Chapter 6: A proactive scheme for fault tolerance

6.1. Introduction

This chapter presents some possible schemes to provide fault tolerance on top of an underlying realtime service. The approach used is to make reservations on multiple paths, using the available capacity and connectivity of the network, to provide fault-tolerant realtime connections, which can offer a variety of grades of service in the presence of network faults. Forward Error Correction (FEC) techniques are used in conjunction with the multi-path reservation when completely transparent fault-tolerance is required. This approach is useful in a network where the transmission capacity is relatively abundant, and for applications where tolerance to failures and non-stop operation is essential. This approach would not make sense in a bandwidth-poor environment such as the current Internet, where depriving other users of bandwidth to increase the tolerance of one's connection would be regarded as anti-social behavior. However, with the gigabit per second networks envisioned in the future, bandwidth requirements may become less of an issue, and the improvement in the service provided may allow some applications to use a shared network, which would otherwise require a private network.

The ability to put many different classes of applications, including those that require some degree of fault tolerance, on the same network opens up the possibility of reducing the cost of the service by exploiting economies of scale. The target applications for this service include remote surgery, space missions, industrial control, and so on, where fault-tolerant realtime audio-visual communication is required. Some interactive multimedia applications may also benefit from fault tolerance, if the associated costs, in terms of increased resource requirement, are low enough.

The underlying realtime service allows for the reservation of buffers on a per-connection basis to eliminate loss due to buffer overflow. That still leaves two sources of packet loss in the network: transmission errors and network failures, which are not covered by the guarantees provided by the Tenet scheme. The primary objective of the mechanisms presented in this chapter are to handle network failures. However, tolerance to errors is provided are a side-benefit of using FEC to control loss due to faults.

An error can be looked at as a transient fault in the network. A long term fault, on the other hand, is one where the link does not transmit information for a long period of time, relative to packet transmission times. There is a continuum of error states in between these two extremes, and, for practical purposes, one may define a threshold of error rates, as described in the previous chapter, beyond which the link is removed from service. For simplicity, we will consider only the two extremes, and assume that either packets are lost with a constant (low) independent probability on all links, or that a link fails and does not transmit any information at all. We will discuss both the error-tolerant and fault-tolerant properties of the schemes.

The schemes for fault tolerance presented here are based on the idea of dispersity routing (44), described in Chapter 1. However, application to realtime networks requires a reconsideration of the capacity and performance requirements. Furthermore, the ability to continue to provide *realtime* service in the event of a fault must be built into the schemes, and validated through simulation. This chapter takes the idea of dispersity routing and modifies it in the context of realtime networks, to form several schemes which provide different levels of service, at different costs to the network. Some of the schemes provide completely transparent tolerance to restricted network fault scenarios (e.g., single or double faults) at the cost of increased bandwidth requirement. This is done by using multiple paths and FEC to introduce sufficient redundancy so that some of the paths can fail without disrupting the flow of data. The principle of realtime channels are used to provide guaranteed performance service on each of the paths, so that the overall service is also guaranteed.

Some other schemes offer gracefully degrading service as the number of faults in the network increase. They use the multiple paths through the network to provide a service whose capacity degrades gracefully, and use appropriate coding techniques to ensure that any of the surviving streams can be used independent of the others. Still other schemes provide a service, which is disrupted in the event of a failure, but the recovery is guaranteed and the recovery time is short. This is made possible by preserving the backup resources, but allowing non-realtime data to use this spare capacity until it is needed to recover from a fault.

A framework to classify the different schemes is developed in Section 6.3, which also describes the service offered by each scheme, and the expected impact on network capacity based on bandwidth requirements. In Section 6.4, we present the design of our experiments. Section 6.5 presents the results of the simulations, which validate the service offered, and provide a more accurate picture of the network capacity issues. We then discuss some practical issues in Section 6.6, such as the changes to the network interface required to support fault-tolerant realtime connections, and the interaction with the recovery schemes described in the previous part of the thesis. Section 6.7 concludes by summarizing the service provided and the network costs of the schemes, and comparing them to another existing approach to fault tolerance (74). We start with Section 6.2, which provides some background information to be used in the course of the chapter.

6.2. Background

This section summarizes some background information on compression algorithms and error-correcting codes. While the fault tolerance schemes to be developed in this chapter are applicable to any realtime data, video traffic requires some special consideration. The realtime paradigm is well suited to audio-visual information, because of the regular time-based nature of the information. Audio, having a lower bandwidth requirement, can be handled much more easily by the fault tolerance mechanisms. Video is harder, firstly, because the larger bandwidth forces us to seek solutions that use resources more efficiently, and secondly, because, due to the large bandwidth requirements and the inherent redundancy of video, it is usually transmitted compressed, and some of the compression techniques introduce dependencies in the video stream. This increases the fault and loss sensitivity of video . We devote Section 6.2.1 to a discussion of the compression techniques, with special consideration of how partial information may be decoded. We provide a brief description of intra-frame, inter-frame, block update and hierarchical compression ideas.

In the sub-section on error correction, we provide a description of diversity encoding and Priority Encoding Transmission (PET). These techniques can be used to increase the redundancy of a stream and to split the stream into multiple streams to enhance the error and fault tolerance properties. Readers familiar with the above can safely skip this section, since we will not go into extensive detail.

6.2.1. Compression techniques

Intra-frame compression

Intra-frame compression techniques take advantage of the spatial redundancy inherent in the digital representation of most pictures. The information required to describe one pixel is usually very similar to that required to describe the next one. The intra-frame compression technique proposed by the Joint Pictures Experts Group (JPEG) depends on dividing the screen into blocks of 8x8 pixels, subjecting each block to a Discrete Cosine Transform (DCT), quantization, and Huffman encoding. This process can typically compress the representation of the picture by a factor of 10:1; the level of compression can be traded off for picture quality, leading to a large range of possible compression ratios.

Pure intra-frame compression does not take advantage of the temporal redundancy inherent in video sequences, especially at the higher frame rates. However, each frame can be decoded independently of all others. Since each lost packet only affects one frame of the video sequence, compression codes like JPEG are relatively tolerant of packet loss. When a packet is lost, an unknown number of bytes of the current frame has been lost, and the decoder state is not synchronized with the bit stream contained in the next packet. Thus, most decoders discard the bits until the next marker, where the state can be synchronized and decoding restarted. Such markers are always present at the start of a new frame, but may occur more frequently, such as at the start of every new packet. However, many current JPEG compression/decompression hardware devices only arrange for resynchronization at the start of each frame. Since resynchronization can always be achieved by the start of the next frame boundary, no more than one frame is ever affected by a single packet loss.

Inter-frame compression

Exploiting the temporal redundancy of video sequences permits an additional improvement in compression ratio. Since the picture does not change much from one frame to the next, a frame can be expressed as the change from the last frame using a much smaller number of bits than required to independently code the entire frame. This idea is used in the coding standards proposed by the Motion Pictures Experts Group (MPEG), where frames can be of three types. I frames are coded using pure intra-frame compression, similar to JPEG compression. P frames are coded on a block basis; each block is coded either in predictive mode or intra-frame mode depending on the accuracy of the prediction based on the motion estimation. The intra-frame mode is the same as for the I frame blocks. In predictive mode, a block is represented as a set of motion vectors, interpolated relative to the closest preceding I frame, and prediction error information. I and P frames are also known as "anchor frames". B frames are interpolated based on the closest preceding and following anchor frames. The compression or decompression of B frames requires waiting for frames later in the sequence, leading to increased delays, buffer requirements, as well as compression complexity. In addition, the video service becomes even more susceptible to errors since there are more dependencies. Thus, interpolative coding (and, hence, use of B frames) may not be suitable for realtime applications (49).

Inter-frame compressed video suffers from dependencies between frames. Thus, the loss of an I frame affects the entire group of pictures (GOP) until the next I frame. Thus, if possible, the network should try harder not to lose an I frame than a P (or B) frame. Some ATM networks will provide differential cell dropping, so that, in case of buffer overflow, cells marked as lower priority will be dropped in preference to higher priority cells. However, the losses due to transmission errors and faults cannot be prioritized directly. In the next section, we will discuss a differential FEC scheme, which allows different levels of redundancy to be imparted to different portions of a video stream, to achieve a similar effect.

Asynchronous video coding

The basic idea behind asynchronous video schemes is to treat video as a series of updates to the screen, so that information can be displayed when the packet containing it arrives, independently of other packets in the same stream. For example, block update compression treats the screen as a matrix of blocks, and transmits a block only if it has changed (by more than a threshold) since the last time it was transmitted, thereby taking advantage of some of the temporal locality in video. Blocks may be compressed, using techniques such as DCT, to further reduce the bandwidth required. This technique is especially suited to video without much action, panning or scene changes, such as usually found in video conferencing, lectures, and so on. Each block is transmitted with the position information necessary to render it. If a packet is composed of randomly selected blocks, the loss of a packet has an effect distributed over the entire picture. This scheme is fairly tolerant of losses, especially for the classes of video applications just mentioned. The typical image is a human in front of a static background; the blocks being updated correspond to the human as she moves around. Thus, a block which is updated is likely to be updated again soon, as the human will move around in the same portion of the image. Even if an update is lost, it will be updated again soon afterwards.

The above technique has been used in a video conferencing tool called `nv`, with a fair degree of success. The compression can be performed in software in realtime, leading to applicability to a large class of hardware platforms. The major complaint with the technique is that people in the image tend to leave body parts scattered around the screen, when packets are lost. To partially compensate for packet loss, a background refresh process sends the entire picture, block by block, at a rate chosen depending on the overall bandwidth constraints. The technique can scale to the available bandwidth by changing the block size, the update rate, the threshold for transmitting a block, and the background refresh rate. The picture quality, as well as loss tolerance, increases with increasing the background refresh rate, increasing the update rate, decreasing the threshold for transmission, and decreasing the blocks size; this improvement comes at the cost of the increased bandwidth needed by the video stream.

Similar ideas are being investigated by at least two groups of researchers (45,48). An additional twist to the basic idea is to generate multiple sub-streams by separating the regions of fast motion and slow motion, and sending more detail for the slow motion regions. The sub-stream for the fast motion regions contains more frequent updates, but less detail. This sub-stream is more important to the perceptual quality of the video. A third sub-stream may contain the refresh information. The idea of multiple sub-streams for compressed video is applicable to non-asynchronous video coding schemes as well, and is generally referred to as hierarchical compression.

Hierarchical Compression

Hierarchical compression consists of coding video sequences into a number of compressed sub-streams with the property that an image sequence can be reconstructed from many subsets of the set of sub-streams. If all the sub-streams are used, the highest-quality video sequence is obtained. However, if a lower bandwidth stream is required, some of the sub-streams may be dropped, and the decoding algorithm can still recover a lower-quality video, where the quality of the video is somehow related to the cumulative bit-rate of the received subset.

One possible scheme for hierarchical compression uses a stream of highly compressed video as the base low-quality signal. The other streams provide different levels of enhancement information, so that in combination with the base stream, various levels of higher quality video may be obtained. Such video compression schemes are also known as scalable or multi-resolution compression. They are useful for multicast, since, if the route to a particular receiver does not have enough resources to support the highest resolution, some fraction of the scalable stream may be dropped in the network, to allow communication at some reduced quality. They also allow variable quality video to be retrieved from a video server, depending on the bandwidth that can be supported by the server and the network at the time. In the context of ATM networks, the cells of the higher layers of a scalable video stream may be marked to be dropped first in the event of congestion. When congestion occurs, the video would degrade to the lower quality, since only the cells of the higher layers would be dropped.

MPEG-2 contains scalable modes which can offer such multi-resolution compression to a limited extent. Temporal scalability can be provided somewhat artificially by treating the I, P and B frames as separate streams of hierarchically encoded video. Another possible approach is to include more frames in the enhancement layer. Spatial scalability can be provided by improving the high resolution image by combining motion prediction, the base layer image, and prediction error information. SNR scalability is supported by using coarse quantization of the DCT coefficients for the lower layer, and sending additional residual information at the higher layer. The number of levels that can be provided, as well as the compression efficiency for the higher layers, is limited compared to true scalable compression techniques such as 3-D sub-band coding, where sub-band transforms are used in the temporal domain as well as in the spatial domain. Unfortunately, the delays incurred in the compression and decompression of true 3-D sub-band coded video are currently too expensive for interactive applications. Research into reducing these problems is in progress (62).

6.2.2. Error correcting codes

This section describes some forward error correction coding techniques. Diversity coding is used to recover from the loss of a known set of code words from a larger set of transmitted words. The Priority Encoding Transmission (PET) extends these coding ideas to multiple levels of redundancy for different layers of a hierarchical stream.

Diversity coding

Given a message, consisting of K equal sub-messages of equal length, can we create L additional code sub-messages, of the same length, such that the original message can be reconstructed from some fraction of the $K+L$ sub-messages? Forward error correction codes to solve this problem exist, and are easier to implement than standard error-correcting codes, because the decoding algorithm knows which contiguous portions of the encoded message have been lost. A simple example of a code which can recover the original message from any K sub-messages from a set of $K+1$ sub-messages is the parity code. The original message is broken into K equal parts, and the $K+1^{st}$ sub-message (of the same length as each of the K parts) is constructed as the bitwise parity of the K parts. If any one sub-message is lost, it can be reconstructed as the bitwise parity of the ones that were received. Codes which can recover the original message from any K of the received sub-messages are known as maximum distance separable codes. A large class of non-binary codes, the Reed-Solomon codes, achieve this limit (42). No code can recover from loss of more than L messages.

The parity code is simple enough to be implemented in software at realtime speeds. The non-binary codes can be implemented in inexpensive hardware at high speeds (5). They may also be implemented in software, and run at close to realtime speeds on current high end workstations. Experiments performed in the context of the PET project (described in the next section) at the International Computer Science Institute at

Berkeley show that a software implementation of the Reed-Solomon codes can run at rates of up to 1.2 Mbps on a SPARC 10 workstation with a 50 MHz processor (39). Recent work showed the feasibility of probabilistic decoding, with high probability for large *N*, using binary codes. Binary codes can be implemented in software at high speeds because bit operations such as XORs and shifts work very fast on general purpose CPUs. However, the restriction that *N* be large limits the applicability of the probabilistic techniques to our fault tolerance mechanisms. Work in progress in the PET project suggests that these ideas may be extended to deterministic binary maximum distance separable codes, which would have all the properties of Reed-Solomon codes, but run considerably faster. There is reason to believe that, with better coding techniques and faster general purpose CPUs becoming available every year, in the near future it may be possible to implement these codes at sufficiently high speeds in software.

Priority Encoding Transmission

The basic idea behind Priority Encoding Transmission (PET) is to provide different levels of redundancy to the different layers in a hierarchically encoded video stream. Various underlying FEC coding technique may be used to provide the redundancy. In (1), the coding is based on interpolation of polynomials. A faster implementation (39) is based on the Reed-Solomon codes described earlier, and work is in progress to speed up the implementation using the probabilistic and deterministic coding techniques described in the previous section. By appropriately encoding the different layers of the hierarchically encoded video with different levels of priority, we can create a combined stream with some overhead with respect to the original stream, depending on the levels of redundancy added. The stream has built-in redundancy, so it may be transmitted on a network which does not provide any further loss protection. On arrival at the destination, depending on the number of packets lost, we can extract some of the higher priority streams. The number of streams which can be extracted depends on the degree of redundancy provided to each of the streams and the number of lost packets. Thus, we can trade-off the bandwidth of the resulting stream against the level of tolerance.

To give a concrete example, consider an MPEG stream that consists of only I and P frames, and header information.[1] The size of the header stream can be neglected compared to the size of the other two streams. Let us assume an average I frame of 25 Kbytes and an average P frame of 5 Kbytes. Assume that the sequence of frames composing the group of pictures (GOP) are IPPPPPPPPP, which is repeated three times a second, giving us a 30 frames/second video stream. Thus, the I stream is 75 Kbytes/s, and the P stream is 135 Kbytes/second, for a total average stream rate of 205 Kbytes/second.

We may choose to send the header information with 3 times redundancy, the I stream with 1.5 times redundancy, and the P stream without any redundancy. The overall capacity requirement of the stream after PET encoding is 1.17 times the original. Any one third of the packets over which the information of an individual I frame is distributed may be lost without affecting the correct and complete reception of that I frame. However, the loss of any packet would affect the corresponding P frame. The header has the highest level of redundancy, since it is needed to decode any stream at all, but it occupies a very small fraction of the bit stream. The PET scheme also includes appropriate packetization techniques that spread out the original and redundant information among the packets, so that really *any* one third of the packets corresponding to the GOP can be lost without affecting the reconstruction of the I frame.

6.3. Schemes for fault tolerance

In this section, we will attempt to set up a framework for our experiments by classifying fault tolerance schemes along four dimensions. Then we will describe the individual schemes in some detail, and show how the application of various combinations of the levels of these four factors give us schemes with different properties and different capacity requirements.

[1] The header contains information such as the color map and is necessary to use any of the other information.

-- --

6.3.1. Classification

The schemes for fault tolerance dealt with in this chapter can be characterized by three variables: dispersity, redundancy, and disjointness. Finally, the schemes with redundancy may be hot or cold. These ideas are explained in this sub-section.

Dispersity

Dispersity is the idea of sending the information across a number of paths in the network. This idea was first proposed by Maxemchuk in (44), as mentioned in Chapter 1. Each message to be transmitted on a dispersity system with N paths is fragmented into N sub-messages. Each of the sub-messages are transmitted on one of these paths. At the destination, the original message is reassembled. The advantages of spreading the information out on N paths are:

- The transmission time is reduced to approximately $1/N^{th}$ of its single-path value.
- The load on any one specific path is smaller.
- The effect of bursts is spread out over the network.
- In the event of a network failure, the transmission capacity of the aggregate system is only partially affected.

Note that this system is not transparently fault-tolerant. It merely reduces the effect of the failure on the client.

In the context of a realtime network, the different paths of a dispersity system are subject to realtime constraints. These constraints can be met by using realtime channels as the component paths of the dispersity system. In other words, after choosing the set of paths which form the dispersity system, the realtime network performs channel establishment on each of the paths. We will use the term *sub-channel* to refer to one of the realtime channels of a dispersity system. One of the constraints the system should satisfy for efficient operation is that the paths should have roughly equal delays bounds. This should be a criterion in the path selection process, as well as in the choice of the end-to-end delay requirements specified in the establishment process for the sub-channels.

If the delays are not equal, buffers will be required at the destination to equalize delays, so as to deliver the packets in order. Buffers will also be required to smooth out network jitter. If the realtime network provides jitter control, the buffer requirements at the destination can be reduced. We will work out these requirements in the next section. An incidental benefit of this process of delay equalization is that the network jitter is removed from the information stream.

Since we use realtime channels as the component paths of the dispersity system, the delivery of messages at the destination is bounded by the performance guarantees in the absence of faults. Even in the presence of faults, the performance of the surviving sub-channels are still guaranteed. We will show how we can use this to provide guaranteed service to the application using the dispersity system.

The number of paths chosen, N, is one of the variables that characterize the system.

Redundancy

Redundancy is the idea of sending more information than the message, in order to be able to reconstruct the message in the event of loss in the network. It may be used independently of the idea of dispersity, by adding FEC to a single realtime channel. In conjunction with dispersity routing, it can be used by sending the redundant information along a separate path. For example, of the N paths, only K may carry the message stream. Thus, for a given message, the system would break it into K equal sub-messages and transmit them on the K paths. The rest of the paths may be used to transmit redundant information, which would be used to reconstruct the original message in the event of loss of some of the K original pieces of the message. A simple redundant system can be designed with $K = N - 1$. In this case the single redundancy sub-channel carries a bit-wise parity calculated over the $N - 1$ pieces of the message. If any $N - 1$ of the total sub-messages arrives at the destination, the message can be reconstructed. Error correction codes which work for arbitrary N and K exist. In the case of maximum distance separable codes, if any $N - K$ of the messages are received, the message can be recovered.

The variable K in relation to N defines the degree of redundancy. $K = 1$ is corresponds to duplicating the same information on N channel, uses N times the bandwidth of a non-fault-tolerant realtime channel, and has the largest fault tolerance. $K = N$ corresponds to a dispersity system without any redundancy.

A dispersity system with redundancy requires N/K times the bandwidth of a non-fault-tolerant realtime channel with the same traffic and performance requirements. When redundancy is used in combination with dispersity, we get the following additional advantages as compared to a dispersity system without redundancy:

- The system is error tolerant. A certain number of the pieces of the message can be corrupted or lost without affecting the decoding of the message. The number depends on N, K, and the error-correcting code. It can be no larger than $N - K$.
- The system is transparently fault-tolerant. A certain fraction of the paths can be affected by failure, without interrupting the flow of the information. Again, the specifics of the error-correcting code determine the number of failures that can be tolerated.

Moreover, since the service of the underlying realtime channels is realtime, we obtain performance bounds on the service provided by the dispersity system. The application sees fault tolerant realtime service, with guarantees on packet-delivery that continue to hold in the presence of restricted faults. The restriction on the types of faults covered depend on the level of redundancy of the system, and the nature of the FEC code used. Examples of types of faults that may be covered include single faults, double faults, and so on.

Disjointness

In general, the paths used in the dispersity systems should be disjoint. For these systems to work, the routing algorithm in the network must be able to recognize channels belonging to a dispersity system, and place them on disjoint paths. If the paths are not disjoint, then many of the advantages of dispersity systems are lost. In particular, the failures of the paths are no longer independent, since, if a shared link fails, two or more paths can simultaneously stop transmitting data. However, disjointness is a very harsh restriction, especially as N approaches the edge connectivity of the network topology. By allowing some links to be shared, we might be able to set up many more connections in the network. We would like to answer the question: can any of the advantages of dispersity routing still be provided after relaxing the disjointness criterion?

If the constraint is completely relaxed, then it is possible for a link to be shared by all the paths in a dispersity system, leading to the undesirable characteristic that if that link fails, the capacity of the system is reduced to zero. Therefore, we must put in some constraint, which, while looser than the strict disjointness constraint, should still allow the dispersity system to have provably good tolerance characteristics. If we allow a link to be used by at most two channel routes, then we know that a single failure will not affect more than two of the paths. Then, by imposing the restriction $N - K \geq 2$, the system can continue to be tolerant to single faults, assuming maximum distance separable codes. Of course, this system would require more network bandwidth than the $N - K = 1$ system with the same capacity. We look at the costs and benefits of such methods below. In terms of our characterization, we can define a variable S, which places a limit on how many paths can share a link.

The routing algorithm in the network must take the variable S into account while routing channels which belong to a dispersity system. Of course, even if the system allows links to be shared (i.e., $S > 1$), the routing algorithm should try to find paths which do not share links, and only use paths with shared links if no paths meeting the delay constraint exist, which do not share links.

Thus, a dispersity system can be characterized by the triple (N, K, S). This triple also tells us how fault-tolerant the system can possibly be. If we use a maximum distance separable code, the system can tolerate up to $\left\lfloor \frac{N-K}{S} \right\rfloor$ faults in the network transparently. For example, the $(5, 3, 2)$ system can tolerate one fault with a maximum distance separable error-correcting code.

Hot vs. cold standby

In addition to the above three variables, a dispersity system with redundancy ($N > K$) can be run in a hot or cold standby mode. In case of hot standby mode, the extra sub-channels actually carry FEC information. In the event of packet loss or failure, the destination can recover without any exchange of messages with the source. In the case of cold standby, the extra sub-channels are not used, except in the event of a failure. This extra capacity can be used to transmit non-realtime traffic during normal usage, with the understanding that, in the event of failure, the capacity will be appropriated for use by the fault-tolerant realtime connection. When the failure occurs, we incur an extra delay before recovery due to the need to inform the source of the failure, so that it can shift the transmission to the back-up sub-channels. However, the capacity is guaranteed to be there, unlike what happens in the reactive fault recovery mechanisms we described in the previous part of the thesis.

6.3.2. Description of schemes

In this section, we discuss in further detail some of the interesting regions in the 4-dimensional space described by the above framework. The gains in service provided by the dispersity systems falling into each region, as well as the attendant costs in terms of increased bandwidth, delay, or implementation complexity, are discussed. We show how to compute the traffic and performance specifications of the sub-channels, as well as bounds on the buffer requirements at the destination. We also mention any additional support, from coding techniques, or from the network that may be required to make each scheme work.

Dispersity routing without redundancy - $(N, N, 1)$ systems.

Schemes within this category spread the data packets out into the network using realtime channels on disjoint paths. If N paths are used, the bandwidth reserved on each path is $1/N^{th}$ of the total required capacity. In the case of network failure, some of the paths survive and continue to carry data while the others are being rerouted. Thus, some lower bandwidth communication is maintained under fault conditions. The system has no overhead in terms of extra bandwidth requirement. However, it suffers from a degradation in service, which can be partially characterized by the loss of $1/N^{th}$ of the bandwidth in the event of a single failure. In the case of video transmission, depending on the compression and/or error-correcting technique being used, the destination may adapt more or less gracefully to this loss instantaneously. In all cases, when the sender is informed, the system can be re-arranged to get the full benefit of the remaining capacity. We will now look at some of the compression techniques, and analyze how they would adapt instantaneously to the failure, even before the sender re-arranges its operation.

The system would work well with video compression codes in which a packet transmitted on a given path is independent of packets on the other paths. Compression codes without inter-frame compression, such as JPEG, fit this model. As long as we make all packets corresponding to a single JPEG frame follow the same path, we have each stream independent of all other streams. In this case, failure of any path would result in a reduction of the frame-rate of the video stream seen at the destination. There would also be a small increase in the irregularity of the displayed stream, since we would see $N-1$ frames displayed smoothly, and then a discontinuity. This could be fixed as soon as the source is made aware of the fault, by adjusting the frame-rate to use the available capacity.

This idea would also work reasonably well with asynchronous video coding schemes, such as the block update scheme described in Section 6.2.1. The following considerations must be kept in mind. The number of paths N should be large, since the fraction of traffic lost in the event of a fault is $1/N$ and the block update scheme is loss tolerant only up to some limit. The source should be notified by the network, and should adjust to using only the surviving links until the fault recovery process restores the failed link. This could be done by scaling back the block compression algorithm to a lower bandwidth as described in Section 6.2.1, and transmitting the information only on the surviving sub-channels. The picture quality after scaling back would be better than the picture quality while $1/N^{th}$ of the packets are being lost.

Compression codes that use hierarchical encoding could also work with $(N, N, 1)$ dispersity systems, in conjunction with Priority Encoding Transmission (PET). By appropriately selecting the levels of redundancy on the various layers in the hierarchical compression scheme, we can create a coded stream which would degrade by layers as the number of faults in the network increases. The example worked out in Section 6.2.2, if split appropriately into 3 streams, and transmitted on a $(3, 3, 1)$ system, would provide both I and P streams in the normal case, and only the I stream in the event of a failure. Note that the degradation is present only during the time it takes the network to recover the failed sub-channel. The overhead in this example is 17%, as worked out in Section 6.2.2.

Audio data, having much lower bandwidth requirements, may be handled using the systems with redundancy, to be described below. Other realtime that can be partitioned into multiple streams in such a way that any of the streams can be dropped without affecting the utility of the other streams can enjoy the graceful degradation characteristics of the dispersity systems without redundancy. The graceful degradation is manifested as a loss of bandwidth when a failure occurs. The guarantees on the other performance bounds provided by the realtime network, such as delay, jitter, and loss due to buffer overflow, continue to be valid.

Realtime data that cannot be partitioned into multiple streams as described above can be carried on such systems with a temporary disruption during failure, notification of the source and destination, and scaling back and adjustment at the source to use the surviving paths. However, since the round-trip delay of the channel has to be small in order to be suitable for interactive use, the latency of this notification process will also be small, and may be acceptable for many applications. In addition, the notification time can be bounded, at least statistically, by reserving resources for the control activity.

Since the coding step is absent for these systems, we do not need to fragment each message across the paths. Rather, each message may be sent on a single path, and the different messages may be sent round-robin on the paths in the system. This saves us the fragmentation overhead. Thus, messages are sent on the sub-channels at a rate $1/K^{th}$ the total rate, but the message size remains the same. No delay is added due to fragmentation or encoding. Thus:

$$\begin{aligned} X'_{min} &= X_{min} \cdot K \\ X'_{ave} &= X_{ave} \cdot K \\ I' &= I \\ S'_{max} &= S_{max} \\ {D_{max}}' &= D_{max} \end{aligned} \tag{6.1}$$

The number of buffers needed at the destination to equalize the delays between paths, absorb jitter, and reorder the packets can be computed using the following formula:

$$B = \left\lfloor \frac{{D_{max}}' - \min_{j<N}(Dp_j)}{X'_{min}} + 1 \right\rfloor, \tag{6.2}$$

where ${D_{max}}'$ is the delay bound on the sub-channels in the dispersity system, as calculated in Equation 6.1, Dp_j is the minimum delay (transmission + propagation) on the path for sub-channel j, and X'_{min} is the minimum inter-packet time on the sub-channels. B is an upper bound on the buffers required. We will compare the buffers actually used in the simulation experiments to this bound. The buffer space for reassembly should be large enough to hold B packets from *each* sub-channel in the dispersity system. The memory required is, thus, $N \cdot B \cdot S'_{max}$.

The buffer space requirement at the receiver could be reduced in a network that implements jitter control by taking advantage of the fact that no packet can arrive before $Dmax' - Jmax'$ at the destination. Thus, the number of buffers required is reduced to:

$$B = \left\lfloor \frac{J_{max}{}'}{X'_{min}} + 1 \right\rfloor \tag{6.3}$$

Our simulation did not implement jitter control, but a description of how jitter control may be provided in a packet switched network may be found in (31).

Dispersity routing with redundancy - $(N, K, 1)$ systems with $K < N$

Dispersity systems with redundancy operate by breaking each message into K packets of equal size. Then, they compute $N - K$ redundant packets using an appropriate coding technique, and transmit the N resultant packets onto N realtime channels. At the destination, the original message can be reconstructed from any K packets, if the coding technique used in maximum distance separable.

Schemes within this category use N/K times the network bandwidth, compared to the bandwidth required to support a non-fault-tolerant realtime channel with the same specifications. These schemes use the FEC information on the $N - K$ extra paths to recover instantaneously and transparently from up to $N - K$ faults. The bandwidth overhead of such a system is, thus, $\frac{N - K}{K}$. For example, a one megabit per second video stream carried on a $(3, 2, 1)$ system would require three sub-channels each with half megabit per second capacity.

Note that we can build systems with fault-tolerance identical to a PET stream being carried on an $(N, N, 1)$ system as a combination of different $(N, K, 1)$ systems. The example provided for PET in Section 6.2.2 is equivalent to a combination of a $(3, 1, 1)$ system for the header stream, a (3,2,1) system for the I stream, and a (3,3,1) system for the P stream, using the same set of three paths. While the details of encoding are different, the amount of fault and error tolerance provided is the same.

The systems for $K = 1$ and $K = N - 1$ are easier to implement, since $K = 1$ corresponds to sending the same data on all N sub-channels, and $K = N - 1$ corresponds to the parity system. The other systems would need some hardware support, such as described in (5), or fast CPUs to perform the coding and decoding in software. New coding techniques are under investigation (39), which might solve this problem. The systems with $K = N - 1$, which we will call the parity code systems, are currently the most practicable systems, given CPU speeds and the complexity of the algorithms for coding and decoding maximum distance separable codes.

Dispersity schemes with redundancy provide completely transparent fault-tolerant service to the application. For fault conditions that are within the recovery ability of the error-correcting code (less than $N - K$ faults, assuming maximum distance separable codes), no degradation of service is seen on. Since we use realtime channels as the component paths of the dispersity system, all messages are delivered according to the performance bounds guaranteed to the application, even after a failure has occurred. Thus, the service provided is fault-tolerant *realtime* service, within the strict Tenet definition of realtime; that is, the guarantees provided are mathematically provable. The ability to recover transparently from the loss or corruption due to transmission errors of $N - K$ packets from the same message is also provided as a side benefit.

The parameters to be used for establishing the sub-channels of the dispersity system with redundancy can be calculated from the parameters of the request as follows. Since each message on the entire system causes a packet to be transmitted on each sub-channel, X_{min}, X_{ave}, and I are the same as in the original request. The packet size is $1/K^{th}$ of the message size transmitted on the system. The delay for fragmenting and encoding the packets must be accounted for while setting the delay bound for the sub-channels.

For some systems, such as the $(N, 1, 1)$ systems, d_{code} and d_{frag} may be zero or negligible:

$$\begin{aligned} X'_{min} &= X_{min} \\ X'_{ave} &= X_{ave} \\ I' &= I \\ S'_{max} &= \frac{S_{max}}{K} \\ D_{max}' &= D_{max} - d_{code} - d_{frag} \end{aligned} \tag{6.4}$$

At the destination, the time to decode the packets must be added to the end-to-end delay, before the decision about acceptance for the entire system can be made. The buffer requirements at the destination for these systems are the same as described in Equations 6.2 and 6.3.

Dispersity routing with some shared links - $(N, K, 2)$ systems

In case a completely disjoint set of paths cannot be found, the scheme can still continue to work using some partially disjoint paths. Relaxing the constraint thus allows us to set up a larger number of connections in the same network. The increase in capacity provided by relaxing this constraint will be investigated in the simulations. We will only look at systems with $S = 1$ (links may only lie on one path) and $S = 2$ (each link may only be shared by two paths). The routing algorithm is constructed to use shared links only when no other path can be found.

The service offered by such systems in the absence of faults is very close to that offered by the $(N, K, 1)$ systems. If we assume that the loss rates due to transmission errors on on all links are identical, and time-independent, the probability of a loss which is unrecoverable is the same for both systems. However, on real links, bit errors are auto-correlated, and thus the probability of loss due to a transmission error seen by two packets which cross the same link closely together in time are not completely independent. Therefore, the service seen by the $(N, K, 2)$ system may be slightly worse than the service seen by the $(N, K, 1)$ system. This effect would decrease with the packet size, since the auto-correlation would not be observed at the larger time scale of the packet transmission time, and hence, the probability of loss for two different packets would be almost independent.

In the event of the failure of an unshared link, the effect on the two systems is the same. If the link which fails is shared, then this results in a failure of two sub-channels in the $(N, K, 2)$ system. There are three possible approaches to offering a meaningful service in this context.

Firstly, if the system under consideration is a non-redundant dispersity system ($N = K$), the loss in capacity of the connection in the event of a single failure will be either $1/N$ or $2/N$, depending on whether a shared or an unshared link failed. This may be an acceptable service for large N, especially since we are operating in a heavily loaded network, where the better $S = 1$ connection could not be established due to existing reservations. The service after a fault is still realtime, since the performance guarantees are still valid; however, the loss in bandwidth for an $S = 2$ system is larger than that for the $S = 1$ system if a shared link fails.

Secondly, in case the user asks for an $(N, N-1, 1)$ connection, but the routing algorithm returns an $(N, N-1, 2)$ connection, we know this can only happen because an $(N, N-1, 1)$ connection could not be created. At this point, the user may be informed of the paths found. From a knowledge of the number of links that are shared between two paths, and the probability of failure of each link, the user (or the network) can compute the probabilities of an uncovered failure (failure of a link which is shared between two paths) and of a covered failure (failure that affects a link that is not shared). The user can then decide if he wants to accept this limited fault-tolerance, perhaps for a lower cost than for full tolerance. In this case, the fault-tolerance provided cannot be guaranteed, since only some of the failures are covered.

Finally, a third way is to use extra redundancy in the coding, i.e., use an FEC code that can survive the failure of two sub-channels. If the original request is for a $(4, 3, 1)$ system and four disjoint paths cannot be found, the user might also be satisfied with a

$(5, 3, 2)$ system or a $(4, 2, 2)$ system, as long as the total data rate is the same, since the fault and transmission error tolerance characteristics of both systems are at least as good as that of the $(4, 3, 1)$ system. The system is transparently tolerant to all single failures (and can even survive double failures, unless the shared link fails). Realtime guarantees continue to be met without disruption after the fault. The questions that remain to be answered are:

- In a heavily loaded network where an $(N, N-1, 1)$ system cannot be routed, is it likely that we find paths for an $(N, N-2, 2)$ system, given such as system requires more network bandwidth than an $(N, N-1, 1)$ system?
- How much of a detrimental effect would such a trade-off have on the capacity of the network to support realtime connections?

Hot standby systems

In the most general interpretation of the term, all the schemes described so far for dispersity systems with redundancy are hot standby systems. This means that the capacity of the extra sub-channels is used to transmit redundant information, which can be relied on to recover the data in the event that some of the sub-channels fail. However, the common usage of the term "hot standby" is for systems where no encoding or decoding is required for the standby. The most common example of a hot standby system is that provided by a $(2, 1, 1)$ system, where the source transmits the same data down two disjoint sub-channels. The destination uses the data from either channel, whichever comes in first. This allows completely transparent tolerance, to both losses and faults, without the need for notification of either source or destination. There is no degradation of the realtime performance of the system for single faults. In addition, the system needs only two disjoint paths in the network, which is a much easier constraint to satisfy than N disjoint paths for large N. Further, there is no overhead for special fragmentation and encoding at the source, or reassembly and decoding at the destination. No extra buffers are needed at the destination to equalize delays. On the negative side, the system uses twice the network capacity of the non-fault-tolerant realtime channel. This idea could be extended to $(N, 1, 1)$ systems, for tolerance to $N - 1$ simultaneous faults, but with increased capacity overhead.

Cold standby systems

For cold standby operation, an $(N, K, 1)$ system is operated with K sub-channels in use during normal operation. The capacity of the other $N - K$ sub-channels is free for best effort data traffic. In the case of a fault the realtime dispersity system switches the data which was being carried by the affected sub-channel to one of the alternate channels. This involves a latency for informing the source and for the source to switch channels. However, no channel setup is required, since the resources are reserved at the time the $(N, K, 1)$ system is established. Only the packets transmitted by the source on the failed channel after the failure would be lost. All packets transmitted on the backup will be delivered within the performance bounds, since the reservations are ready and waiting. The duration of the disruption would depend on the time for the notification message to reach the source, and could be statistically bounded by reserving resources for the control messages.

No extra support from the network is required to use the resources of the spare sub-channel for best effort data traffic during normal operation. The realtime network is already designed to allow the unused capacity of realtime channels to be used by best effort data traffic. Of course, leaving the spare capacity unused is only useful if sufficient best effort datagram traffic exists to use it.

IFI and SFI channels

Finally, we discuss two other schemes which have been proposed for fault-tolerant realtime networks, in order to compare the network requirements and the services offered to those of the schemes proposed here. These schemes were also mentioned in Chapter 1. Single Failure Immune (SFI) channels, proposed in (73), provide immunity to single failures, but use roughly three times the capacity required by a non-fault-tolerant realtime channel. The basic idea is to start with a realtime channel, remove single links from it and run routing algorithms to find the smallest additional set of links, which would allow the system to keep running if that link failed. The routing algorithm used to add the links is similar to *link rerouting*, but this process is performed *a priori* during establishment, rather than at the time of the fault. At each stage, the smallest set of links is added, which would allow the system to survive the fault, subject to the delay constraints. This process is repeated for all the links in the original realtime channel. Finally, resources are reserved on each link in the system. The aggregate set of links has the property that in the event of failure of any single link of the original realtime channel, the data stream can be rerouted by the node adjacent to the failed link, to follow an alternate path which consists only of links within the aggregate set of links (excluding the failed link), and still obey the delay constraints. In the example provided in the paper, the fault-tolerant channel has close to three times the number of links as the original realtime channel. Thus, under the approximation that the delay constraints on all the links are close to each other, roughly three times the resources are required by the SFI channel as compared to the non-fault-tolerant realtime channel. However, since the scheme is a cold-standby one, the reserved capacity can continue to be used by best effort traffic until the fault occurs. The system requires network support to allow the intermediate nodes to update channel identifier mappings after detecting the fault. The disruption of service is limited to the amount of time to perform this activity.

This idea is extended in (74) to cover isolated failures, i.e., to provide Isolated Failure Immune (IFI) channels. Node faults are defined as isolated as long as the source and destination are not faulty, and no two adjacent nodes are faulty. A link is defined to be faulty not only if it fails, but also if the node to which the link leads fails. Link failures are said to be isolated if any two faulty links are not originating from the same functioning node, or directed to the destination node of the channel. The routing constraints of SFI channels

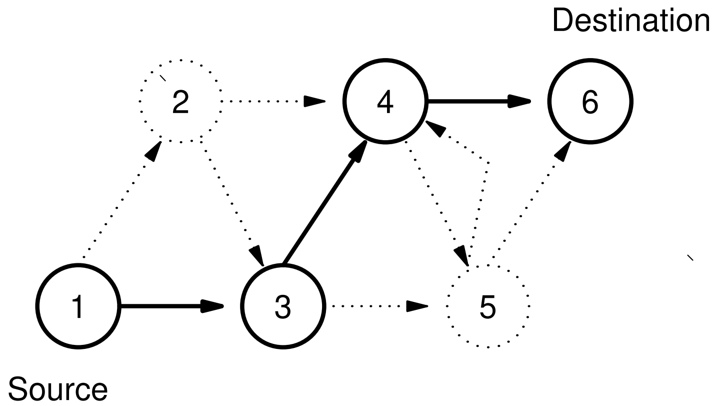

Figure 6.1: One set of isolated link failures which the IFI channel can survive

are modified to extend the fault tolerance to all sets of faults which do not violate the restrictions defined above. The overhead (number of extra links added to the realtime channel) continues to be roughly three. The other restrictions of the SFI scheme, such as network support to change the channel identifier mappings to make sure the data follows the new path in the event of a fault, hold. The IFI scheme offers a better service, since tolerance to multiple faults is provided. In fact as Figure 6.1 shows, a very large number of links can fail before an IFI channel is disabled. However, the IFI techniques are only valid in very restricted network topologies, since IFI sets of links do not exist in most common topologies. For example, in the mesh topology of Figure 6.2, the failures of links (2,3), (7,8), (12,13), (17,18), and (22,23), while isolated, break the graph into two disconnected components. Even if reservations are made on all of the links of the graph, the result is still not IFI. Thus, no IFI set of links exists in this (and most) graphs. The topology of the HARTS network, in the context of which the IFI channel is defined, is a wrapped hexagonal mesh. This is the only commonly-used network topology in which IFI channels exist; thus, the applicability of the algorithms and theorems of (74) is very restricted.

6.4. Simulator design

The simulator described in Chapter 4 was modified to conduct experiments concerning the fault tolerance provided and the network resources used by the dispersity

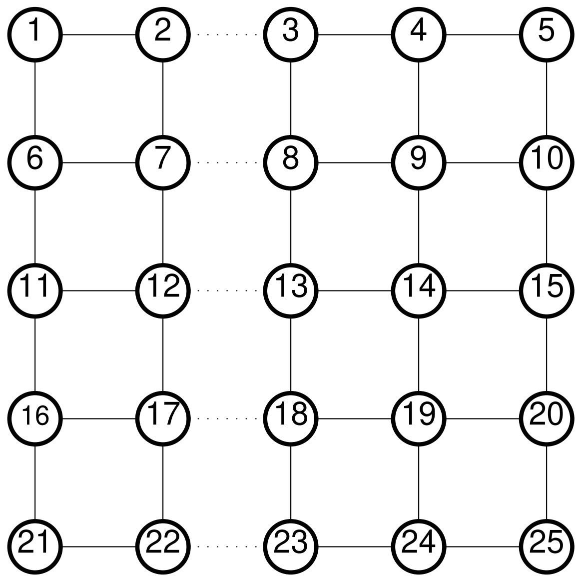

Figure 6.2: No IFI set of links exists in a square mesh topology

systems described in this chapter. The following changes to the simulator had to be implemented.

- The parameters for the sub-channels are calculated according the dispersity systems being simulated, as described in Equations 6.1 or 6.4.
- The routing algorithm had to be modified to route the sub-channels belonging to a dispersity system on disjoint paths. If disjoint paths cannot be found links, can be shared by at most S sub-channels. The routing algorithm attempts to keep the number of such shared links as small as possible.
- The routing algorithm tries to find paths with equal delay. Any differences in the delays on the different paths are equalized by buffers at the destination. Statistics are maintained about the number of buffers used.
- The generation of a packet at the source causes the transmission of N sub-messages, one on each path in the dispersity system.
- The statistics maintained by the simulation depend on the dispersity system being simulated. For example, for a $(5, 3, 1)$ system, the message is counted as being

successfully delivered to the application, when any three of the sub-messages corresponding to the message have arrived.

Routing for the sub-channels of a dispersity system is performed one by one. The first sub-channel is routed as a normal realtime channel. The additional constraints of disjointness are added for each subsequent route calculation. The changes to the routing algorithm were implemented as a set of preprocessing steps on the graph representing the network before the Bellman-Ford algorithm was run. To enforce disjoint paths, the links corresponding to the paths which should not be used are removed from the graph by setting the delay on the path to infinity. To allow links to be shared S times, we maintain a variable U_e associated with each link e, which is the number of times the link e has been used by sub-channels in the dispersity system. When $U_e \geq S$, the link e is removed from the graph by setting the delay to infinity as before. If $0 < U_e < S$, we want the routing algorithm to find paths including e only if other paths do not exist. We replace each such link by n links, each with a delay of $1/n$ times the delay on the original link. Since the algorithm returns the shortest path which meets the delay constraints, it finds paths which do not use the shared links before it find paths including them, up to a difference in path lengths which can be tuned by changing n. We do not change the end-to-end delay on any path by this replacement process; thus, we are guaranteed to find all paths eventually.

The method of reassembly used in the simulator consisted of a circular buffer, each slot capable of holding N sub-messages. The structure is shown in Figure 6.3. A newly arriving sub-message is placed into the appropriate slot in the circular buffer. The index into the circular buffer is the remainder after dividing the sequence number on the packet modulo B. As soon as K sub-messages have arrived for the same sequence number, the message may be reconstructed. Reconstructed messages are passed in sequence number order to the application, and the corresponding slot in the circular buffer is freed for reuse. Alternatively, we may hold each message up to the end of its delay bound, thereby removing all the jitter from the stream. This may be done using time-stamps on packets and the knowledge of the delay bound D_{max}, if the clocks on the source and destination machines are synchronized, as may be done using a time protocol such as NTP. Since we know that no sub-message can be delayed by more than $D_{max}{}'$, the number of buffers required is bounded, and we are guaranteed not to wrap around in the circular buffer. Of course, the reassembly code checks this condition, to ensure that older packets do not get assembled with sub-messages from current messages. Such a case would be counted as a violation of the delay bound guarantee, and the old packet discarded.

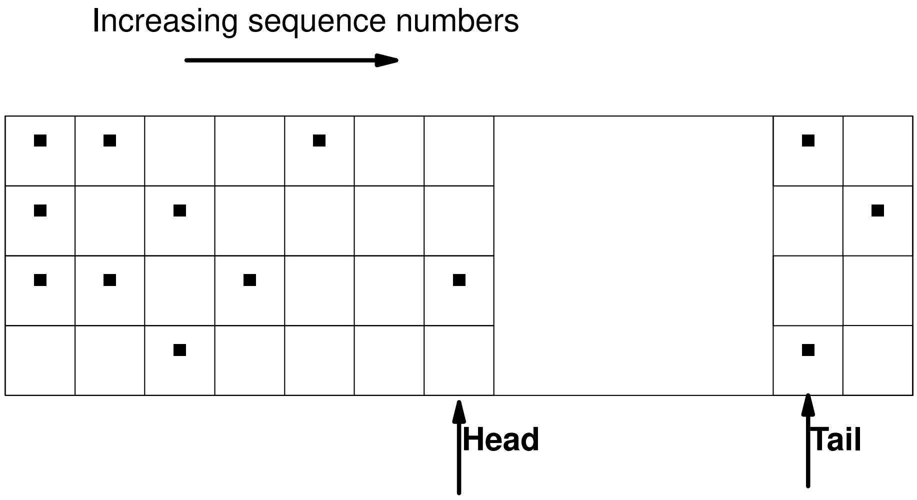

Figure 6.3: Structure of reassembly buffer for $N = 4$

The simulator models this behavior at the level of detail of individual packets. Thus, the encoding and decoding operations, which occur at the bit-level, are not simulated. The transmission of a message on the dispersity system is modeled as the transmission of packets on each sub-channel of the dispersity system, but the bit level contents of the packets are not simulated. The arrival of the appropriate number[2] of packets from the same message at the destination is interpreted as a successfully decoded message, and the statistics are updated accordingly. Encoding and decoding times are not modeled,

[2] Depending on the dispersity system being simulated.

since they are dependent on the coding algorithms, which are still under development, and on the hardware available on the workstation.

We modeled the process of notification of the source to initiate switching, since a fault message is sent from the detector to the source to initiate fault recovery, and the same message can be assumed to trigger the switching for the cold standby systems. The process of switching itself can be assumed to take a negligibly small amount of time, as compared to the notification time, which involves network communication.

The requirements of a realtime dispersity system are described, in the event file from which simulation is started, by the end-to-end traffic specifications and performance requirements and the variables N, K, and S. The parameters of the sub-channels are calculated as defined in Equation 6.1 or Equation 6.4. The additional restrictions on the routing algorithm are automatically generated, using the routing extensions described above.

6.5. Simulation results

The experiments performed using the simulator fall into two categories. The first set of experiments are intended to verify the level of fault tolerance provided by the dispersity systems. Since the reservations are pro-active, and the guarantees provided on the sub-channels are mathematical and deterministic in the absence of losses in the network, we can prove that the service guarantees hold, given that the assumptions about the fault(s) and other losses are satisfied. The simulations serve to confirm these expectations, as well as to demonstrate the workability of the systems to a fair level of detail. In the presence of network losses due to transmission errors and faults, which are not covered by the Tenet guarantees, the dispersity systems can provide lower packet loss than the basic realtime channel; this performance is experimentally evaluated and compared to values computed from a simple probabilistic model of network loss in the first part of this section. For the dispersity schemes with redundancy, the service provided is mathematically guaranteed even in the event of a failure; thus, we expect that, in the absence of transmission errors, no message will miss its deadline. This is confirmed in our simulation experiments. We also measure the buffer requirements of the dispersity systems, to cross check them against the computed values from Equation 6.2.

The second category of experiments performed deal with the impact of the dispersity systems on network capacity. The simple analysis provided in the previous section shows that the resources consumed by an (N, K, S) system are approximately N/K times the resources consumed by an equivalent non-fault-tolerant realtime channel. In simulation, we compare this result to the number of instances of each type of system that can be established in the network under varying load conditions. This provides us with a better empirical understanding of the effect of the more subtle issues, such as the effect on the network capacity of splitting the resource requirements among N sub-channels on different paths and the probability of finding resources on N disjoint paths.

6.5.1. Performance results

The first set of experiments conducted consisted of setting up one connection using the dispersity system under consideration, in a network carrying different levels of background load, and simulating the service provided in the presence and absence of faults and losses due to transmission errors in the network. The network being simulated is the "core" topology discussed in Chapter 4. The background load is obtained by choosing a set of channel requests from the "original" distribution of Chapter 4. Zero or one faults are simulated, depending on the experiment. Losses due to transmission errors are simulated by dropping each packet with a given probability on each link. Though the simulator supports specifying loss probabilities for each link separately, in the experiments described here all links were assigned the same loss probability.

All the experiments described in this section were performed for all of the dispersity systems. The performances observed for all the dispersity systems meet the expectations of Section 6.3.2. We will not present all the results for each separate dispersity system. Rather, we will use a running example, to present some specific results. The general

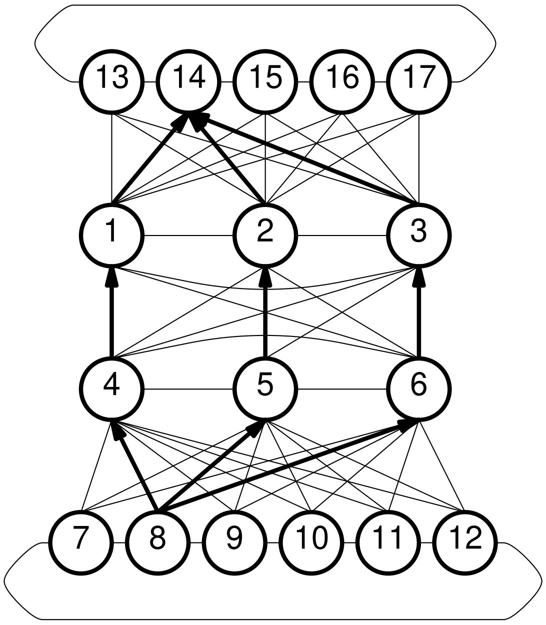

Figure 6.4: (3, 2, 1) **dispersity system simulated**

conclusions presented below from the specific experiment are applicable to the entire set of dispersity systems, unless otherwise noted. Where there are differences, we will show the performance of some of the other systems as well.

The example system is a (3, 2, 1) dispersity system, established from node 8 to node 14 in the network depicted in Figure 6.4. The traffic parameters for the fault-tolerant dispersity system were $X_{min} = 5$ ms, $X_{ave} = 5$ ms, $I = 500$ ms, and $S_{max} = 10000$ bits. This corresponds to a 2 Mbps JPEG compressed stream, called a class A channel in Chapter 4. The end-to-end delay bound for the system is seventy milliseconds, and the minimum propagation and transmission delay is thirty-five milliseconds. B as computed by Equation 6.2 is eight. The simulator does not support statistical delay bounds, jitter control, or variable buffer overflow probabilities. Thus, by default, $Z_{min} = 1$, $J_{max} = D_{max}$, and $W_{min} = 1$. In terms of Figure 6.3, we need at the destination (node 14) a circular buffer eight deep and three wide. We have $S'_{max} = 5000$ bits or 625 bytes. This is the size of each buffer unit in the circular buffer. The total memory needed is, thus, 15,000 bytes.

The simulation was performed for zero and one fault in the network, at various probabilities of packet loss on the links, and various extraneous loads in the network. The network load was created by establishing varying numbers of simple realtime channels in the network, before starting the observation of the fault-tolerant system. The measurement was performed by allowing the experiment to run for 39 seconds of simulated time. This took an hour or more of actual time, depending on the speed of the machine running the simulation and the other load on the machine, because of the level of detail being simulated. The simulator kept track of the number of packets dropped, the number of messages decoded, the buffers used, and so on. When a fault was simulated, all packets on the failed link were lost. The fault recovery process described in Chapter 4 was activated, and the affected channels rerouted. Unlike the experiments in Chapter 4, data transmission across a link could also fail randomly with a probability determined by a parameter of the experiment, simulating losses due to transmission errors. At simulated time equal to 39 seconds, the measurements were written out to a file. Further simulation, involving tearing down the affected channels to compute the efficiency of resource usage, was performed, and at simulated time equal to 43 seconds, more statistics were collected and the simulation was stopped.

The general results which apply to all the experiments performed, but for which no graphs need to be shown, are described first. The service provided met the realtime guarantees in all cases. No packets were delivered after the delay bound. In the absence of losses due to transmission errors, no messages were lost for zero and one link failures. These observations were not affected by the level of the extra realtime load on the network, since the guarantees on the sub-channels hold irrespective of the other traffic.

In the experiments in which failure was simulated, one of the links in the central core was removed from service. This caused the fault recovery process described in the previous chapters to be triggered. The fault message reached the source in under 20 ms, because of the topology and proximity of the failed link to all sources. This is a measure of the time for notifying the source to perform switching for the cold standby sources, and is dependent on the topology. The fault recovery process also rerouted the failed sub-channel to an alternate path. In all our experiments, the reroute was successfully completed.

We also measured the maximum buffer occupancy in the circular buffers during the simulation. No more than eight buffers were needed for the running example of the $(3, 2, 1)$ system, which matches the calculated bound exactly. The actual usage was less in the experiments with no faults and low loss rates due to transmission errors, since the maximum buffers are only needed when a message fails to be successfully decoded until its deadline. In case more than $N - K$ sub-messages from the same message are lost, the reassembly algorithm waits until the delay bound for the message expires. The message is then deemed lost, and the buffers for the sub-messages are freed. In the meantime, other messages may have come in with the minimum propagation delay in the network. Thus, in the case of experiments involving significant loss due to faults or transmission error the buffer size is predicted by Equation 6.2 exactly. The load due to the other realtime channels in the network did not affect the buffer requirement at all.

The numbers of buffers required by all the dispersity systems shown here are fairly moderate. The calculation for B is identical for all of them, since the delay bounds and the minimum propagation delay on the shortest path are the same. This analysis was confirmed by our simulations in all cases. The only difference in buffer requirements among the various dispersity systems is the width of the buffer, which is given by N. Thus, the largest amount of memory (25,000 bytes) is required by the systems with $N = 5$. This amount of memory is fine for most general-purpose computers, and would also be acceptable for specialized devices like set-top boxes. The lesson here is that, by controlling the end-to-end delay for the component sub-channels, we have reduced the problem of buffering at the destination. In case the buffer requirements do become significant, as might happen for higher bandwidth connections than the 2 Mbps channels we used in our simulations, we can reduce them by using jitter control (Eq. 6.3).

When we introduce losses of packets due to transmission errors in the network, we demonstrate the loss-tolerant nature of the system. The transmission error process used in the simulation is very simplistic, since we drop packets with a fixed low probability on every link. This is not an accurate model of real transmission errors, since the error process displays temporal auto-correlation at the bit level. However, at the longer interval corresponding to the transmission time of a packet, this correlation is much less noticeable. In our experiments, the packet size is 10,000 bits, which would make the probability of loss even for consecutive packets almost independent. Thus, our assumption of stochastic independence for the transmission error process is reasonable.

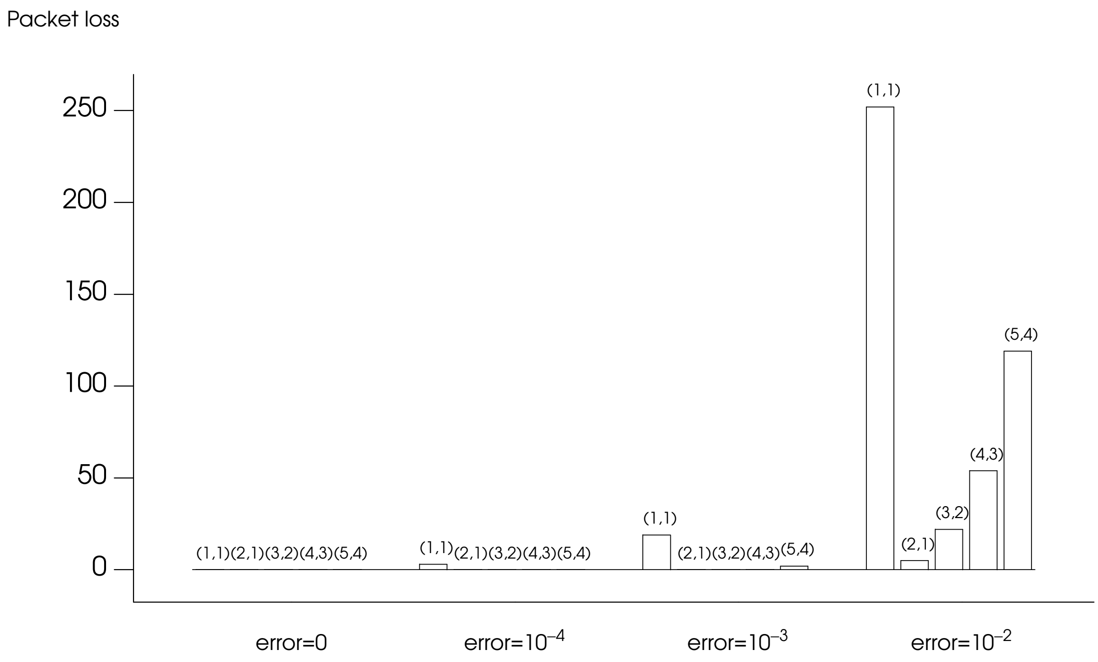

Figure 6.5: Packet loss vs. loss rate for dispersity systems with N - K = 1, no network failure case

Note that the loss process we are simulating does not include losses from buffer overflow at the nodes, since those are controlled by the realtime network. While the observed loss characteristics of packet switched networks do display auto-correlation at the packet level, the main source of such losses is buffer overflow due to congestion at network routers. This source of loss is eliminated in our experiments[3] by the realtime protocols, by reserving buffers on a per connection basis, so that buffers are guaranteed to be present when a realtime packet arrives at any node.

Figure 6.5 shows the effect of the loss rate due to transmission errors in the network on the loss of messages as seen by the application using the dispersity system. On the x-axis, we have the probability of losing a packet on any link. Note that the graph shows loss rates in the network up to a 10^{-2} probability of losing a packet during each transmission on every link in the network. These are not meant to be representative of transmission error rates in real networks. The actual error rates seen in fiber-optic transmission systems are much lower. We simulated high transmission error rates so that we could observe a significant number of lost packets and its effect on the overall dispersity system. On the y-axis we have the number of messages lost by the system. There are 5 bars for each loss rate, but up to a loss rate of 10^{-3}, most of them are of close to zero height. The height of each bar denotes the number of messages lost by the dispersity system due to lost packets. The number of packets lost on the sub-channels is, of course, larger, since a message is lost only if sufficient packets are lost to defeat the redundancy in the system. The tuple shown over each bar is (N, K) for the dispersity system represented by the bar. The variable S is 1 for all the systems simulated, so it is not shown. Only the systems with $N - K = 1$ and the $(1, 1)$ system are shown here. The $(1, 1)$ system is the simple realtime channel without any fault tolerance.

Since the simulator models packet loss by dropping packets with a fixed independent probability (one of the parameters of the experiment) during each packet transmission event, the loss rates are very close to that calculated from the knowledge of the loss rates on each link and assuming independent events. For example, the $(1, 1)$ system is a

[3] Buffer overflow probability is always zero in the performance model implemented in the simulator.

simple realtime channel with no added tolerance. With a loss rate due to transmission errors in the network equal to a probability of 10^{-3} of dropping a packet during a transmission event, and the path length equal to three, the probability of a message being successfully received is $0.999^3 \approx 0.997$. Since 7669 packets were transmitted during this experiment, the expected number of lost messages was 23. 19 messages were observed lost in the experiment. If the experiment were carried on for a longer time, the observed number would converge to the expected mean. The expected number of lost messages for the fault-tolerant systems can be similarly calculated, since the paths are disjoint, and therefore independent. The calculated numbers match the simulated behavior. The number of messages lost is also *independent* of the load on the network due to other realtime channels, because the effect of loss due to congestion is eliminated by buffer and bandwidth reservation at intermediate nodes by the realtime protocols.

The loss process being simulated is extremely simple and the graph for Figure 6.5 could have been easily computed analytically. We present it to give the reader an idea of the error tolerance benefits of the dispersity schemes. The results merely confirm that the simulator is behaving as expected, and demonstrate the workability of the schemes to a certain extent.

Figure 6.6 shows the performance of the dispersity systems when a fault also occurs during the experiment. Now the number of packets lost in the simulation includes the number lost on the failed channel. The system continues to provide realtime performance with increased tolerance to loss as compared to the non-fault-tolerant realtime channel, but the losses noted are slightly higher. The loss is controlled by two mechanisms. First, the duration of the outage of a sub-channel is limited by the *recovery* process that reroutes the faulty sub-channel. We will return to this point in the next section. Secondly, the redundant coding provided (for systems with $N > K$) controls the number of lost messages during the outage. Thus, though the non-fault-tolerant channel (the $(1, 1)$ system) lost 15 messages during the period of outage in the experiment with loss rate = 0, the other systems did not lose any messages. In a network with extremely high loss rates, the dispersity systems do lose messages, but far less than the system without redundancy. These high loss rates are not expected to be seen in most networks under normal conditions, since we are referring only to losses due to transmission errors here, and the transmission error properties of most current networks (and very likely that of future networks too) are far better than the rates at which we see lost messages in these experiments.

Figure 6.7 shows the performance of the systems with $N - K = 2$ and with a fault during the experiment. These systems are tolerant even at very high loss rates and one fault in the network. These systems also need the same amount of buffering at the destination to equalize delays and reassemble the packets. The cost of using the higher level of redundancy is the higher bandwidth requirement of the system. A $(4, 2)$ system needs twice the bandwidth of a non-fault-tolerant realtime channel, as opposed to a $(4, 3)$ system, which needs only 1.33 times the bandwidth. However, the effect on the network capacity is more complex than can be understood by just considering the bandwidth of the component sub-channels, for reasons which are explored in the next set of experiments.

Overall, in this set of experiments we have shown that, at the level of detail that our simulator models a realtime network, and subject to the limitations in the faults being simulated, the service provided by the dispersity systems meets our expectations, even in the presence of single link faults and reasonable transmission error rates. The realtime guarantees are never violated, and the error and fault tolerance is enhanced. Further, the performance is guaranteed independently of the load, since the component sub-channels of the dispersity systems are realtime channels. With the error rates seen in current transmission systems, any of these dispersity systems will provide adequate protection against single failures. Thus, the choice of which one to use is dictated in part by the question of the degree of tolerance required, that is, whether tolerance against a single fault, two faults, or n-faults is required. The degree of tolerance is determined by $N - K$,

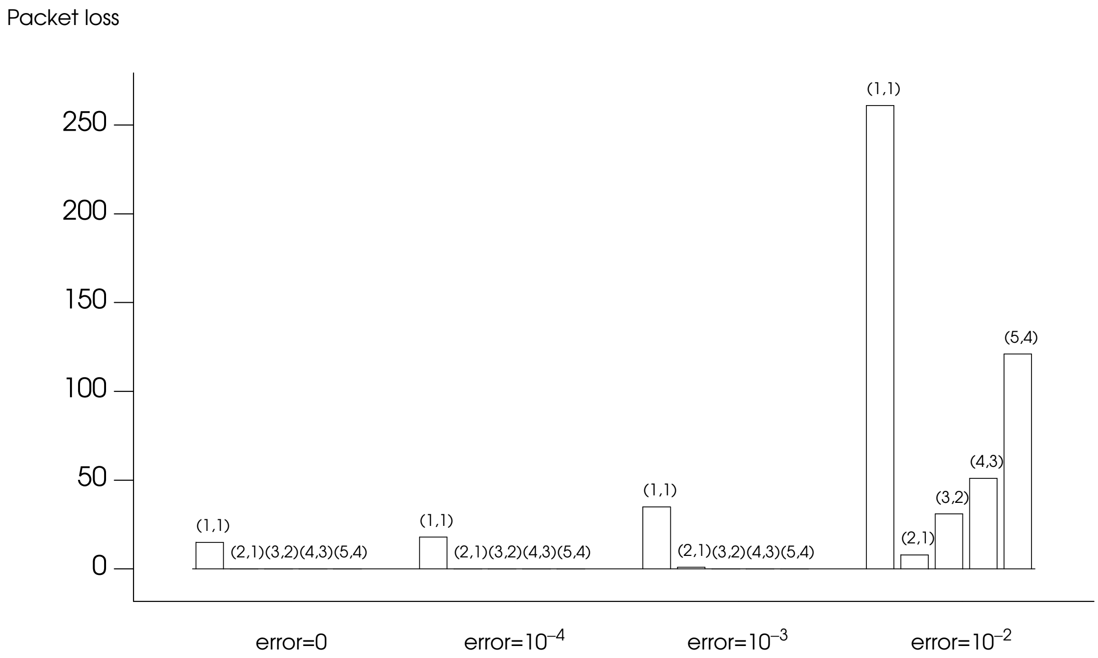

Figure 6.6: Packet loss vs. loss rate for dispersity systems with N - K = 1, one failure

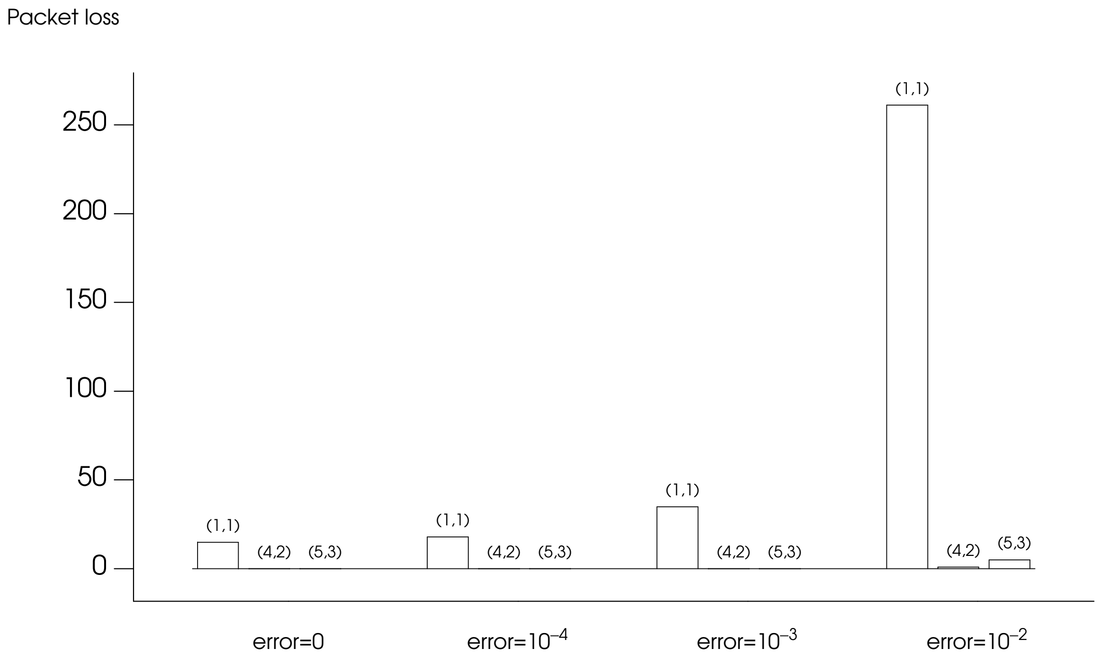

Figure 6.7: Packet loss vs. loss rate for dispersity systems with N - K = 2, one failure

assuming maximum distance separable codes. If the decoding needs to be performed in software using a low-performance CPU, the parity code restricts us to $N - K = 1$ systems. Between systems with the same value of $N - K$, the simple analysis based on the amount of extra bandwidth used ($\frac{N-K}{K}$) seems to suggest that the higher the values of N (and K), the lower the impact on the rest of the network. This issue of network capacity will be explored in the second set of experiments with the simulator.

6.5.2. Network capacity

The effect of the different dispersity systems on network capacity was tested by seeing how many connections of each type could be established starting with a given initial load. The initial load is determined by the set of simple realtime channels already established in the network before the experiment starts, and is fixed by including a prefix with the same set of channel requests in the event file from which the simulation is started. The experiment is conducted for each type of dispersity system, for four initial load conditions. For each value of load, we repeat the experiment for three sets of channels generated from different random seeds, which have the same load index in the "core" network. The results are averaged from these three experiments. The number of connections that can be established is determined by attempting to set up a larger number than can be possibly successful, and then counting the number of successful establishments. The source and destination are chosen from opposite sides of the core topology randomly. The traffic and performance parameters of the dispersity systems are the same as in the last section. The parameters of the background load are picked from the "original" distribution again.

The effect of introducing dispersity without redundancy is explored first. These systems can be characterized by $N = K$. As explained is Section 6.3.2, the service provided by such systems degrades partially in the event of a fault, until the failed sub-channel is rerouted. During this time, a lower bandwidth connection is maintained. The bandwidth requirement of the dispersity system is the same as that of the non-fault-tolerant realtime channel; thus, one might expect that the same number of total connections can be established for each dispersity system as for the non-fault-tolerant realtime channel. However, the simple model based on bandwidth does not take into account the external fragmentation due to the size of the bandwidth requirement of the channel relative to the capacity of the link, and the probabilities of finding multiple paths meeting the delay constraints and having adequate resources. This depends very much on the particular network topology; for example, the fact that we can find five link-disjoint paths is based on the excellent level of connectivity of the "core" topology. It also depends on the background load. The experimental results show the effect of the level of dispersity on the ability to set up fault-tolerant connections for different levels of network load, and allow us to draw conclusions about the effect of the establishment of a fault-tolerant connection on the network's ability to accept subsequent realtime requests, as well as about the level of difficulty of finding the required number of disjoint paths in the network.

Figure 6.8 shows the number of connections of each type which could be set up, for the dispersity systems without redundancy, at different realtime load levels in the network. The $(1, 1)$ system is the non-fault-tolerant realtime channel, which serves as the control for the experiment. We note that, at all load levels, the number of connections that can be set up first increases and then decreases with N. This requires some explanation. The initial increase is due to the fact that each sub-channel in a dispersity system has a smaller-bandwidth requirement than the original request, and the requirement gets smaller with increasing N. Thus, the level of external fragmentation on the links decreases, since smaller bandwidth channels can be packed more efficiently on the links. This effect increases with the load on the network, since the remaining capacity of the network is smaller compared to the size of a single request, leading to more external fragmentation. This effect would be more pronounced in a network with small capacity, and less pronounced in networks where the capacity is much larger than the requirement of a single

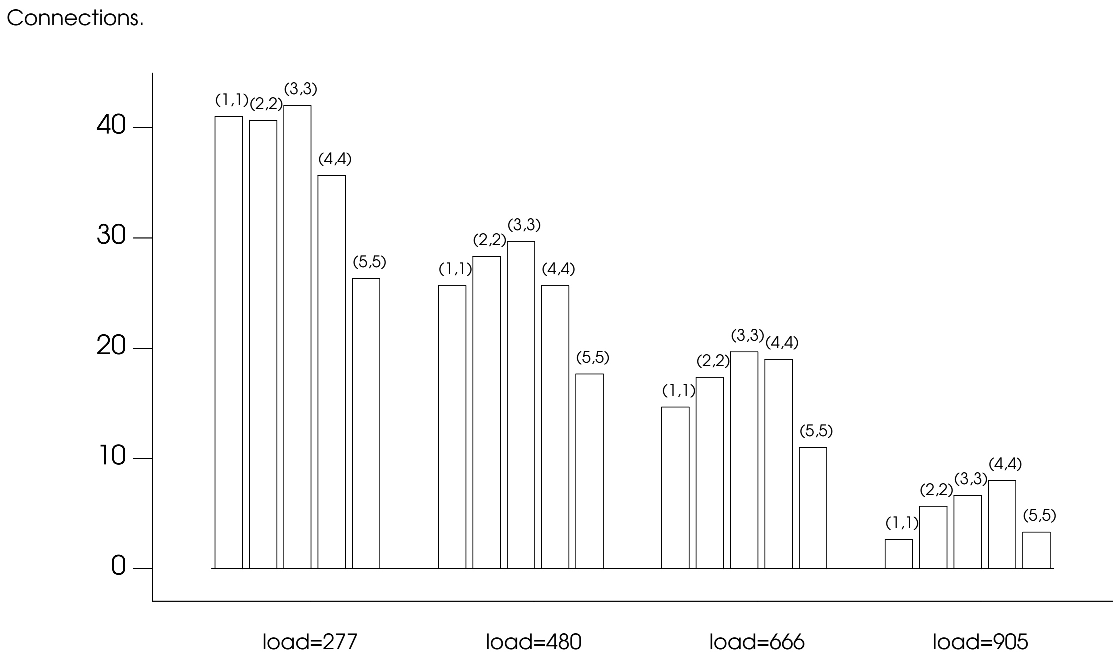

Figure 6.8: Number of connections established for dispersity systems without redundancy

realtime channel.

The fact that the bandwidth requirement of each individual sub-channel is smaller than the total requirement of the dispersity system also has an important effect on the capacity of the network to accept subsequent realtime requests until the dispersity system is torn down. Since the amount of resources used on each path is lower, the effect of the request is spread out over the entire network, so more capacity remains on each individual path as compared to a realtime channel that places all its resource requirement on a single path. The amount of actual traffic from the dispersity system on each path is also lower than the traffic on the single path of an equivalent realtime channel. Thus, the dispersity system is friendly to other users, both realtime and non-realtime, of the network.

The subsequent decrease in number of connections established with increasing N is due to the limited number of disjoint routes in the network. As mentioned before, the maximum number of link-disjoint paths between nodes on different sides of the core topology is five. Thus, as N approaches five, the number of choices between alternative paths for the set of routes decreases. At $N = 5$, we need to find resources on all the available paths; if any path is too highly loaded, the request fails. Thus, the ability to establish a large number of fault-tolerant connections, and the probability of being able to establish one in a highly loaded network, decreases as N approaches the number of alternate paths available. For $N > 5$, no connections can be established in this network. This observation is important, since we do not anticipate that even very large integrated-services networks will have more than four or five disjoint paths available between any source-destination pairs. This shows that the practicality of any dispersity scheme with large N is limited by the network topology.

The number of connections that can be established starting from a lightly loaded network is an indication of the capacity of the network regarding the specific dispersity system. Thus, in this topology, the number of $(5,5)$ connections that can be established is a little more than half the number of non-fault-tolerant realtime channels that can be established. As explained before, this is because N is at the limit imposed by the number of disjoint paths available in the network. On the other hand, we can actually establish more $(3,3)$ connections than non-fault-tolerant realtime channels, because at this stage the effect of reduced external fragmentation is more important. Thus, bandwidth is just one of the three factors that determine the number of connections that can be established in a network.

The number of connections that can be established in a heavily loaded network is indicative of the relative probability of being successful in establishing a connection using a specific dispersity system in a similarly loaded network. In other words, we can interpret the sizes of the bars at load=905 to mean that we can expect to establish a $(4,4)$ connection with greater probability of success than even a non-fault-tolerant realtime channel if the network is heavily loaded. This result is surprising, since it counters our initial intuition that finding resources on four paths successfully, especially when there are only five available, should be harder than finding resources on just one. However, we should interpret the result to mean that at the high load level, because of the small amount of resources left on each link, the influence of external fragmentation is more important than the effect of the disjoint path restriction.

An apparent contradiction can be pointed out in this argument. We note that the number of (4, 4) connections that can be set up at low loads is less than that number for the non-fault-tolerant realtime channels. But we are claiming that at high network loads, it is easier to set up (4, 4) connections. Thus, in the experiment starting at low loads, when the network is becoming saturated after establishing 35 connections, we should be still be able to establish more (4, 4) connections than (1, 1) channels, which is contradicted by the observed results.

The explanation for this apparent contradiction is that the nature of the high network load seen by the (4, 4) connections towards the end of the experiment starting with load=277 is not the same as that seen by the (4, 4) connections in the experiment starting with load=905. The network state in the second case is much more random, and therefore likely to be uniform, because of the distribution of the requests which created the load. They are composed of requests from three classes of applications, most of which are much smaller in bandwidth requirement. Thus, the resources reserved on all the links are much more likely to be smoothly distributed. On the other hand, the set of requests that created an equally high load index in the first experiment is not so random; they are all (4, 4) requests from Class A. The dispersity system breaks the request into four smaller sub-channels, but the capacity requirements of each is still larger than that of most of the requests in the distribution comprising the load in the second experiment, since the "original" distribution is composed mostly of Class B and C requests. The routing algorithm used is deterministic; so, it tends to send the sub-channels to the same set of links. Towards the end of the experiment starting at low load, some of the links had sufficient available capacity, but four disjoint paths with sufficient capacity could not be found, because of the unbalanced distribution of the load. On the other hand, in the experiment starting at high load, the load is evenly distributed on all the links, so that several $(4,4)$ connections can still be established. Thus, we should restate our earlier observation to say that, with the given topology, when the network is highly loaded and subject to the constraint that the load is evenly distributed, it easier to set up a $(4,4)$ connection than a non-fault-tolerant realtime channel.

This leads to the conclusion that the routing algorithm as implemented does not deal effectively with a large number of requests of the same class of dispersity system. The behavior of the network can be improved by suitably randomizing the routing algorithm for all channels, and specifically for the dispersity sub-channels. We needed the deterministic property of the routing algorithm to formally define the load indices, since they are only well defined if the path followed, and hence the load generated, is deterministic for the same network state. We also needed a deterministic behavior to make the experiments repeatable. But in an implementation of the realtime network, some care should be taken to design the routing algorithm to spread out the requests on the available paths, and randomization is one possible way of achieving this.

Figure 6.9 shows the change in behavior when we lift the strict disjoint-paths constraint and set $S = 2$. This allows the routing algorithm to return paths that share links between at most two sub-channels. Thus, in the event of a single fault, depending on which link failed, up to two of the sub-channels may fail. However, while the degradation of service will be more severe, some communication will still continue if $N > 2$. Thus, for $N = 5$, in the event of a failure in a shared link, the surviving capacity of the system would be 3/5 of the original, while in the case of an unshared link it would be 4/5. As mentioned before, intra-frame compression codes such as JPEG can continue to function under such conditions, since the data on one path is not dependent for its decoding on the data from another path. The graph shows that the negative effect of increasing N is

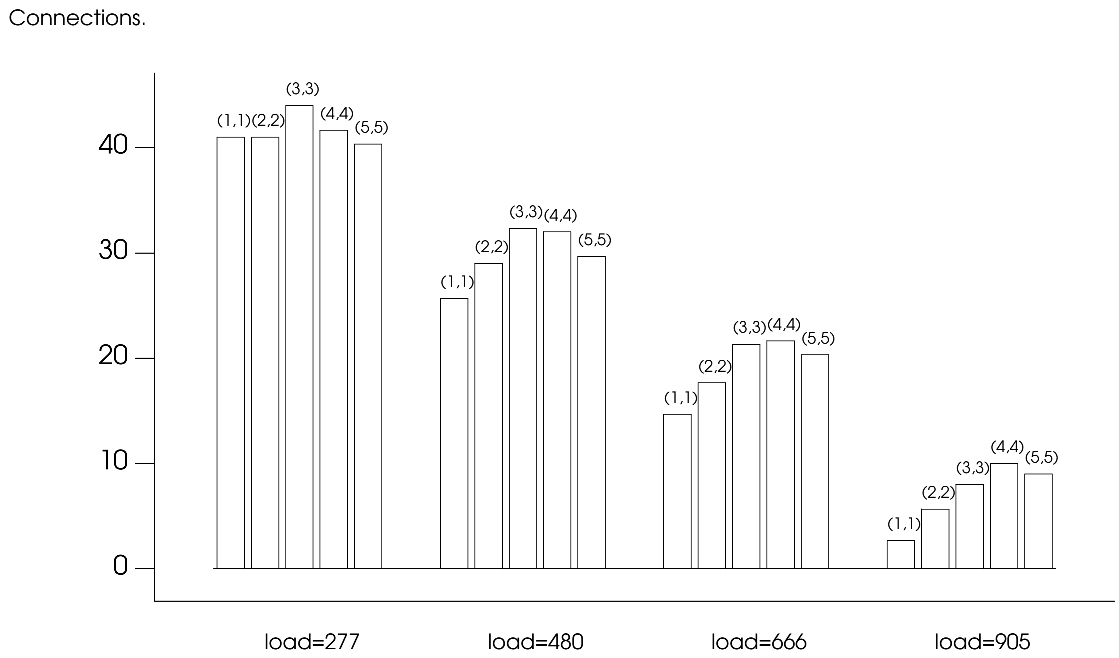

Figure 6.9: Effect of allowing S = 2 on dispersity systems without redundancy

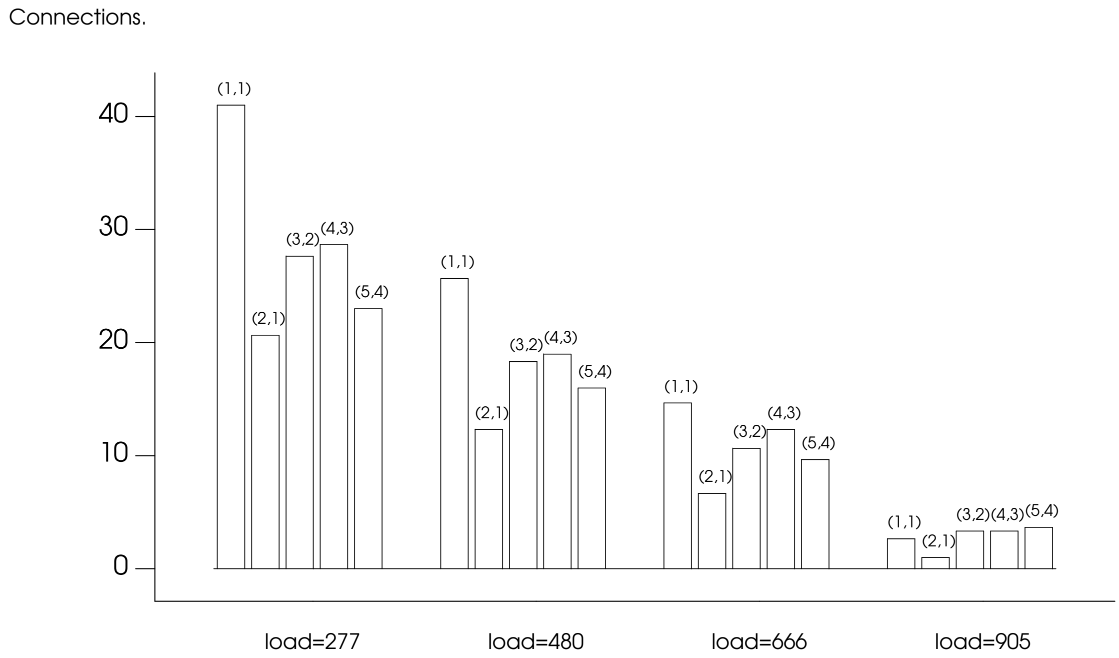

Figure 6.10: Number of connections established for dispersity systems with N - K = 1

greatly reduced, so that even the $N = 5$ system can establish as many connections as the non-fault-tolerant realtime channel in a lightly loaded network. At higher loads, the dispersity systems establish far more connections than the non-fault-tolerant realtime channel. They would also have a lower effect on future connection requests, since the effect is spread out over the network, and not concentrated on one route. The implementation of such systems is simpler than for the systems with redundancy, since no FEC computation is required at the source or destination, and fragmentation and reassembly are also avoided. Thus, these systems have no major disadvantage compared to non-fault-tolerant realtime channels, apart from the slight increase in implementation complexity to handle multiple paths, and the restriction about the kinds of data that can be carried. Only data that can be broken into N streams, each of which can be decoded and used independent of the arrival of the other streams, can enjoy the full benefit of the redundancy systems without redundancy. Other forms of data can also be carried, but in this case service will be interrupted in the event of failure, for the interval of time needed to inform the source, so that it can arrange to use the surviving links.

Figure 6.10 shows the effect of introducing redundancy into various dispersity systems with $N - K = 1$; the $(1,1)$ system is the non-fault-tolerant realtime channel shown for reference. We see that adding redundancy reduces the capacity of the network to support these systems, since they add some overhead to the network. This effect is combined with the effect of the improvement due to reduced external fragmentation, and the effect of the disjoint-paths requirement. The dominant effect at low load and with small N is that of the bandwidth overhead. Thus, the $(2,1)$ system can establish half the number of connections as the basic realtime scheme, and the $(3,2)$ system roughly two thirds of the number. As N approaches 5, the effect of the connectivity of the topology starts being felt, so that the number of connections which can be established for the $(4,3)$ system is less than three fourths. For $N = 5$, the effect of the network connectivity becomes large enough to drive the number of connections for the $(5,4)$ systems below that of the $(4,3)$ system. However, as we increase the load, the effect of reduced external fragmentation becomes more important, since the remaining capacity on some links becomes comparable to the bandwidth requirements of the channels. At extremely high loads, we can actually establish as many or slightly more $(5,4)$ connections as a $(1,1)$ system. At this stage, the effect of reduced fragmentation, caused by splitting the requests into smaller sub-channels, dominates. Note that we do not get the benefit of reduced fragmentation for the $(2,1)$ system at all, since the sub-channels have the same bandwidth requirements as the original. Thus, for this system the total capacity requirements determine the number of channels that can be established at all load levels.

The conclusions we can draw from the above observations are:

- the effect of the increased bandwidth is partially offset by splitting a request into smaller sub-channels;
- this process also decreases the effect the scheme has on other traffic, and on future realtime requests;
- the advantage of splitting the request increases with the realtime load on the network, so that, in a highly loaded network, the dispersity systems can be set up as easily as a non-fault-tolerant realtime channel; and
- the disjoint-path constraint becomes significant as N approaches the number of available disjoint paths, reducing the number of dispersity systems with large N that can be established in a network, as well as making it more difficult to set up a dispersity system with large N in a highly loaded network.

Figure 6.11 shows the impact of allowing $S = 2$ on the ability of the network to accept dispersity systems with $N - K = 1$. We notice that the effect of the network connectivity is reduced, so that, for low loads, the number of connections that can be successfully established is determined primarily by bandwidth considerations. At higher loads, the reduced fragmentation caused by the smaller bandwidth requirements of the sub-channels allows many more connections to be set up for the dispersity systems. In

Connections.

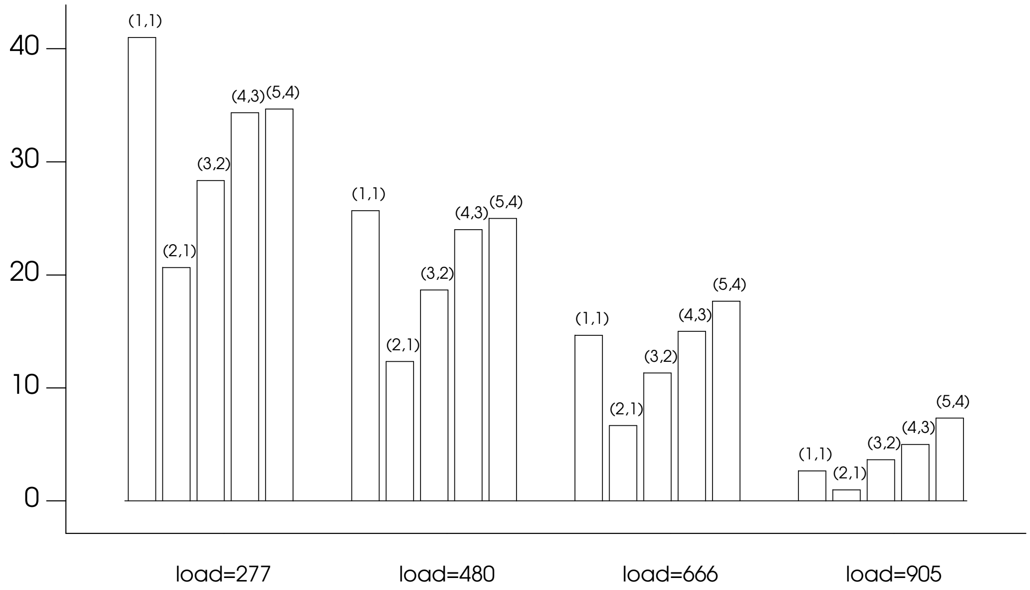

Figure 6.11: Effect of allowing S = 2 on dispersity systems with N - K = 1

Connections.

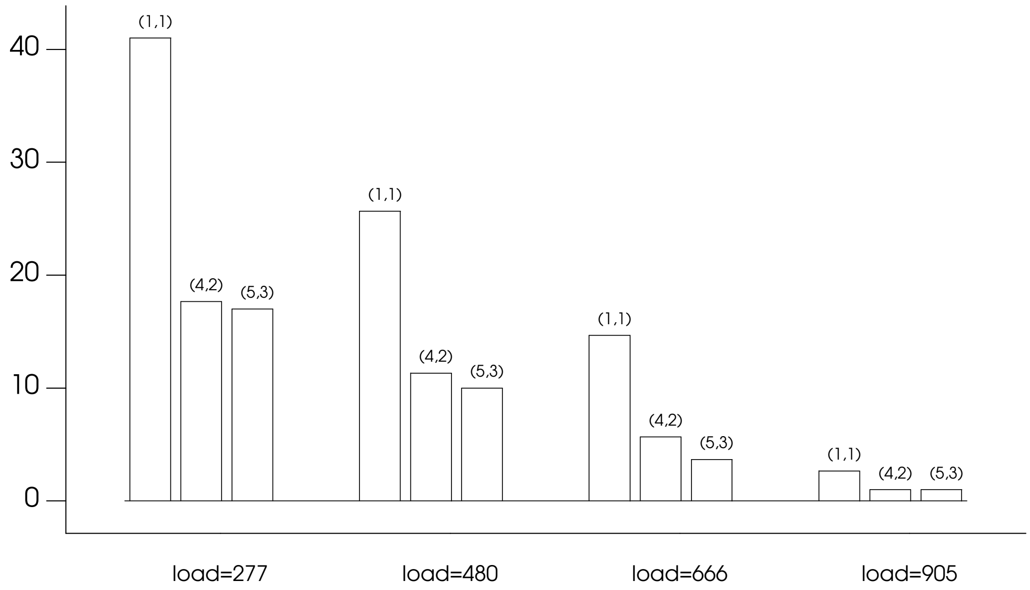

Figure 6.12: Number of connections established for dispersity systems with a N - K = 2

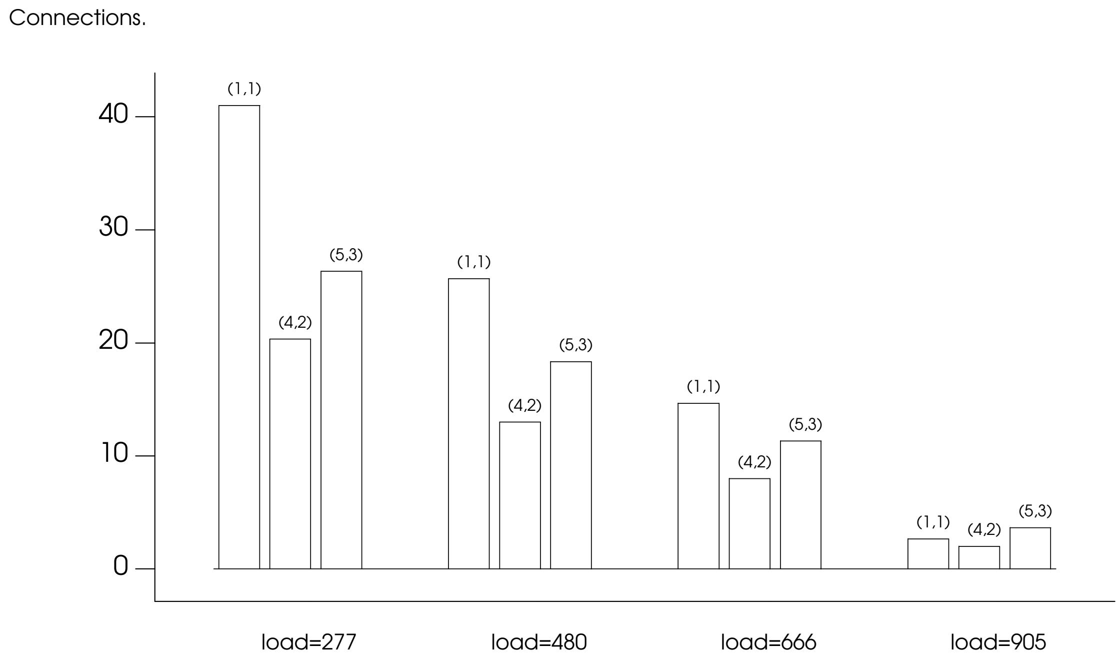

Figure 6.13: Effect of allowing S = 2 on dispersity systems with N - K = 2

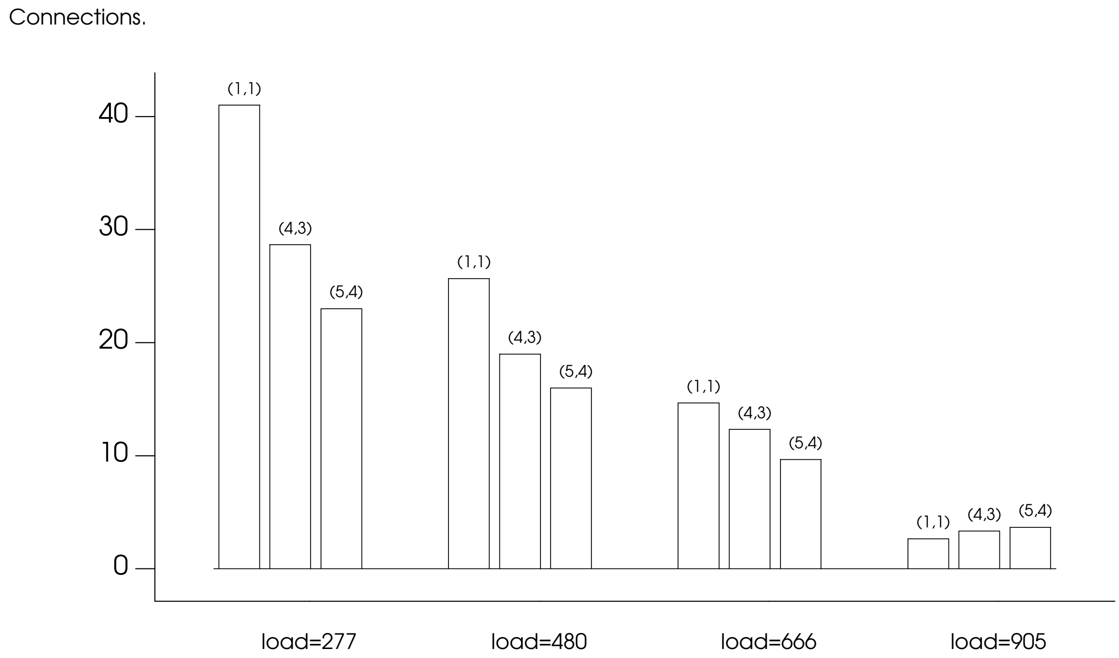

Figure 6.14: Equivalent dispersity systems with no shared links and N - K = 1

fact, the (4,3) and the (5,4) systems are comparable to the non-fault-tolerant realtime channel at low loads, and better at high loads, in terms of the number of connections established. Unfortunately, if a shared link fails, the service provided by such systems is interrupted, so the fault tolerance in these systems only applies to failures of the unshared links in the system.

Figure 6.12 shows the effect of increasing the level of redundancy. The dispersity systems shown have $N - K = 2$; the (1,1) system is again shown for reference. We note that now the effect of the increased capacity dominates, so that, even at very high loads, fewer dispersity connections can be set up than non-fault-tolerant realtime channels. The main reason is that the sub-channels are now as large as the original channel, so we get no decrease in fragmentation. The effect of the disjoint-paths constraint continues to be visible. When we modify the constraint, by setting $S = 2$ (Fig. 6.13), we get no improvement for the (4,2) system. The explanation for this behavior is that the reduced fragmentation effect is not applicable, since each sub-channel has the same bandwidth requirements as the original request. Thus, the only factor is the bandwidth requirement, which restricts the number of (4,2) connections to roughly half the number of basic realtime connections. For the (5,3) system, the fragmentation effect does play a small role, so that, at higher loads, the number of (5,3) connections is comparable to the number of non-fault-tolerant realtime channels. *If* the network had a higher degree of edge connectivity, higher values of N would yield better capacity characteristics for $(N, N-2, 2)$.

The fault tolerance characteristics of the $(N, N-2, 2)$ systems are comparable to or better than those of the $(N, N-1, 1)$ systems. Thus, if one were attempting to establish a (4,3,1) connection, and the network failed to find a fourth disjoint path, one of the options available to the user would be to set $S = 2$, and attempt to establish a (4,2) or a (5,3) system. Comparing the number of connections that can be established in Figure 6.13 and Figure 6.14, we see that the (5,3,2) system may indeed be a viable alternative to the (5,4,1) or the (4,3,1) systems, since the probabilities of establishing connections for these systems in heavily loaded networks seem to be comparable.

6.6. Implementation issues

The mechanisms to support the dispersity schemes can be easily separated from most of the network and transport layer issues. The dispersity mechanisms are mostly end-to-end ones and may be implemented on the source and destination of the connection, on top of a realtime service provided by the transport layer of a realtime protocol suite. In the OSI model, we may place this functionality in the presentation layer, since it deals with issues such as encoding and decoding the data. For the Internet model, we may imagine implementing the dispersity mechanism as a library of functions which the user may load into his applications. The services to be provided by this library would include the control interface to set up fault tolerant channels, the coding and fragmentation needed at the source, and the reassembly and decoding required at the destination.

An interesting question is whether a user-level implementation can operate fast enough to satisfy the typical application's bandwidth requirements, since data may need to be copied several times in the process of fragmentation, coding and finally transfer to the kernel. These issues can only be completely solved by full implementation in the context of a particular operating system. However, we believe that implementation at the user level is feasible, since techniques such as shared memory can be used to minimize the number of times data needs to be copied, and the problems of efficient user-kernel interactions have been successfully dealt with in networking applications before. We will leave these very important problems to future work, and focus here on two subjects: the changes to the control interface to the network, which must be provided to make the task of implementing dispersity systems above the transport layer possible; and the interactions with the fault recovery process described in the earlier part of this dissertation.

6.6.1. Network interface

The network must provide a richer control of the routing process to the higher layers than is currently provided by the control interface offered by the realtime protocols of the Tenet Suite 1. The control interface of the transport layer we assumed in Chapter 4, based on the DCM approach, allowed routing and rerouting requests of realtime channels to be specified, but did not permit the specification of relationships between channels. For the dispersity systems, we need to be able to specify relationships such as disjointness and partial disjointness.

The simplest extension to the control interface, which we implemented in the simulator, includes the optional specification of a set of channels **C** and the variable S, in addition to the parameters already provided by the client, while making a channel establishment request. The dispersity system, establishing connections on behalf of the client, uses this interface to inform the establishment protocol of the set of channels for which the level of disjointness specified by S is desired. This is property of the channel is maintained by the channel management software (RCAP+DCM), as part of the parameters of the channel, so that if the channel needs to be rerouted in the future, the disjointness constraints need not be specified again. The network uses these parameters, and its knowledge of the routes traversed by the channels, to enforce the correct level of disjointness as described in Section 6.4. A more general interface, which allows the specification of an extendible set of relationships, is provided in (36) in the context of the Tenet Scheme 2.

6.6.2. Interactions with fault recovery

One of the factors mentioned in the experimental verification of the loss tolerance provided by the dispersity systems is that the network automatically recovers the failed sub-channels, restoring the connection to full tolerance. The sub-channel affected by the fault is subjected to the same recovery process as the other realtime channels. In all the experiments performed to simulate the effect of a network fault, this recovery process successfully rerouted the affected sub-channel. As we saw in Chapter 4, this process is very fast, so that, for the "core" network topology, the average recovery time is around 100 ms, and the worst case recovery time is around 400 ms. Thus, in most cases, at least in a properly dimensioned network running at reasonable load levels, the recovery will be completed rapidly, restoring the connection to full fault tolerance, so that a second fault occurring at a later time is handled correctly.

However, the recovery is not guaranteed, since, as we observed in Chapter 4, at high loads some of the affected channels may not be successfully rerouted. In addition, the sub-channel has constraints on rerouting, determined by the paths of the other channels and the value of S, These constraints are part of the parameters which the channel management software (RCAP+DCM) maintains, so that when the reroute is requested by RTCMP the correct routing action is taken by the routing module. However, the constraints make it harder to find a valid path. Thus, it is possible that the network will fail to find a path within the limits imposed by the desired recovery response time.

For a non-fault-tolerant realtime channel, it makes sense to stop the retry process after the desired recovery response time has elapsed. However, for the fault-tolerant connections, the service to the application is either not affected at all, or partially degraded, by the failure. Thus, if the recovery protocol described in Chapter 5 cannot find a reroute path, the fault tolerance scheme should, at its own level, continue to try to reroute the remaining sub-channel. This should be done at a low frequency to avoid imposing a high load on the establishment mechanisms of the network. Even if the recovery is achieved several seconds after the failure, it is useful, since the connection is restored to full fault tolerance, so that the service is not interrupted in the event of a subsequent failure.

The mechanisms to continue to generate routing requests for a failed sub-channel, in the event that the automatic network-initiated recovery process fails, should be implemented in the software of the dispersity system. The resources corresponding to the failed sub-channel should be released, to prevent deadlock. This should also reduce the expected waiting time till a channel can be rerouted, since these resources will allow some other waiting channel to succeed. Then, at a low frequency, so as not to impose a high load on the establishment process, the dispersity software should generate routing requests with the appropriate disjointness constraints. Eventually, when the realtime load on the network decreases, the sub-channel will be rerouted and the system restored to full fault tolerance. We take the position that, in a properly dimensioned network, it should rarely happen that the network fails to recover a connection, which then must be reestablished by the dispersity system. If such events were frequent, this would imply that the network was being operated close to its realtime capacity. In fact, in a network where the realtime resources available are limited, so that only few realtime connections can be established, fault-tolerant realtime channels should probably not be offered.

6.7. Conclusions

In this chapter, we have proposed dispersity routing as a mechanism to provide fault tolerance to realtime communication networks. We worked out the detail of the schemes by looking at compression, error correction, routing, and reassembly issues. We presented a framework that we used to classify the various schemes. The schemes discussed include dispersity systems with various levels of dispersity (N), various levels of redundancy (K), and various levels of strictness of the disjoint-path constraint (S), and hot/cold standby systems. We looked at some other existing approaches to providing fault-tolerant realtime communication, to compare benefits and costs. Finally, we presented a simulation model, which we used to study the performance of the dispersity systems and the capacity of the realtime network needed to support them.

The properties of some of the schemes are summarized in Tables 6.1 and 6.2. For the dispersity systems without redundancy (i.e., the $(N, N, 1)$ systems), the service provided in the absence of faults in the network is the same as that provided by a non-fault-tolerant realtime channel. The service provided in the event of a fault is communication with the same delay constraints, but a degraded bandwidth availability. A fraction of the

Property	$(N,N,1)$	$(N,N-1,1)$	$(N,K,1) \mid K < N-1$
B/W Overhead	None	$1/(N-1)$	$\frac{N-K}{K}$
Impact on network capacity	Increased capacity	Slight decrease	More decrease
Level of fault tolerance	None	1 fault	$N-K$ faults
Level of error tolerance	None	1	$N-K$
Duration of disruption	Recovery time	No disruption	No disruption
Service during disruption	Lower B/W realtime	No disruption	No disruption
Routing constraints	Easy for small N, hard for large N	Easy for small N, hard for large N	Easy for small N, hard for large N
Encoding/decoding complexity	None	Low	High

Table 6.1: Properties of various dispersity systems

Property	$(N,N-1,2)$	$(N,K,2) \mid K < N-1$	**hot standby**	**cold standby**
B/W Overhead	$1/(N-1)$	$\frac{N-K}{K}$	1	0
Impact on network capacity	Small decrease	Small decrease	Large decrease	No impact
Level of fault tolerance	Partial	$N-K-1$ faults	1 fault	1 fault
Level of error tolerance	1	$N-K$	1	0
Duration of disruption	Recovery time	No disruption	No disruption	Notification time
Service during disruption	No service	No disruption	No disruption	No service
Routing constraints	Easy	Easy	Easy	Easy
Encoding/decoding complexity	Low	High	None	None

Table 6.2: Properties of some more dispersity systems

realtime packets are lost due to the fault, but the rest reach the destination before their deadlines. For the systems with disjoint paths ($S = 1$), the extent of the degradation is a $1/N$ decrease in capacity; for the systems with $S = 2$, the extent of degradation is either $1/N$ or $2/N$, depending on whether a shared or unshared link failed. These systems require limited buffering at the destination, bounded by Equation 6.2, to equalize delays on the paths. This buffer can be also used to remove network jitter from the message stream delivered to the application, assuming NTP synchronized clocks on the source and destination hosts. The effect of non-redundant dispersity on the capacity of the network is good, since the resource usage is spread out over the network. The number of connections that can be established is larger than for the non-fault-tolerant realtime channels, except when N approaches the degree of edge connectivity of the network. In this case, relaxing the disjointness constraint allows more such connections to be established, with a controlled decrease in the reliability provided. The service in this case can be modeled based on independence of probabilities of failure and error events on each link, allowing simple calculations of the expected fault tolerance to be used in deciding which dispersity system to use. On the negative side, these systems require a small increase in complexity to set up multiple sub-channels, support from the network to do the constrained routing of sub-channels, and reassembly support at the destination. In addition, these systems can only be used with realtime data that can be split into independent streams, so that an entire stream can be lost temporarily without rendering the other streams useless. Video compressed with intra-frame compression schemes such as JPEG satisfies this requirement. Block update techniques can also be used on top of these dispersity systems, since each block can be rendered independently of the others. Scalable media, such as hierarchical video, can also be used in conjunction with PET encoding to provide a system that degrades gracefully with failed sub-channels.

The dispersity systems with redundancy (i.e., $(N, K, 1)$ systems with $K < N$) can be used to provide completely undegraded service in the presence of restricted network faults. They also improve the tolerance of the connection to losses due to transmission errors. The redundancy introduces some overhead in terms of the extra bandwidth required from the network. However, the dispersity spreads this effect over a number of paths, so that the effect of a single sub-channel on the links it traverses can be made smaller than that of the original request. Thus, the effect of the increased overhead is partially compensated. The smaller requirements of the individual sub-channels also reduce external fragmentation on the links, allowing more connections to be established. However, for large values of N the connections are hard to establish because of the disjointness constraint. This can be relaxed by setting $S = 2$, thereby allowing more connections to be established. Since as many unshared links are used as possible, systems with

$S = 2$ may provide some of the fault tolerance properties of disjoint-path systems, in cases where disjoint paths cannot be found, but this tolerance is only applicable to the unshared links in the system. Further, extra redundancy ($N - K = 2$) can be used to improve the fault tolerance, allowing an $S = 2$ system to be fully tolerant to single faults. This gives the user a number of options in case the disjoint paths cannot be found. Dispersity systems with $K = N - 1$ can be implemented using parity codes, which allow deployment on low-end general purpose computers. Systems with $K < N - 1$ require more sophisticated encoding/decoding support.

All the systems described above can be considered hot standby systems. A canonical example of hot standby system is the $(2, 1, 1)$ system, where the same information is sent down to disjoint paths. Whichever stream reaches the destination is used for display. This system is very useful in networks with limited connectivity, since only two disjoint paths are required. However, the system uses twice the network resources that a non-fault-tolerant realtime channel would use. A cold standby scheme, which uses only one of the sub-channels, but reserves resources on the second sub-channel, so that in the event of a failure it can be sure of having the capacity, is less wasteful of network resources. It allows best-effort traffic to use the capacity of the second path in normal usage, but suffers disruption of service in the event of a fault, for a period of time determined by the time needed to inform the source and switch transmission to the standby channel. In WANs, this disruption may be substantial, since the delay-bandwidth product may be large. However, the duration of the disruption less than the time for a round trip on the channel, which must be small if the communication is interactive in nature. Most of the applications of realtime networks fall into this category.

Finally, in comparing the schemes described here with IFI realtime channels, we note that the IFI technique is usable in the highly regular wrapped hexagonal mesh topology of the HARTS network, but, in a general graph, the algorithm would not always find a valid set of links. In fact, in most networks IFI sets of links would not exist. The dispersity technique is much more general, since in most networks we can at least find two disjoint paths between two hosts. Of course, dispersity works better, in terms of resource usage, for larger N, but, if such a set of disjoint paths cannot be found, a number of alternatives exist. Secondly, the IFI scheme uses three times the network resources needed by the non-fault-tolerant realtime channel, while the dispersity systems offer a variety of services, with a corresponding variety of capacity overheads. To be fair, the IFI scheme is a cold standby scheme, so that the resources can be used by best-effort traffic until the fault occurs, but the resultant reduction in realtime capacity is significant. In addition, the scheme requires network support, to allow the nodes to change routing information automatically in response to the failure. It is not clear from (74) what the latency associated with this action might be. Most of the network support needed by the dispersity schemes is already provided by existing realtime protocols, such as DCM; the incremental support required is limited to adding to the network control interface the ability to specify the disjointness constraints.

The major advantage of the IFI channel is that it continues to provide realtime communication in the face of several failures. We would argue that such failures are extremely unlikely, except in the event of major natural disasters, and that in that event, most applications will have to stop transmitting to allow emergency communications to occur. For the vast bulk of applications, the tolerance provided by the dispersity systems is sufficient. For mission critical applications, something like IFI channels would be appropriate, though it would have to run in more general network environments than the HARTS network.

SFI channels are more general than IFI channels, and may be used with all topologies. However, the service provided is tolerance to single faults, and their the realtime resource usage of three times the requirements of a non-fault-tolerant realtime channel is excessive. Again, the technique is cold, so that the resources may be used in the absence of faults by non-realtime data, but it reduces a little too much the realtime capacity of the system. Dispersity systems provide tolerance to single faults at a fraction of the overhead, and a far larger range of useful levels of fault tolerance.

In conclusion, we have shown that dispersity routing is a very general method for improving the failure and loss tolerance of realtime channels. It can provide a very large variety of services: from transparent tolerance to graceful degradation of service; from instantaneous recovery to recovery within one round trip for message exchange; and from tolerance to single restricted failures[4] to complete tolerance to $N - K$ faults. The cost of the system, in terms of the network bandwidth needed, depends on the level of redundancy provided, but is ameliorated by the effect of spreading the resource usage among multiple paths, and by the reduced external fragmentation of the link capacity. Most of the network support required is already provided by realtime protocols such as those of the Tenet Suite. The additional support required, limited to supplementary routing constraints, has been described in Section 6.6. The remaining mechanisms of the fault-tolerant systems can be implemented at or above the transport layer.

[4] Restricted to failure of the unshared links in an $(N, N-1, 2)$ system.