Computer Architecture III Study Guide Interconnection Networks

Maximillien Courchesne-Mackie (mcour010@uottawa.ca)

I. Messages and Packets

The interconnection network's literal is called a message. A message can be of any size, thus, when sending it across a network, it must be broken into packets to avoid congesting the network. A packet consists of a payload (the contents of the message or a piece of a message), a header (which contains the routing information), an optional error code (which, if error detection is enforced at the network protocol layer, is used for error detection and correction) and a trailer (which contains other information for delivery such as the request type and response type).

A message can be cast into the network in a variety of ways. A *unicast* is a message which is intended for only one node in the network. A *multicast* is a message which must reach many nodes (but not all). Finally, a *broadcast* is a message which is received by each node in the network.

A. Network Latency

There are several factors influencing the latency of a network. These include:

- The **sender overhead** is the time it takes to prepare a packet for the network. This includes creating the packet envelope and moving the packet into the sender node's network buffer.
- The **time of flight** is the amount of time needed to send a single bit from the sender node to the receiving node.
- The **transmission time** is the amount of time needed for the rest of the bits of a packet to be sent across the network. This is largely a function of the link bandwidth and can be represented as $(N_p + N_e)/(fN_{ph})$ where $(N_p + N_e)$ is the total number of bits in the packet (including the envelope and the payload), N_{ph} is the phit size of the network and f is the clock frequency.
- The **routing time** is the time spent deciding the route a packet should take through the network. The logistics of this can vary depending on the switching strategy.
- The **switching time** is the time spent moving the packet between switches.
- The **receiver overhead** is the time it takes for the receiving node to remove the packet envelope and move the payload out of the input buffer.

The above factors, when added together, give us the end-to-end packet latency for a network.

B. Bandwidth

Bandwidth is a measure of how many bits a network (or switching element) can "move" in a fixed amount of time. Of course, in interconnected networks, we wish to increase bandwidth and decrease latency across the network. The bandwidth of a network can be calculated using the following formula:

$$\text{bandwidth} = \frac{\text{packet size}}{\max(\text{sender OH, receiver OH, trans. time})}$$

The *bisection bandwidth* of a network is a useful measure of a network's bandwidth. If the network is seen as a connected graph, the bisection bandwidth is the number of edges that must be "cut" in order to separate the network into two equal parts. In the case of a $2x2$ mesh network, we must cut through two edges to separate the network into two $1x2$ networks. That makes the bisection bandwidth $2b$. A higher bisection bandwidth is best.

A network may have high bandwidth, but could still suffer from *contention*. This is when a switching element receives several packets at once (from different nodes) and must decide which to service first. This process is known as *arbitration*. The flow control strategy is the algorithm which determines which packet gets to go first and the delay in choosing this called the *flow-control time*.

II. Switching Strategies

The switching strategy of a network is the way a route is established for a message and how a packet is transferred across switches through a network. In this section, we define a *phit* which is the amount of information which can be sent through the network in one network clock cycle.

A. Circuit Switching

In circuit switching, the route for a packet is first set up before transmission begins. Once the route is determined, the packet is pipelined through the network one phit at a time. While the route is active, the resources of the network are put on hold for the transfer to complete. This makes circuit switching not very efficient bandwidth-wise. The end-to-end latency for a circuit switching network is:

$$\text{latency} = \text{sender OH} + L(R+2) + N + \text{receiver OH}$$

Where L is the time of flight, R is the amount of network cycles needed to make a routing decision and N is the transmission time.

B. Packet Switching

Packet switching is comprised of two switching techniques: store-and-forward switching and cut-through switching. Unlike circuit switching, the route the packet takes in the network is not decided beforehand and is established as the packet proceeds from switch to switch.

1) Store and Forward: This technique consists of storing the whole packet in each switch before routing it to another switch. Each node along the route will accumulate the phits of the packet in a buffer and only forward the packet once it has received the whole thing. The packet spends N cycles in each switch and takes a total of LxN cycles to reach its destination (where L is the amount of switches along its path). End-to-end packet latency is the following:

$$\text{latency} = \text{sender OH} + N(L+1) + LR + \text{receiver OH}$$

Where R is the routing overhead.

2) Cut-Through: With cut-through switching, the packet is inspected by each switch and routed atomically. This means that the packet is pipelined through the network freely as long as there is no contention. If contention occurs, the transmission of the entire packet is blocked until arbitration finishes and the path is free once again. The latency using this strategy is as follows:

$$\text{latency} = \text{sender OH} + L(R+1) + N + \text{receiver OH}$$

There are two approaches which can be considered when contention occurs in a cut-through network. The first, *virtual cut-through switching*, dictates that each switch must have buffers able to hold the entirety of a packet. When contention arises, the packet continues to traverse the network and buffers at the busy switch, behaving more and more like a store and forward system. *Wormhole* switching, the switch only buffers a couple phits instead of the whole packet making the packet stored across many switches instead of just one.

III. Network Topologies

The layout (or topology) of a network can greatly affect its latency and scalability. In general, there is no "best" network topology, but certain topologies are best suited for different uses.

A. Indirect Networks

1) Bus: A bus is a common topology which grants exclusive access to one node at a time (arbitration). Most bus requests are followed by a response from the receiving node. Seeing as the bus is exclusive, it may go unused for a period of time if the response from the receiving node takes a long time. To make this process more efficient, pipelined or *split-transaction* buses allow other nodes to place requests on the bus while a previous request is being fulfilled.

Buses are not very scalable. As the amount of nodes increases, so does the length of the buses' wires, which leads to physical transmission latencies.

2) Crossbar Switch: A crossbar is a type of network which connects nodes through input and output ports. An NxN crossbar network can be seen as N vertical and horizontal buses interconnecting at N^2 points. The bandwidth of a crossbar network scales linearly with the amount of nodes added, however, seeing as it is essentially comprised of buses, the latency increases for the same reason as it does for buses.

3) Multistage Interconnection Networks (MINs): A *MIN* uses NxN crossbar switches to connect a set of nodes. The amount of switch "stages" is $log_2 n$ where n is the amount of nodes in the network. If the switch degree of each crossbar switch is k, then the amount of switches needed to make a working MIN is $Nlog_k N/k$. The number of switches in a MIN grows logarithmically at a rate of $O(Nlog_k N)$, which is good for scaling, but contention can be introduced which will increase latency.

4) Trees: A *k-ary tree* has N nodes and a depth of $log_k N$. Trees are simple designs, however, bottlenecks can occur at the root node of the tree. The bisection bandwidth of a tree network is always b (only one edge needs to be broken), which is a significant disadvantage. To mitigate this, it is popular to use *fat trees* which has links with higher bandwidths closer to the root of the tree.

5) Butterfly Networks: A butterfly network is essentially a MIN implementation of a tree network. The routes form a binary tree.

B. Direct Networks

1) Linear Arrays and Rings: The simplest implementation of a direct network is connecting each node to another forming an *array*. Consequentially, we can also connect the start and end of this array together to create a *ring*. These network topologies can send multiple messages at once, up to $N-1$, unlike a bus. However, a message travelling to a far away node must go through each node in the array until it reaches its destination. This leads to longer latencies. Connecting an array together to form a ring essentially cuts the network diameter in half and doubles the bisection bandwidth to $2b$.

2) Mesh and Tori Networks: To increase the bisection bandwidth of array networks, we arrange several of them together to create a mesh network. A mesh is a two dimensional array network with n rows and n columns and a node at each intersection. The bisection bandwidth is n and the network diameter is $2(n-1)$. If we want to improve the bandwidth, we can connect the end nodes of each row and column to the first node in that row or column, creating a torus network. This improves the network diameter and bisection bandwidth by a factor of 2.

3) Hypercube and k-ary n-cubes: An n-dimensional hypercube is a network topology with n dimensions and only two nodes per dimension. This creates a network with a very high bandwidth and low latency. It can connect up to $N = 2^n$ nodes with a network diameter of $n = log_2 N$ and a bisection bandwidth of $N/2$. A k-ary n-cube is built from k k-by-k tori networks stacked on top of each other and is very difficult to imagine in higher dimensions.

IV. Routing Techniques

A. Routing Algorithms

Different network topologies warrant different routing algorithms. In **MIN-type omega networks**, the routing address of a packet is the same as the destination node's address. In this scenario, the routing address is read starting with the least significant bit which determines if the packet will be routed through the upper port (bit $= 0$) or the lower port (bit $= 1$) for each stage.

In **butterfly networks**, the routing address is created by doing a bitwise XOR on the source and destination node.

Routing in **hypercubes**, called *e-cube routing*, consists of performing a bitwise XOR on the source and destination's three-dimensional address. For example, the routing address for a packet sent from node $(1, 0, 0)$ to node $(0, 1, 1)$ would be $(1, 0, 0)$ XOR $(0, 1, 1) = (1, 1, 1)$.

The above algorithms all suffer from possible *deadlock* scenarios where switches could find themselves in a locked routing loop making them unable to progress. **Dimension-order routing** is a form of restrictive routing which is deadlock-free. In the context of a mesh network, this algorithm would route packets in an xy or yx fashion, meaning they would travel in one direction before turning into the other. This removes any possibility for deadlocks.

B. Deadlock Avoidance and Adaptive Routing

A way to analyse if a routing algorithm is prone to deadlocks is to develop a channel-dependence graph. If it graphs shows that the routing algorithm has cycles, then it can be said that it is prone to deadlocks. Likewise, if there are no cycles in the graph (the graph is acyclic), the algorithm is deadlock-free.

While channels are physical entities, we can also conceive of a **virtual channel**. Virtual channels between two physical switches use different physical resources, allowing relaxed routing restrictions and also minimizing the chances of developing a deadlock. Take for example a mesh network with two "sets" of virtual channels. One with an xy routing rule and the other with a yx routing rule. Packets are initially routed on the xy set of channels, but can switch to the other virtual channels once during its transmission. As long as it never goes back to using the original virtual channels, deadlocks will be avoided and the latency of the network will be minimized.

No matter what deterministic routing algorithm is used in a network, packets can still fall prey to contention which greatly slows the network. A possible design response to this problem is to use *adaptive routing*. Adaptive routing will choose a different route for a packet if the network is suffering for contention, faults or hot spots.

V. Switch Architecture

The design of a switching element can greatly influence the latency of a network. The traditional switch design is comprised of an n-by-n crossbar with n input buffers and n output buffers.

However, this buffer-crossbar-buffer design suffers from *head-of-line* (HOL) blocking. This is when two buffered inputs both want access to the same output. The arbitration process will decide which packet gets to go first, but the blocked packet ends up blocking the entire input buffer (even if packets behind it aren't going to be routed through the busy output). A possible solution to this problem would be to have n buffers per input (one for each output). However, this solution doesn't scale well.