AN OVERVIEW OF A FAULT TOLERANT COMMUNICATION NETWORK

C. S. Vaidyanathan, David Callele, and Carl McCrosky

DSG Communications Inc.

P.O.Box 439, Saskatoon, S7K 3L6

Saskatchewan, Canada

ABSTRACT

An essential component of future space station communication systems is a highly reliable network which can handle high data throughputs. The network must be adaptable and reconfigurable to accommodate different applications. For routing messages in these networks a distributed mechanism is considered appropriate because of the inherent fault isolation characteristics that may be embedded in the design of such networks. This paper provides an overview of requirements to be considered in designing a fault tolerant communication network. The underlying concepts of packets, packet routing, and fault tolerance are introduced. An overview of the simulation of one such network performed by the research team at DSG Communications Inc. is given. Proposed areas for future investigations and other potential areas for further research are highlighted.

INTRODUCTION

Future space stations, such as FREEDOM[1] will require communication networks as part of their basic capabilities. A fundamental requirement for a data communications network used for space applications is that the network must provide high communications capacity to accommodate teleoperation, remote sensing, and distributed computing. The network must be highly adaptable; new instruments and experiments will be incorporated on board over the lifetime of a space station. The system must also be highly fault tolerant to prevent degradation and to protect vital functions in the face of circuit faults or physical damage.

There are two approaches to routing and load balancing in communications networks: centralized control and distributed control. Centralized control relies on a single system element to monitor traffic throughout the system. Routing and load balancing decisions are based on this system view. Distributed control makes routing and load balancing decisions at each element in the network.

Centralized control traditionally provides better load balancing than distributed control due to more detailed knowledge of overall system traffic. Distributed control systems typically have lower performance than centralized control systems because each system element only has knowledge of the traffic that passes through it.

However, centralized control is much more susceptible to failure than distributed control. If the central controller fails the entire network fails. If a network element in a suitable distributed control system fails only a small portion of the network fails. Ideally, the performance degradation is linearly proportional to the number of failed network elements.

This paper provides an overview of requirements to be considered in designing a fault tolerant communication network. The underlying concepts of packets, packet routing, and fault tolerance are introduced. Configuration of a high speed communication network suitable for fault tolerant communication is presented. An overview of the simulation of one such network performed by the research team at DSG Communications Inc. is given. Proposed areas for future investigations and other potential areas for further research are highlighted.

Interconnection Networks

Information to be passed from one device to another is grouped together into packages called packets. A packet can be thought of as an envelope and the information as the letter within. The network connecting these devices consists of a collection of routers wired together in a specific fashion. Each device (which may be both a sender and a receiver of information) is directly connected to one of the routers. When senders need to communicate they form their messages into packets and pass them to their router. The routers then systematically pass their packets to routers nearer to the receiver (similar to the way Canada Post delivers your mail - but much faster). When a packet arrives in the router connected to the packets receiver, the router passes the packet to the receiver.

The network itself is formed by wires connected between routers. There are 8 wires connected to every router in the network used in this investigation. These 8 wires provide many alternative paths from the sender to the receiver for each message. In the event of congestion (a traffic jam), or if wires or routers quit working, packets can be routed on alternative paths and still reach their destination.

Given the potential variation in load on a network the bandwidth of any edge may be exceeded at higher loads. The network load variation can cause messages to block i.e. the messages are prevented from being forwarded to the routers closer to their destinations. Blocking is a function of the network topology - any network with less than full connectivity (i.e. every node has a direct connection to every other node) will experience blocking as the communications load increases beyond a certain bound. Network designers attempt to use less than fully connected networks, such as the hyper cube, in an effort to reduce the cost of implementing the network. For example, the number of routers in a crossbar switch (a class of non blocking network) grows as N^2, where N is the number of nodes in the network.

The routing mechanism works best when the routers are connected to each other with many wires. Several alternative connection patterns are possible but one in particular, called a hyper cube, is particularly attractive. A hyper cube has N routers which result in a dramatic reduction in complexity as N grows large.

[1]Space Station Freedom is a 100-metre-long Manned Base to be constructed by 1999.

Hypercubes are generalizations of the cubes that we generally associate with the term. Several hypercubes are shown in Figure 1.. D is the dimension of each cube (D = 3 is the "normal" three-dimensional cube); N is the number of corners in the cubes (these are the routers in our networks); and W is the number of edges in the cube (these are the communications links in our networks). Hypercubes are built and extended (dimensionally) in a very straightforward manner[8].

Hypercube Networks

The hyper cube has long been of interest as an interconnection network for parallel computing systems using packet switched data. Communications network designers are drawn to its geometric regularity, susceptibility to mathematical analysis, and exceptionally simple, yet powerful routing algorithms.

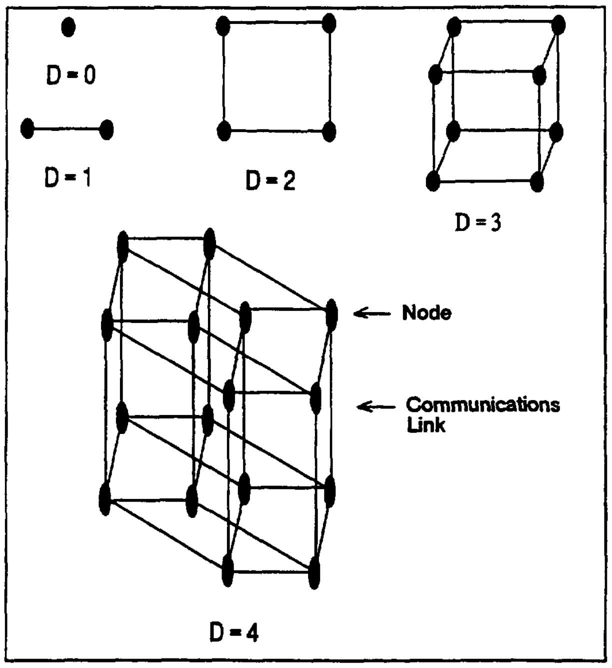

Figure 1. Low dimension hypercubes

Hypercubes are networks with 2^D nodes. Any two nodes are connected by a communications link if and only if the numeric identifiers for each node differ in only one bit, for nodes numbered from 0 to (2^D-1) in base 2 notation. For example, node 0101 and node 0100 are connected by a communications link along the zeroeth dimension and nodes 1111 and 0111 are connected by a communications link across the third dimension. D is referred to as the dimension of the hyper cube. D is also the fan-in and fan-out of each node and the diameter of the network (the maximum distance across the network). A hyper cube of dimension D has $D \cdot 2^{D-1}$edges (communications links) where nodes are connected via these edges. The average distance between nodes is D/2. As described thus far, hypercubes are interconnection networks with:

1) physical wires corresponding to each edge,
2) routers corresponding to each node, and
3) communication sources/sinks corresponding to each node.

DESIGN CONSIDERATIONS

Two routing algorithms have been developed and investigated by the DSG team. The algorithms permit the conceptual advantages of hypercubes to be exploited without limiting the designer to constructing a physically complete hyper cube. The network designer can incorporate as many nodes and edges as are required to provide the required levels of throughput, symmetry, and fault tolerance. Selected edges and nodes can easily be removed to reduce the system capacity if a complete hyper cube implementation is excessive.

Additional flexibility is gained by the ability to add new edges that are not in the hyper cube structure. Thus, edges that span two or more dimensions of the hyper cube can be added. These extra edges reduce the effective diameter of the network and increase throughput and fault tolerance.

Any network, up to a completely connected network, can therefore be imbedded in a hyper cube address space of arbitrary size. However, practical implementations are likely to limit the fan-in / fan-out of any particular node. Consequently, our approach does not limit the freedom of the packet-switched network designer in any way. Instead, the benefits of fault tolerant routing in a conceptual hyper cube are supported.

The routing algorithms can dynamically reconfigure to adapt as wires or routers fail. Detection of faults and reconfiguration are entirely local operations: no global reconfiguration is required. A network can continue correct operation until severed into unconnected segments. Even then, the unconnected segments of the network will continue correct operation for local traffic.

Most common network topologies, such as rings, stars, meshes, toroidal rings, trees can be directly mapped onto a hyper cube network. It has been shown that a hyper cube is a super set of those communications networks that we are most familiar with.

Packet Switching Versus Virtual Circuits

Communications networks are defined by the manner in which they deliver information. *Virtual circuits* guarantee delivery in the order in which it was transmitted. *Packet switched* networks guarantee delivery but make no guarantee as to the order in which information is received. A virtual circuit is an identified path connecting the Source (of the message) and the Sink (destination of the message) over which the sequence of transmission and reception is maintained. Virtual circuits are relatively expensive to set up and to reconfigure. The most common use of virtual circuits is voice calls over the public switched telephone network. In contrast, packet switched networks are relatively inexpensive to implement since the packets are simply "dumped" into the network and the network routers can direct them to their destinations without concern for priority or ordering. The most common use of packet switched networks is for wide area computer communications.

One concern in packet switching is the degree of out-of-orderness of the packets in a message upon delivery. When simulating a virtual circuit with a packet switch, the packets must be buffered at the last network element so that they can be delivered to the sink in correct order. The degree of out-of-orderness is a measure of how many packets may arrive out of order, and it governs the size of the buffer required. A virtual circuit, by definition, has an out-of-orderness of degree 0. An out-of-orderness of degree 3 in a packet switched network means that the network must be able to store up to 3 packets, while waiting for the arrival of the next packet, in order to present the packets to the sink in correct order.

All packet switching networks need to supply virtual circuit services and that packet ordering is typically a responsibility of higher levels of the network than those under investigation.

Two Hypercube Concepts

The hyper cube is a blocking packet switching network where the number of message links between a given router and the rest of the network varies as the logarithm (to the base 2) of the number of nodes in the network. Maximum (ideal) use of the network occurs when every router has D packets to move and all packets are forwarded with no wire assignment conflicts. Thus, every wire in the network is used to move a packet closer to its destination. No greater throughput is possible - all wires are always full moving packets toward their destination along optimal paths.

Average Packet Travel Distance is D/2 hops

A hyper cube of degree D has $D \cdot 2^{D-1}$ bi-directional wires or edges and 2^D routers. In the absence of conflicts, $D \cdot 2^D$ packets can be moved in one Packet Exchange Cycle (PEC). The average distance packets in a random message load must travel is $\frac{D}{2}$.

Let D = the dimension of the hypercube.
The possible travel distances are $0, 1, 2, \ldots, D$.
Each travel distance has a probability equal to:

$$\text{Number of Routers at that Distance} = \frac{2^D}{\text{Total number of Routers}}.$$

The Expected value of the travel distance

$$(\text{with router 0 as the reference point}) \text{ is } = \sum_{\text{Router } 0}^{\text{Router } (2^D-1)} \text{distance to } R_i.$$

where:
distance to R = number of bits set (= 1) on the binary representation of R
R = routing code in binary notation.
Assuming a stationary random process, the average travel distance is given by:

$$\sum_{i=0}^{i=D} \frac{i}{D+1} = \sum_{i=0}^{i=D} i$$
$$= \langle \frac{1}{D+1} \rangle \cdot \langle D \rangle \cdot \langle \frac{D+1}{2} \rangle.$$
$$= \frac{D}{2}.$$

For example, with D = 3, the distance distributions are:
1 node at distance 0,
3 nodes at distance 1,
3 nodes at distance 2, and
1 node at distance 3.
Thus, the average travel distance for D = 3 is:

$$\frac{\langle 1 \cdot 0 \rangle + \langle 3 \cdot 1 \rangle + \langle 3 \cdot 2 \rangle + \langle 1 \cdot 3 \rangle}{2} = \frac{3}{2}.$$
$$= 1.5 \text{ Q.E.D.}$$

Sustained Maximum Injection Rate

Every router in a hyper cube manages a message queue. Incoming messages received from neighboring routers are placed in this queue. Messages to be injected into the network from a source are also placed in this queue if space is available. The attempt to inject new messages is rejected if no space is available. The inherent load limiting can be analytically determined as follows:

Let D = the dimension of the hyper cube and the number of edges connected to any given router. A router has a capacity of D packet-steps per PEC. That is, a router can move up to D packets (the number of edges connected to it) one step (to its nearest neighbors) per PEC. There are 2^D routers in the hyper cube, therefore $D \cdot 2^D$ forward steps may be accomplished in every PEC. The average distance per packet was shown (in the previous section) to be $\frac{D}{2}$. Therefore, we can expect on average, $\langle D \cdot 2^D \rangle \frac{D}{2}$, or $2 \cdot 2^D$ packets to be delivered per PEC. Since there are 2^D routers, this represents a value of 2 delivered packets per router per PEC. Since the number of packets injected is limited by the number of packets that are delivered, on average, 2 packets can be injected and delivered by each router every packet exchange cycle.

OBJECTIVES OF THE SIMULATION STUDY

The research team carried out computer simulations of a high capacity fault tolerant communications networks proposed for use on the Space Station FREEDOM. The proposed network is based on routers connected in a mathematically regular (easily defined and repeated throughout) fashion. The routers implement adaptive packet switching routing algorithms.

In brief, the objectives of this technical study were to:

1) determine, through computer simulation, the performance characteristics of the network routing algorithms developed by McCrosky (SCS) and Callele (the Friendly algorithm).

2) Compare the performance of the proposed algorithm with those methods commonly in use: in particular, dimensional routing (the Naive algorithm), and worm hole routing (Dally's Algorithm);

3) determine the "crossover" points for the network loads that would justify choosing between the four algorithms for the given loading conditions;

3) determine the fault tolerance and survivability of the proposed algorithms to a number of fault conditions (loss of nodes, edges, and combinations thereof).

The performance of each algorithm for distributed control of routing, load balancing, and fault tolerance was investigated via detailed computer simulation. Performance was measured under conditions ranging from 20% to 85% of theoretic maximum load for the condition of no faults in the network. The performance of the SCS and Friendly algorithms was further investigated for a variety of fault conditions ranging from 1% to 8% of the total number of links in the network.

Network Model Used in the Simulations

The performance of four routing algorithms is examined by simulation of network traffic. The simulator is designed as a network of communicating objects, each object representing a node in the network. The input data sets for simulation are obtained from mathematical distributions based on the following considerations:

The network load can be modeled as being composed only of full packets or of partial packets as well. That is, packets are only introduced into the network if there is enough information available to create a full packet (partial packets are never injected) or packets are injected into the network on a time division basis (if there is any information to be transmitted during this time slot, inject a packet containing this information). The full packet network model conserves network bandwidth until maximum efficiency (in terms of actual information delivered each PEC) is reached since no transmission capacity is wasted in transmitting partially full packets. The full packet network model does not account for the injection delay as messages are accumulated to form full packets.

The simulator, as implemented, uses the full packet model - it does not differentiate between full packets or partially full packets. Instead, the simulator uses the concept of *streams* to create a *frequency of packet injection* rather than an actual messaging load. The frequency of packet injection can be used to model full or partial packet injection. A partial packet is a packet that uses an entire packet exchange time slot without transmitting the maximum amount of data during that period. Since a partially full packet uses as much network communications bandwidth as a full packet, there is no requirement to differentiate between the two mechanisms. Partial packets can be represented as full packets, and if desired, the efficiency of communications bandwidth utilization can be calculated by correcting for the 'wasted' space inside of partially full packets.

The routing algorithms investigated consist of mechanisms used to achieve a balance between queuing delay, latency, and throughput while routing the message packets. Four algorithms were selected for investigation. They are:

1) Naive routing algorithm,

2) Dally's algorithm,

3) McCrosky's Saturated Constant Shuffle (SCS) algorithm, and

4) Callele's Friendly algorithm .

OVERVIEW OF THE SIMULATION PROCESS

For simulating the four routing algorithms, two input and three output files were used. These are:

Input:

1) the tuple source file,
2) the fault file,

Output:

3) the packet output file,
4) the final channel data output file, and
5) the final router data output file.

The relationship between these files and a block diagram of the simulation process is presented in Figure 2..

Generation of Random Packet Loads

The simulation load is modeled as a set of unidirectional streams of packets from sources to sinks. Each stream is described by a source, a sink, a start time in PECs, a duration in PECs, a rate in packets per PEC, and a stream identification number.

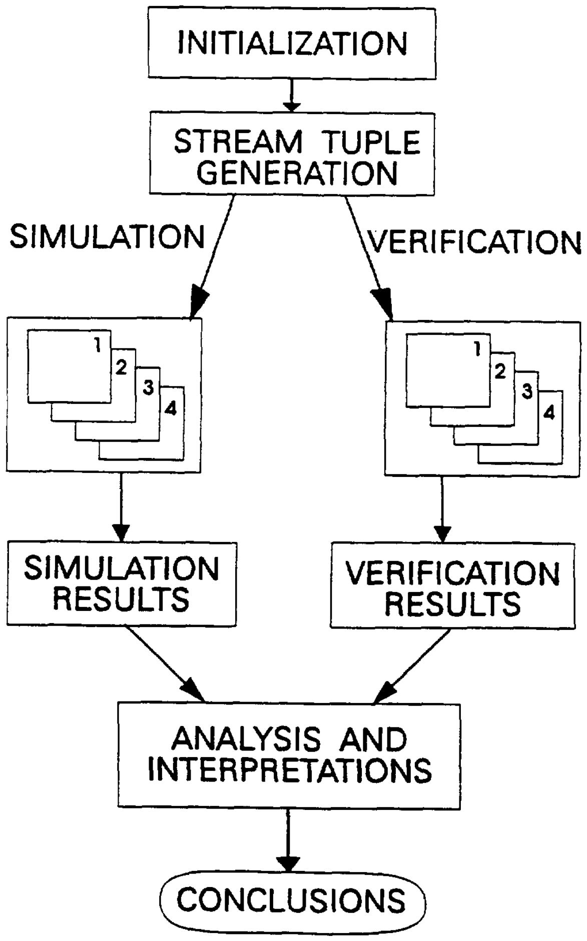

Figure 2. Outline of the simulation process.

The simulation assumes bi-directional links that can move one packet in each direction during each PEC. Consequently, each hypercube's capacity per PEC to move messages is the number of routers, N, times the number of dimensions, D, giving $N \cdot D$. With random selection of source and sink, the average packet travels D/2 edges. Thus the $N \cdot D$ moves per PEC divided by the average distance per packet, $\frac{D}{2}$, gives the average number of packets moved from source to sink per PEC. This figure, $\frac{N \cdot D}{\langle \frac{D}{2} \rangle} = 2 \cdot N$, is the ideal number of packets that can be delivered per PEC. For $D = 8$, we ideally expect 512 packets to be delivered per PEC. Each router can ideally source and/or sink a total of four packets per PEC. The simulation is designed for loads where each router can source two packets and sink two packets per PEC.

The overall ideal capacity of the hyper cube to deliver packets during the simulation period can be represented as a table of ideal routing capacity versus PECs, as shown in Figure 3..

PERFORMANCE RESULTS

Determination of Simulation Run Length

In any simulation study it is necessary to determine the run length of the simulation. If the run is too short random fluctuations may dominate, potentially leading to unreliable results. If the run is too long, valuable computing resources may be needlessly wasted. As well, the 'tail' of the random distributions can also influence the results. Unfortunately, run length determination for non-trivial simulations is more of an art than a science.

An artifact in the generation of streams in the reported load model makes it clear that the initial cycles of the simulation should not sampled. Stream start times and duration's were randomly sampled and at any time past the maximum duration of a stream from the start of the simulation the work load will be uniform. However, from the beginning of the simulation to the end of the maximum duration of any individual stream, a lower than average load density is expected. This is a consequence of using a continuous distribution sampled at a discrete time. Until all streams obtained from this initial portion of the sampled distribution have been eliminated, the load will not exhibit the desired characteristics. Consequently, data obtained during this startup period was ignored. Only after the presented load had reached the desired level was data collected.

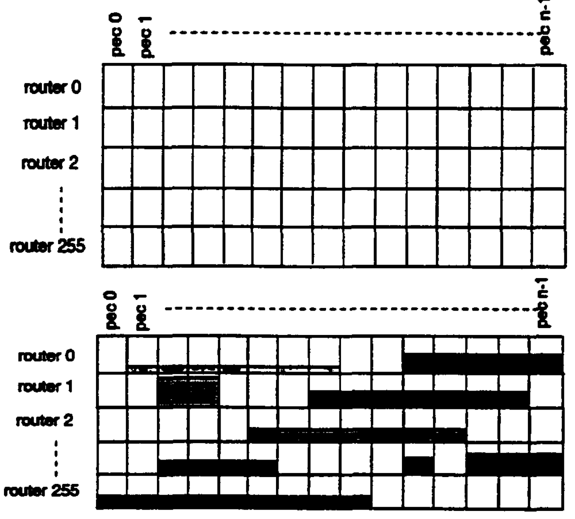

Figure 3. Layout of the allocation table for random load generation.

A set of experiments were conducted to determine the period required to obtain results that were statistically 'stable'. Simulations with progressively longer sampling periods (from 500 to 3000 PECs in increments of 500 PECs) were run until it was determined that few fluctuations in the critical statistics were occurring. A run length longer than required (2000 PECs) was chosen as the standard run length for all subsequent simulations.

The results of the run length tests for Friendly and SCS are given in Table 1.

Execution of the Simulations and Collection of the Results

This section reports the results of the simulation studies. The principal objective was to characterize the performance of each of the four algorithms under varying load and fault conditions. The performance criteria used to compare the algorithms is briefly reviewed.

Table 1. Determination of Simulation Run Length.

Run Length, PECs	Average Latency SCS algorithm PECs	Average Latency Friendly algorithm PECs
500	4.31	4.27
1000	4.32	4.29
1500	4.32	4.29
2000	4.33	4.29
2500	4.31	4.27
3000	4.32	4.28

Each algorithm was coded as a separate specialization of a set of basic packet routing objects in an object-oriented programming language (C++). Each simulator used a stream file (describing the load) and a fault file (describing the link fault conditions) and produced a packet file containing statistics on each delivered packet, a link file containing aggregate statistics on the use of each link, and a router file containing aggregate statistics on the use of each router.

Statistics were collected only after an initial period to accommodate the stabilization of presented load. Router and link statistics were collected only from PEC 1000 to PEC 3000. Packet statistics were collected for all packets from the start of simulation to the end of simulation allowing verification of the latency equations for all packets, not just a subset

The four algorithms were tested under loads ranging from 20% to 85% of theoretic maximum capacity and 0% fault conditions. The total number of packets to be delivered varied as the load increased. Precise values for each load are given in Table 2..

Figure 4. presents the proportion of presented load that was actually delivered as the communications load rose from 20% to 85% with no fault conditions.

Friendly and SCS successfully deliver 100% of their packets under all presented loads up to 80%. Neither Dally nor Naive deliver all of their load even under the lightest presented load. The behavior of Dally and Naive degrade seriously as presented load rises.

It was surprising to note that Dally and Naive performed so poorly on this first test. The reason is believed to be that the Dally's algorithm is intended for a load model that consists only of relatively long communications bursts at the maximum transmission rate. This traffic model is typical of a collection of computers occasionally exchanging files. The file transmissions are intermittent, but relatively long and at high speed. A routing algorithm that can establish a dedicated pipe for such traffic will be superior. However, the communications load anticipated for the Space Station FREEDOM (and for terrestrial applications of hyper cube routing structures) is more varied. We expect a wide range of transmission rates - from a minute fraction of a packet per PEC to a complete packet every PEC.

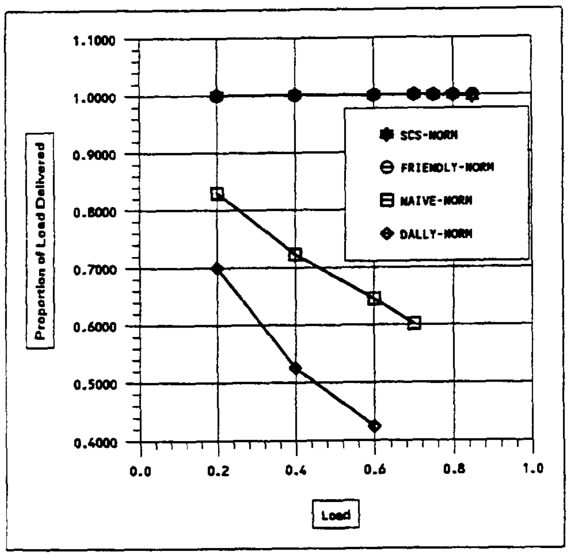

Figure 4. Proportion of Load Delivered

Table 2. Total Number of Packets for Varying Loads.

LOAD	STREAMS	TOTAL PACKETS	PACKETS PER ROUTER	PAC KETS PER PEC
20%	786	205,244	802	0.401
40%	1,435	410,470	1,603	0.802
50%	1,817	512,945	2,004	1.002
60%	2,206	614,779	2,402	1.201
65%	2,493	665,076	2,597	1.299
70%	2,856	715,953	2,797	1.399
75%	3,380	767,152	2,997	1.499
80%	4,126	818,662	3,198	1.599
85%	5,199	870,174	3,399	1.700
90%	6,796	921,381	3,599	1.800

Dally's response to these loads is to open a dedicated channel for each stream without regard to the transmission rate. Consequently many channels (worm holes) waste large amounts of bandwidth idly waiting for the next packet to come down the channel. In contrast to SCS and Friendly, Dally reserves the channel for the exclusive use of the conversation causing all other conversations that require this channel to block.

Dally's algorithm, although successful in the specialized load conditions of multiprocessor systems exchanging files and memory images, is simply inappropriate for the more general load conditions anticipated for FREEDOM and other terrestrial applications.

The Naive algorithm also performed very poorly in delivering the presented load. Unlike Dally (but like SCS and Friendly) Naive does not dedicate channels but moves packets in a store-and-forward manner as available links permit. Why does Naive perform so much worse than SCS and Friendly? Naive is not an adaptive algorithm. There is only one possible path through the cube for each packet and the lowest numbered available dimension is always traversed first. Therefore, the hyper cube is completely unable to respond to the varying dynamic load presented to it. Naive can not route around congested areas. This failure causes even further congestion. The phenomena is much like 'hot spots' in Banyan Tree networks. Any tendency to congestion compounds itself and spreads. The non-adaptive nature of Naive not only reduces performance but prevents Naive from being fault-tolerant since there is no capability to route around congestion or faults.

Latency Versus Load

Given that SCS and Friendly manage to deliver all their presented load under no fault conditions, how quickly are those packets delivered? Figure 5. illustrates the average latency for all delivered packets versus the presented load for Friendly and SCS.

The ideal average latency for a D = 8 hyper cube with random selection of source and sink is 4. Figure 5. includes this ideal latency line for comparison. Friendly and SCS exhibit a slow increase in latency from near ideal at 20% load to less than six at 70% and 80% respectively.

Friendly is superior to SCS at all loads. Friendly remains useful (latency < 10) until 85%, whereas SCS remains useful only to 80%.

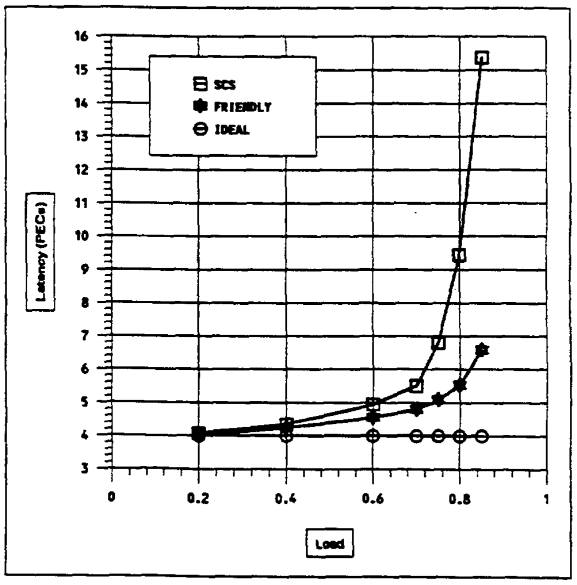

Figure 5. Latency versus load curves for Friendly and SCS.

The goal of each algorithm is to maximize the throughput of each router (as measured in packets per PEC). Each algorithm is considered useful under increasing load only as long as the algorithm continues to deliver all the packets presented by the load.

Simultaneously, each algorithm must attempt to minimize the delivery latency of packets throughout the network. Under conditions of light load, all algorithms should produce near ideal latencies. The message loads used were designed to present a load with an ideal latency of 4 PECs in an 8 dimension hyper cube. It is expected that, as the communications load and fault levels increase, latency will increase.

CONCLUSIONS

The novel components contributed by McCrosky and Callele are the design of new packet routing algorithms that control the actions of the individual routers. These two new algorithms (SCS and Friendly) avoid several pitfalls of previous work, and provide very high performance.

Because there are many routers and many wires connecting them, these networks have very large message handling capabilities. This characteristic is particularly appropriate for demanding applications where many packets must be moved quickly.

These networks are also able to deal very effectively with the failure of wires and routers in the networks both because of the number of routers and wires as well as certain aspects of the algorithms. The surviving network is still capable of successfully delivering messages if many of the wires or routers fail. The performance of these networks degrades gently as these faults occur. This is a highly attractive feature for any application where it is very difficult to perform repairs (such as the Space Station FREEDOM) or where a disaster could occur if the network breaks down (such as nuclear power plants).

The SCS and Friendly algorithms both provide many advantages over traditional approaches for control of packet switched communications networks. Both algorithms operate in a completely distributed manner and eliminate all overhead associated with centralized control systems. Their distributed nature also makes them much more fault tolerant than centralized mechanisms. Both algorithms are also simple enough to allow direct implementation in hardware as state machines. A direct hardware implementation allows systems to be created very inexpensively and with higher levels of performance than software based systems. Both algorithms are robust with respect to technological change and can be implemented in the most suitable technology for a given application.

The performance results obtained from the simulation are very promising. Both the Naive algorithm and Dally's algorithm performed so poorly on fault free simulations (as compared to the other algorithms) that they were dismissed from further consideration. The SCS and Friendly algorithms both deliver highly acceptable performance for loads up to 80% of the theoretic limit. The Friendly algorithm has marginally better performance than SCS until the load exceeds approximately 70%. As the load increases beyond 70%, or under more extreme fault conditions, the performance of SCS degrades more rapidly than Friendly. However, Friendly's performance margin comes at a price. The hardware required to implement SCS is expected to be considerably less complex than that required by the Friendly algorithm.

Therefore, for loads of less than 70% or where the level of faults expected is not extreme, the best choice is SCS. If loads in excess of 70% of theoretic maximum are expected or a high level of fault tolerance is required, then the Friendly algorithm is the best choice.

In summary we have established that two viable routing algorithms for highly-fault tolerant, adaptive packet routing exist. The performance and fault-tolerance of these algorithms is dramatically better than known alternatives under a wide range of load conditions. If one keeps in mind that these are fully distributed algorithms, with NO use of communications bandwidth for network management overhead, then both SCS and Friendly deliver truly incredible performance with very punishing loads. Both SCS and Friendly appear to be viable control algorithms for a wide variety of applications of hyper cube-based packet routing algorithms.

ACKNOWLEDGEMENT

The authors thank the Government of Canda and the Government of Saskatchewan, via the Canadian Space Agency and the Saskatchewan Research Council, for their support of this research under the Strategic Technologies in Automation and Robotics (STEAR) program.

REFERENCES

[1] Dally, William J., "*Performance Analysis for k-ary n-cube Interconnection Networks,*" IEEE Transactions on Computers, June 1990, pp. 775-85.

[2] Geoffrey, Gordon., "*System Simulation*", second edition, Prentice_Hall, Inc., pp. 112-172, 1978.

[3] Brooks, E.D. , "*The Shared Memory Hyper cube*", UCRL-92479 Preprint, Lawrence Livermore National Laboratory, March 1985.

[4] Valiant, L. G. and Brebner, G. J., "*Universal Schemes for Parallel Communication*", Proceedings of the 13th Annual ACM Symposium on Theory of Computing, Milwaukee, May 11-13, 1981, pp: 263-277.

[5] McCrosky, C. and Reid, I., "*Nearly Optimal Packet Routing in Hyper cube-Connected MIMD's*", Technical Report 86-4, Department of Computational Science, University of Saskatchewan, March 1986.