within a constant factor of the optimal time.

Chapter 4

Conclusion

In this thesis, we constructed a routing algorithm called the virtual router. Its purpose is to simulate several virtual networks on one physical network, in much the same way as virtual memory simulates many virtual address spaces on one physical address space. It guarantees that each virtual network is able to make progress, that each virtual network receives a fair share of the network bandwidth, and that the overall routing scheme has good performance. The virtual router is useful for programmers of parallel machines who are worried about deadlock and are looking for a clean and easy way to reason about the deadlock-freedom of their algorithms. The virtual router is also useful for builders of parallel machines who are looking for alternatives to conventional methods of avoiding deadlock.

We proved that the virtual router will deliver packets in one virtual network independently of the progress of packets in any of the other virtual networks. This let us show that deadlock-freedom in a program is easy to achieve by simply choosing packet types so that there is no cyclical dependency among those types. We also proved that the virtual router guarantees an $\Omega(1/V)$ fraction of the network bandwidth to each virtual network, ensuring that near-deadlock situations cannot occur. Finally, we proved under modest assumptions that the overall performance of the virtual router was close to that of the original physical network.

Bibliography

[1] Andrew A. Chien and Jae H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. *The 19th Annual International Symposium on Computer Architecture*, 20(2):268–277, May 1992.

[2] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[3] Srinivasan Keshav. *Congestion Control in Computer Networks.* PhD thesis, University of California at Berkeley, September 1991. Available as Tech Report UCB/CSD 91-649.

[4] Srinivasan Keshav, Ashok K. Agrawala, and Samar Singh. Design and analysis of a flow control algorithm for a network of rate allocating servers. Technical Report CS-TR-2492, University of Maryland, July 1990.

[5] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes.* Morgan Kaufman, 1992.

[6] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffery V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The network architecture of the connection machine CM5. *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, June 1992.

[7] Daniel H. Linder and Jim C. Harden. Adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, C-40(1):2–12, January 1991.

[8] Chien-Chun Su and Kang G. Shin. Adaptive deadlock-free routing in multicomputers using only one extra virtual channel. In *Proc. of the 1993 International Conference on Parallel Processing*, volume 1, pages 227–231, August 1993.