Fault Tolerant Networks

Lecture #11: Tuesday, 11 May 1999
Lecturer: William J. Dally
Scribe: Tom Fountain, TJ Giuli
Reviewer: Li-Shiuan Peh

Topics for this lecture

1. Definitions
2. Scenarios
3. 3-Step Approach
4. Link-level Recovery
5. End to End Recovery

1 Administrative

Checkpoint 1 was collected today and Professor Dally will send feedback on the project design to each group. Midterm solutions were distributed and graded midterms can be picked up from Shelly Russell in Gates 310. The mean on the midterm was 66 and the median was 67 and the distribution of scores is shown below in Figure 1.

2 Definitions

Fault tolerance is one of the more important topics in interconnection network and one that is often ignored or incorrectly defined. Understanding fault tolerant interconnect networks requires understanding two basic definitions:

Fault Tolerance - interconnect network continues to transparently operate in the face of a failure

Fault - something physical stops working (ex. open link caused by tripping over a cable, failure in memory table, etc.)

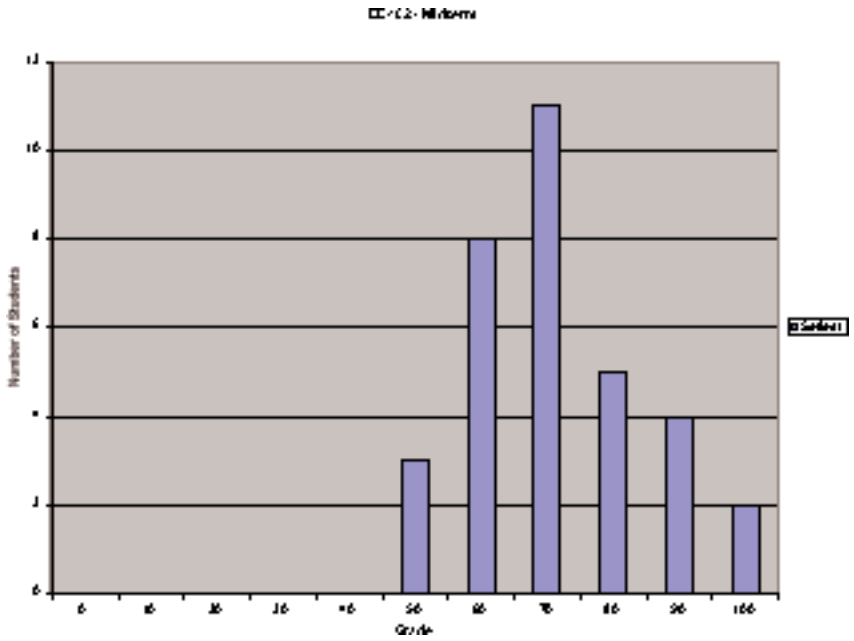

Figure 1: Grade Distribution

Faults can be caused by a large number of different events and usually take the form of shorted wires, open wires, or bit flips. Such faults rarely take out an entire node at a time, but often disrupt some portion of a node. The method for addressing the fault can take many forms, and depending on the recovery technique may take the node down or leave it up and simply drop a packet.

The first step in reasoning about faults is to create a fault model. A fault model provides an abstract classification scheme encompassing a number of actual faults that might occur. For instance, maliciously removing a cable, tripping over a cable, and poor soldering on a link could all be considered part of a Stuck-At Model. In a Stuck-At Model, we reason about the network assuming that the links are stuck low at 0 or high at 1. This abstract classification no longer considers the root cause of the network fault, but instead allows us to reason about the effects of the fault.

Another classification we might consider is a transient model. In a transient model, there is some probability that a bit is either right or wrong at any given time. The Bit Error Rate (BER) is used as the measure of that probability. An example of a transient model is a bit flip in a router's memory table.

2.1 Measurements

In addition to being able to discuss the causes of faults and fault models used to reason about them, it is also important to be able to evaluate how well a particular network performs. Some of these metrics include reliability, availability, mean time between failure,

mean time to failure, and mean time to repair.

Reliability is a measure of how well the network operates as a function of time. We define $R(t)$, as shown in Figure 2 , to be the probability at time t that the system is working given that it was originally working at $t = 0$ without repair.

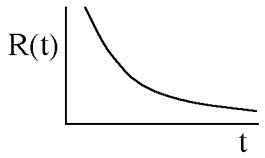

Figure 2: Reliability

Since most networks can be repaired, a more important consideration in the construction of a network is availability. The availability of the network is a measure of the uptime of the network and is formally defined as the probability that the network is working at any given time.

Two more important metrics in evaluating networks are Mean Time Between Failure (MTBF) and Mean Time To Failure (MTTF). The Mean Time To Repair (MTTR) is the average amount of time before the network has been repaired and is operational again. A diagram illustrating these two concepts is shown below in Figure 3.

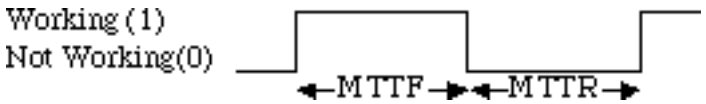

Figure 3: Working State

Using Figure 2, we can calculate availability as the probability that the network is working at any given time, given by the equation:

$$A = \frac{MTTF}{MTTF + MTTR} \approx 1 - \frac{MTTR}{MTTF}$$

This means that a highly available network can be constructed by either building it such that it has a long mean time to failure or short repair time. Systems with high availability automatically reconfigure themselves without any user intervention.

3 Fault Scenarios

To help understand the different faults we may have on the network, we consider five different scenarios:

Using Figure 4:

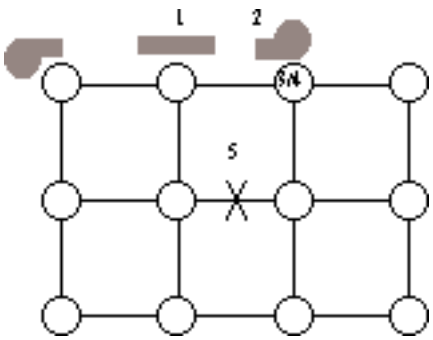

Figure 4: Fault Scenarios

1. Bit Flip in Data - A single bit in the data portion of the packet is flipped, such as by a transient failure on a channel.

2. Bit Flip in Header - A single bit in the header is changed, causing a change in the VC, destination, packet type, or one of the other header fields. This type of fault is considered a control state error and is in general referred to as a "hard" error since it is particularly difficult to detect and recover from these problems. A great many routers in practice fail miserably when this occurs.

3. Bit Flip in State - While sitting in memory, the global state, output port, pointer, credit, etc. changes.

4. Bit Flip in Flit Buffer - Bit flips while sitting in the flit buffer. This fault scenario is equivalent to #1 where a data bit is flipped while in route.

5. Lost Channel - A channel goes down with data possibly on the link.

A sixth scenario not addressed at this time is the hard failure of a node. This type of failure can be considered to be equivalent to #5 where all channels connecting a given node are lost at the same time and it can no longer be reached by the rest of the network.

4 3-Step Approach

There are three steps involved in handling a fault. The three steps are:

1. Detect the fault and diagnose where the problem is and what type of problem exists.

2. Contain the fault to limit the scope of possible problems a single fault may cause. It is not uncommon that a single fault can have dire consequences and spread to surrounding nodes, often resulting in a complete crash of the network.

3. Recover and correct/fix the fault, or somehow reconfigure such that the network can continue to operate.

Many different approaches to these three steps can be applied to the scenarios outlined.

4.1 Scenario #1 - Bit Flip in Data

In this scenario, a single bit in the payload of a packet is flipped. A bit error could be caused by a number of different sources such as a stuck bit or nearby electro-magnetic noise.

The first step is to detect the fault. The most common solution is to use parity checking. The parity check should be used on a flit-by-flit basis. Checking parity on a packet basis would make containment extremely difficult and likely mean that corrupted data would be sent on at each stage until it reached the destination which could then check the parity of the entire message. Using per-flit parity checking, the receiving node verifies the parity at arrival of the flit.

A common approach to containing the error is to reboot the system. While this method causes significant loss of data, it does ensure that the incorrect data does not cause other errors.

If an error is detected, a better approach is to recover using link-level retry, sometimes referred to as flit-level retry. In this scheme, the receiving node sends a NACK and the VC number to the previous node to request a retransmission. The previous node does not delete its copy of the flit until it knows that the next node successfully received the data (the Reliable Router goes on to protect against node failures by ensuring that there are always two good copies in the network so that the previous node's predecessor maintains its copy until the current node acknowledges successful receipt). An illustration of this process is shown in Figure 5.

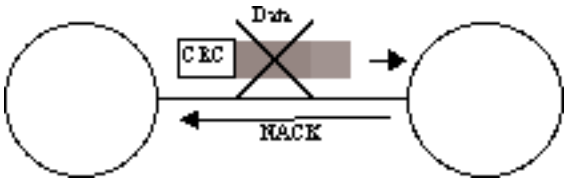

Figure 5: Per-flit Parity and NACKs

Unlike the other scenarios, the bit flips in this scenario does not cause a problem for the router, but instead causes a problem only for the application using the router. Therefore, a second approach to the bit flip in data is to simply let the end-to-end application detect and correct the problem. This approach, however, seems unsatisfactory in a number of ways. First, it will require retransmission of the entire packet through the entire network,

thereby incurring a significantly larger retransmit time. More importantly, the end-to-end approach does not solve the other faults identified by the five scenarios. Designing a reliable flit delivery system addresses the other potential faults.

In their well known paper relating to operating systems, Schroeder and Saltzer argue that nothing less than end-to-end recovery will suffice (ex. file transfer over a network). Therefore, they argue that all other recovery mechanisms are redundant. While this may be true, additional recovery techniques are often useful for improving performance.

The Bit Error Rate (BER) in the T3E network was 10^{-30}. Using link-level retry, the BER can be remarkably high. Consider the case where the BER is 10^{-15}. If the BER is 10^{-15} in a network with 1,000 nodes and 16 wires/channel, then $10^{-15} BER * 10^9 Hz * 1,000 nodes * 16 wires * 6 links/node$ is 1 failure every 9.6 seconds. Link-level retry makes this acceptable.

A third approach is Forward Error Correction (FEC) which would use $log_2(n)$ correction bits instead of just the parity detection bits. While this method allows the node to correct for a single bit error, it means that additional bandwidth is wasted. If bit errors are expected to be infrequent, it will be better to simply ask for a retransmit.

4.2 Scenario #2 - Bit Flip in Header

In this scenario a bit is flipped in the header and causes a corruption of control state, making it difficult to recover. End-to-End recovery mechanisms are simply unrealistic in this model since it is possible that data will never arrive at the destination, or any end-point at all.

A bit flip in the header could change many different fields. For instance, consider the case where a network has only two VCs and the bit flip occurs in the VC field of the tail flit. The second node has a full VC1 buffer with two flits A,B. Therefore the first node has 2 credits for VC0 and 0 credits for VC1. A flit is sent on VC0 but is changed in flight to VC1. The second node looks at its pointer and places the packet over the top of A, thereby destroying it (an innocent bystander). The packet sending through VC0 is now in an incomplete state and the resource will not be released. End-to-End recovery will therefore not work. Figure 6 illustrates this example.

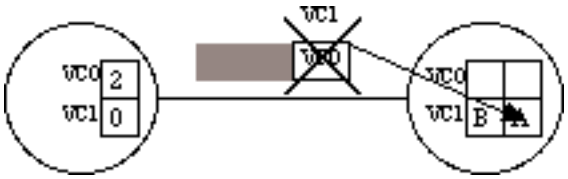

Figure 6: Corrupt Control State

In this example, it is necessary to contain the failure since VC1 will send it on to the next router with a premature tail flit. Once virtual channel states are inconsistent,

downstream routers are often affected. If the flit in this example were a head flit rather than a tail flit, downstream routers would allocate new resources.

The first approach to solve this problem is to use a crude Error Correction Code (ECC) scheme and send multiple copies of key pieces of information. An example of this is the Reliable Router's flit format shown in Table 1 where the flits contain duplicate copies of certain critical fields. These fields are also split across different flits in case an entire flit is lost and on different wires in case of a wire failure.

	Bit Field					
	22	21	20:18	17	16	15:0
Subframe 0	pc	USR0	VCI	BP1	BP0	Data[15:0]
Subframe 1	Copied Kind		Copied VCI	BP3	BP2	Data[31:16]
Subframe 2	U/D	USR1	Kind	BP5	BP4	Data[47:32]
Subframe 3	Freed			BP7	BP6	Data[63:48]

Table 1: Reliable Router Flit Format

The parity and duplicate information allows detection of a bad flit. One containment strategy is to throw away the flit. Even if the flit is discarded, it is still necessary to properly credit the upstream router. It is also worthwhile to periodically send an absolute credit to ensure a consistent state. This approach is often known as a "healing protocol" since interaction corrects inconsistencies in state. Assigning a direction to returning to a consistent state is important. The design philosophy used by Dennis Allison is that "shit flows down hill" since a fault can corrupt data as it flows toward the destination. Rather than forcing a reconfiguration state uphill, only downhill nodes are affected and can simply destroy packets until they are trustworthy again. After several iterations, the problem has been contained and everything has been healed.

One problem with dropping flits to heal the network occurs when one of the dropped flits is a tail flit. Without a tail, the virtual channel is never deallocated. A timeout should be set to signal the release of a virtual channel that has been held for too long. This is difficult, however, because state must be modified on multiple nodes and therefore coordination between nodes is required.

Another issue caused by header corruption is trying to ensure that the network remains deadlock free. If the destination of a packet is flipped, it is possible that the packet could violate an imposed ordering, thereby causing deadlock. Thankfully, the probability of such an event is realistically extremely low.

As a general design philosophy, implementing containment at the link-level provides the best performance and is recommended so long as the solution is not too costly. Similarly, the earlier a fault can be contained, the simpler the correction.

4.3 Scenario #3 - Bit Flip in State

Bit errors in the status of a channel could change state from Routing to Active. One solution to this problem is to use multiple bits to encode state such that at least a double-bit error would be necessary, which is an extremely unlikely event. Handling non-control state errors also requires modifying the state machine so that flits can be retransmitted. Figure 7 shows the state machine used in that network.

Figure 7: Reliable Router State Machine

4.4 Scenario #5 - Lost Link

A lost link can occur under a number of different conditions, such as when a wire is bad or two wires are shorted together. A link could be marked as "suspect" after a single transient failure and removed from service if a second failure occurs before sufficient time passes to return it to its normal status.

"Heartbeat" packets ping-pong between two adjacent nodes to constantly test the link when the channel is idle. The constant testing allows a failed link to be detected before any data is lost. Since in practice links are idle more often than in use, the result is that most failures do not occur when live data is travelling across the wire.

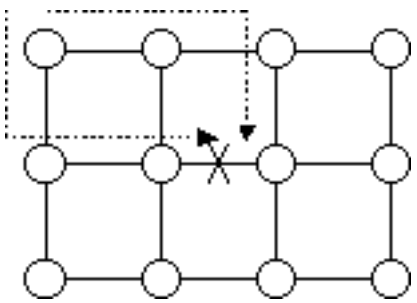

Figure 8: Bad Link

Recovery from link failure requires design consideration of both topology and routing strategy. Simply having multiple paths is not sufficient and the topology must have multiple edge-disjoint paths from every possible source to every possible destination, where edge-disjoint paths are paths that do not share any links. A k-ary n-fly network, for instance, is not able to recover from a link failure since there is only one path from the source to the destination. In contrast, however, k-ary n-cubes do provide path diversity. An 16-ary 1-cube is simply a ring and flits can travel the opposite direction around the ring to reach their destination. Figure 8 is a mesh network that provides at least two edge-disjoint paths from any source to any other destination.

The routing scheme must also be able to handle link failures. Random selection between multiple paths at the routing level will successfully deliver the packets if there is an end-to-end retry mechanism already in place. If node-table routing were used instead of random-choice, one possible way to handle bad links would be to mark paths containing the bad links in the route table as unusable. A third possible routing algorithm that can handle link failures is non-minimal routing, since it increases the number of edge-disjoint paths. And finally, a fourth method is to use backtracking.

If source-routing were used, downstream nodes can "deflect" packets headed for bad links. Deflection is when a source-routed packet passes through a node that knows a link on the downstream route is bad and so it replaces the packet's route with a route that it believes is good.

A good topology and routing algorithm may be able to recover from a link fault, but the network may no longer be deadlock free. Using Duato's algorithm, if the escape channel for a particular node is over a bad link, a deadlocked network could result. To address this problem, the Reliable Router added an additional set of virtual channels that bridged over adjacent links.

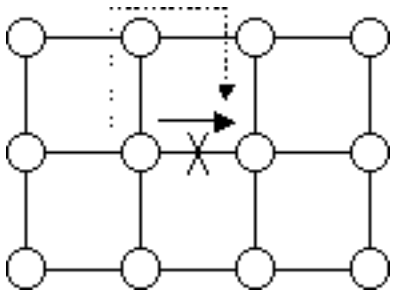

Figure 9: Rerouting to Avoid Deadlock

Adding this third set of virtual channels that bridge adjacent links guarantees that the network will continue to be deadlock free. To maintain the ordering of the escape channel, once a flit is routed onto a bridge, it must continue all the way over the bridge until it reaches the adjacent node, as shown in Figure 9.

What happens to the packet spread across the network when a link between two parts of the packet goes down? The Reliable Router uses a "unique token" scheme to

ensure that no data is lost from in-flight packets. Each tail flit in a packet has a token that uniquely identifies the packet. When a link goes down, the packet is effectively cut into two separate packets, each with its own head and tail. The last downstream flit to be successfully transmitted over the now-bad link becomes the new tail and is given a duplicate token from the original tail. The flit preceding the new tail becomes the new head. When the destination receives two packets with duplicate tail tokens, it knows that it must merge the two packets back together. A diagram illustrating the tokens progress through the network is shown in Figure 10. The advantage of this algorithm is that extra processing only takes place in the case of duplicate tokens - the common case where each packet arrives whole is not slowed down.

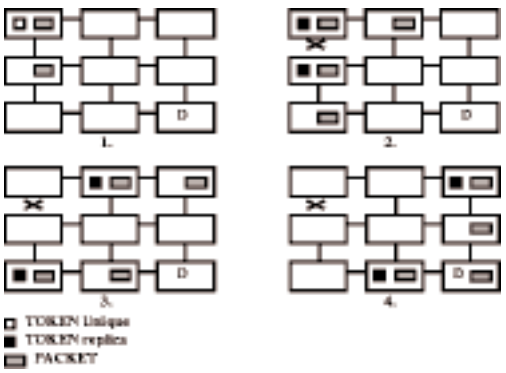

Figure 10: Unique Token Protocol