1. Introduction — 3
1.1 Objectives of RTFC — 3
2. Segment Topology — 3
2.1 Why rings? — 5
2.2 Why stars? — 5
3. Rostering Overview — 6
4. Hub Port and Node Bypass Rules — 7
5. Rostering Requirements — 7
6. The Rostering Process — 9
6.1 Rostering in a Hubless Segment — 9
6.2 Rostering in a Segment Having Hubs — 11
6.2.1 The Population Explosion is Contained — 13
6.3 Exit from Rostering Mode — 16
7. Entry and Exit of Nodes — 16
7.1 Detailed Node Insertion and Exit Sequences — 17
7.1.1 Insertion — 17
7.1.2 Exit — 18
8. Performance of RTFC segments — 18
8.1 RTFC is Random-Access, not Sequential-Access — 18
8.2 Ring Tour Time — 20
8.2.1 Node Delay — 20
8.2.2 Link Delay — 21
8.2.3 Hub Delay — 22
8.3 Theoretical Throughput of RTFC Variants — 24
8.3.1 Theoretical Throughput using Large Block-Mode Data Micropackets — 26
8.4 Segment Transmission Error Model — 27
8.4.1 Micropacket Loss — 27
8.4.2 Wrongful acceptance of damaged micropackets — 28
8.4.3 Control-micropacket spoofing by damaged micropackets — 28
8.4.4 Message Damage — 29
8.4.5 Effect of Cut-Through Routing — 29
8.4.6 Forward Error Control Considered — 30
8.5 RTFC Flow Control — 30
8.5.1 Flow Control Objectives — 30
8.5.2 Flow Control Algorithm — 31
8.5.3 Flow control with Cut-Through Routers and Early Packet Discard — 32
9. Damage Tolerance of RTFC Segments Quantified — 33
9.1 How Many Ringlets can a Segment Have? — 34
10. Fault Isolation and Resilience Features of RTFC Segments — 35
10.1 Detection and isolation of faults in the RTFC hardware itself — 35
10.1.1 The first line of defense — built-in power-up diagnostics — 35
10.1.2 Handling Chattering Inputs — 35
10.1.3 Verification of correct configuration — 37

10.1.4 Recourses against misbehaving nodes ____ 37
10.1.5 Detection and isolation of faults in data transmission ____ 37
10.1.6 Maintenance Diagnostics Support ____ 38

10.2 Detection and isolation of faults in the host software ____ 38
10.2.1 Watchdog Timer ____ 38
10.2.2 Protection of network cache from confused host software ____ 39
10.2.3 Protection of network cache from confused other nodes ____ 39
10.2.4 Isolation by segmentation ____ 39
10.2.5 Listen-only mode ____ 39
10.2.6 Segment loop-around tests ____ 39

11. Notes on the RTFC Packet-Handling Flowchart ____ 39

11.1 Per-Node and Roster-Packet Data Structures ____ 40
11.1.1 Roster-packet data ____ 40
11.1.2 Per-Node Data ____ 40

11.2 RTFC Packet Handling Flowchart ____ 41
11.2.1 Roster Mode ____ 41
11.2.2 Transition Mode ____ 41
11.2.3 Data-Transfer Mode ____ 42

11.3 Rostering Process Flowchart ____ 42

11.4 Roster Packet Processing Flowchart ____ 43

11.5 Data/Interrupt Packet Handling Flowchart ____ 44

12. The annotated original requirements for RTFC ____ 45

13. References ____ 55

1. Introduction

The purpose of this document (RTFC-2001 “RTFC Principles of Operation”) is to explain how RTFC (Real-Time Fiber Communications) handles data-transfer mode traffic, and also how RTFC achieves damage and fault tolerance. The intended audience of the present document is engineers implementing RTFC hardware. As data-transfer mode is fairly simple, the bulk of the present document addresses how damage and fault tolerance are achieved. The general approach is to first explain the physical setup of an RTFC segment (“ring”) and its components, to describe the requirements on the rostering algorithm and where those requirements came from, and then to follow the rostering process step by step. A few sections describe data-transfer mode. Also described are some variations on data-transfer mode, such as unicast (point to point) addressing, early packet purging, and cut-through routing.

This document is intended to be more a tutorial than a standard; that will come later. Specifically, the base standard for on-glass micropacket formats appears in RTFC-2002 “RTFC Micropacket Formats”, and the base standard for roster mode appears in RTFC-2003 “RTFC Rostering Algorithm”.

1.1 Objectives of RTFC

The driving requirement was damage tolerance; existing gigabit reflective memory products were already fast enough and predictable enough. The hardware had to support host software (and middleware), which implies among other things that causal order of updates must be absolutely preserved, and that overlapping updates of different areas of memory can not interfere with each other, or any other area. Specifically, the units of atomicity in RTFC shall be the naturally-aligned 32-bit longword, the naturally-aligned 16-bit word, and the 8-bit byte.

Some reflective memory offerings do not preserve causal order under all conditions. RTFC preserves causal order within two classes: those updates that tour the ring visiting all nodes, and those (unicast) updates that cut through the hub. Within each class, causal order is preserved. Between classes, it is not and can not be preserved, as cut-through traffic will by definition always outrun non-cut-through traffic.

2. Segment Topology

A “network” consists of one or more segments interconnected by bridge nodes. As the unit of hardware design is the segment, we will focus almost exclusively on the segment in the present document. A “segment” is a collection of interconnected nodes, links, and hubs. Hubs bypass nodes having broken node-to-hub and/or hub-to-node links. In segments with hubs, links connect nodes to hubs, and hubs to nodes, but do not generally connect hubs to hubs or nodes to nodes. In segments without hubs, links connect nodes to nodes, but damage tolerance is largely lost.

Some hubs allow direct hub to hub links, resulting in what is in effect one large logical hub, but with some tolerance for the loss of either hub or the link between them. The policy of hub ports connected to nodes is to always transmit a copy of all traffic to the connected node, but to bypass that node if no modulated light is received from the connected node. This is called the “answer darkness with light” policy. Nodes, by contrast, implement the “answer darkness with darkness” policy. If two hub ports are connected to each other, a broken inter-hub fiber will result in one closed ring and one open ring, rather than the desired two closed (but isolated) rings, if the hub ports use their normal answer darkness with light policy. Only closed rings work. This problem is solved by enabling (or emulating) Fibre-Channel [FC-PH] Open Fiber Control (OFC) only on the two hub ports that are connected together. OFC will disable both transmitters, and probe periodically for restoration of the link. On restoration, both transmitters will come on, and the two hubs will un-bypass the hub ports. This kind of distributed hub can be used to implement one-ring segments with a limited but still useful degree of damage tolerance.

There can be from one to four parallel “rings” in a given RTFC network segment. Each such segment is logically a ring, but physically a star. The center of each ring’s physical star is a hub (aka, concentrator). For each ring there is exactly one hub, and for each hub there is exactly one ring. The rings, which must precisely parallel each other in every detail except link length, are joined only at the nodes. Nodes are identified by a network-unique 8-bit “NodeID” defined by hardware jumpers either on or connected to each RTFC node circuit card from the VMEbus "P2" connector. The “wrap/purge node” (also known as the “scrub node”) has the numerically lowest NodeID of the currently non-bypassed nodes, and therefore by definition the wrap node’s nearest upstream neighbor has a higher NodeID. Hubs, which are (for the purposes of the present discussion) invisible to nodes, do not possess NodeIDs (but may possess PortIDs). A given node can connect to four or fewer hubs, while a given hub can connect to a given node exactly once. Hubs are never directly connected to each other (except as noted above), and are in addition generally widely dispersed physically, for damage and power-failure tolerance. The hubs and nodes are bound into these rings (stars) by multiple unidirectional point-to-point gigabit fiber optic links, used to carry information from node to node via the hubs.

In operation, the segment chooses a single composite logical ring to connect the nodes together. This single ring is composed of various more-or-less randomly chosen pieces of the up to four individual rings, joined only at the currently-available nodes.

Hubs can be either “dumb” or “smart”. Hubs are designed to be dumb for a number of reasons: cost, circuit card real estate, latency, the requirement that a segment be able to be configured to operate without a hub (for use in labs), and for compatibility with some prior non damage tolerant reflective memory products. Hubs are designed to be “smart” to support added functionality, such as diagnostic packets and cut-through routing, if the node hardware supports such functionality. Smart hubs must therefore be able to act dumb, for compatibility. In the rest of the present document, hubs are assumed to be dumb, unless specifically stated otherwise.

As will be discussed later, some smart hubs can assign “PortIDs” to nodes capable of accepting such assignment. These PortIDs act as NodeIDs for the purposes of the present document, so only NodeIDs will be discussed.

It should be noted that in the present document and in conversation, we use the term “ring” for two things, the up to four physical rings (stars, physically), and also for the one composite (logical) ring currently being used for communication, differentiating only by context.

In the present document, “R” and “N” are always the number of rings and nodes respectively, of the segment being discussed. The range of R is one to four, inclusive. The range of N is zero (the empty ring) to 256, inclusive. An RTFC network is composed of a number of RTFC segments joined by a number of bridging nodes. The number of segments per network is determined by the design of the middleware. Only segments can roster, so the present discussion focuses on segments. The segments of a network roster independently of one another, and without mutual interference.

RTFC NodeIDs are physical station addresses and as such are the logical equivalent of the 48-bit Media Access Controller (MAC) physical addresses defined in Ethernet (IEEE Std 802.3), and in FDDI.

2.1 Why rings?

We have four strong reasons to choose a logical ring topology, as discussed in the following four paragraphs.

First, the system requirement of gigabit data transmission in a network of 300-meter (to 30-kilometer) radius requires the use of unidirectional fiber-optic links, as no other widely available commercial communications technology comes close. Point-to-point links between N boxes requires N^2 links, which is impractical and unnecessary. Although a typical value for N is thirty or forty, N can be as high as 256. Busses are impractical to implement with the required gigabit optical technology, especially in the required physical diameters, and are slow by comparison because they are a single media shared by time multiplexing. Busses also lack damage tolerance. A standard solution to these issues is a ring topology.

Second, rings behave very well under heavy traffic, going into gradual saturation, rather than collapsing, unlike busses.

Third, use of a ring topology allows the rostering time to scale linearly with segment size, measured both in nodes and in meters of fiber. Discovering the best path through an unknown network requires exhaustive exploration of that network. The network is always unknown because one cannot know in advance what will and will not survive a damage event, and rostering must always run just after such a damage event. Networks of unconstrained topology can require considerable time to explore all paths fully, making rapid automatic reconfiguration of such networks computationally intractable if the network is of any practical size. By contrast, rings can be fully explored in linear time using an algorithm simple enough to implement directly in hardware.

Fourth, use of a ring topology allows solutions to the flow control problem to become tractable, even simple, as, in a suitably-designed ring, node-local observations suffice to determine the global flow state of the ring. (Unicast and cut-through traffic undermine this perfect observability to some extent.)

2.2 Why stars?

The logical ring is physically arranged as a star, with an electronic hub in the center and the nodes as the points of the star. Why this topology, rather than a ring with optical bypass relays in or near each node? There are a number of reasons:

First, the only practical way to passively bypass an unpowered node is by use of normally-bypassed optical relays, which take about ten milliseconds to switch, compared to the sub-millisecond switch time of an electronic hub port.

Second, optical bypass relays are sensitive to mechanical shock. After a weapon strike, all the bypass relays are likely to chatter for several seconds, disrupting the segment until the vibrations in the ship's hull die down.

Third, the bypass relays have to be remotely mounted from the nodes they bypass, to avoid destruction in the same damage event that destroys the associated node. These remotely-mounted relays are then points of vulnerability all by themselves. Survivability analysis shows that it’s better to collect these relays into a few small and hard to hit citadels.

Fourth, experience has shown that ordinary maintenance is made quite difficult when components are scattered about or located in dark and inaccessible places.

3. Rostering Overview

Rostering, the process by which the currently-available components of an RTFC network segment organize themselves into a working ring, is invoked whenever the segment’s topology changes in any way, by either the loss or the gain of a hub, a node, or a link, as detected by the link receiver hardware in the hubs and nodes. The rostering process assumes that the segment topology will remain constant for the duration of rostering; if any violation of this assumption is detected, the entire rostering process is simply repeated.

Changes in segment topology are detected by the optical receivers in the hubs and nodes. Whenever the hardware in an optical receiver reports either the loss or the acquisition of both adequate light and FC-PH code longword phaselock, a “glitch” is declared. Loss or acquisition of phaselock depends in turn on loss or appearance of modulated light, but with an acquisition delay. If the glitch was detected by a node, that node enters rostering mode after a very short delay. If the glitch was detected by a hub (any kind), it immediately issues (only to the next non-bypassed node in sequence) a PC-PH “K30.7” 10-bit symbol (defined in the Fibre Channel hardware), which symbol, upon receipt, causes this next node to enter rostering mode, also after a very short delay. All enabled receivers, active or not, listen for and report glitches, so that the arrival of a new component can always be detected and responded to. Receipt (in data-transfer mode) of a micropacket having bad parity is not reported as a glitch, and does not provoke rostering, because parity errors do not generally signify that the segment topology has changed.

Rostering explores the segment and discovers the surviving nodes, hubs, and links by flooding the segment with roster packets. Except to ensure that the considerable physical capacity of the segment is not exceeded, no attempt is made to minimize the number of roster packets exchanged. This stands in contrast to the vast majority of distributed fault tolerant systems, where every effort is made to minimize the number of messages that must be exchanged.

All data in transit at the moment rostering begins is summarily discarded, to clear the way for rostering, and so must be re-sent by the host software (middleware) after rostering is complete. This simple policy suffices because there is no efficient way for the middleware to determine which data was and was not received everywhere, so the middleware must re-send all in-transit data anyway.

4. Hub Port and Node Bypass Rules

Hubs will bypass nodes having failed node-to-hub and/or hub-to-node links. Link failure is declared when a receiver detects loss of optical “carrier” (modulated light). All links are monitored, active and inactive alike. Bypassed nodes continue to get a copy of passing traffic, unless the hub-to-node link is broken. This is done to allow the node’s receivers to acquire phaselock in advance, thus speeding healing of subsequent faults, and reducing the duration of disruptions to ring traffic caused by the entry of a node into the ring. Acquisition of full FC-PH code and longword phaselock takes up to one millisecond, controlled by the physics of the phase-lock loops in the FC-PH integrated circuits used to implement the receivers [FC-PH].

Hub ports shall bypass nodes from which no modulated light has been received for more than a few microseconds. This covers failures in the node-to-hub link.

To prevent broken (open) rings, the hub port must also bypass nodes which receive no modulated light from that associated hub port, to cover failures in the hub-to-node link. However, because the optical links are strictly unidirectional, the hub has no direct way to observe this, so the next set of rules is needed.

Answer darkness with darkness: If a node receives no light from any of its inputs, that node shall stop sending light on all outputs, thus forcing the associated hub ports to bypass the node. If a node receives light from only one input, it shall disable one of the transmitters that do not correspond to this one surviving input, to ensure that we don't disconnect the node from its last surviving cable while also ensuring that at least one hub bypasses a node with most of its input fibers broken.

Hub ports shall continue to send a copy of passing traffic to bypassed nodes.

Hub ports must generate Fibre-Channel K30.7 (10-bit) symbols, while nodes are forbidden to pass K30.7 symbols through, so that a K30.7 will automatically be transmitted to the next unbypassed node, and no further. All K30.7 symbols are alike, so it is impossible to tell where one came from.

K30.7 symbols must be passed from hub card to hub card within a hub, emerging at the first non-bypassed hub port, to support fault configurations involving multiple bypassed nodes and also cases where the next unbypassed node is on a different hub card than the card that detected the glitch.

Although there is no requirement to do so, a hub port designed to pass K30.7 symbols through will allow such hub port cards to be used as repeaters, able to double the link range limit. This would also allow hubs to be connected together while retaining the full damage-tolerance capability. If repeater capability is not desired, the hub port optical inputs are required only to be able to generate K30.7 symbols, but not to receive them, which somewhat simplifies the hub port hardware.

5. Rostering Requirements

The intent of rostering is to cobble together the best logical ring one can from the surviving nodes, links, and hubs in the segment, where “best” is defined as the ring containing the maximum number of nodes. A secondary intent is to prevent, to the extent possible, the formation of isolated subnets (“ringlets”), even at the expense of leaving some nodes isolated. In practice, it may not be possible to utilize all available assets in the segment, so the rostering process is designed to include the maximum number of nodes possible given the specific fault pattern. Avoidance of ringlets (to the extent possible) is intrinsic to the basic choices made in the design of RTFC, so there is no identifiable place where multiple ringlets are explicitly handled and prevented.

The only point of control available to us is the setting of each node’s R-way input-source selector. There are five possible choices: no input, input 1, input 2, input 3, or input 4. If there is a tie between two or more inputs, the lowest-numbered input is generally chosen. The data from unselected inputs is ignored during normal data-handling operation. Unselected inputs continue to be monitored for glitches. Nodes send data out identically on all R output ports.

RTFC is designed to survive considerable damage, not just random faults occurring rarely. Fault tolerance and damage tolerance are not the same thing. Faults (failures) occur relatively infrequently, and are independent in both time and place. For instance, at high temperatures, the average failure rate may become quite high, but these failures are still random and uncorrelated. On the other hand, damage causes multiple things to fail simultaneously and all in one place. Power failures can mimic damage, in that whole sections of a segment can simultaneously vanish (and later reappear). Because the usual assumptions of independence and rarity are violated, Damage Tolerance is by far the stronger requirement than Fault Tolerance, although the two are often confused.

A four-ring segment must operate with any number of nodes, from zero to many, and also survive the loss of hubs and/or links totaling up to three. The unit of failure is the node, the hub, and the link. If any component within such a unit fails, the entire unit is deemed to have failed. Fiber links, rather than cables, are the unit of failure because the notion of “cable” is ill-defined mathematically, because a cable can contain any number of fibers, while the fiber link is the smallest breakable unit.

There can be no single point of failure. Therefore, no single piece of equipment, be it a hub, a node, or a link, can be critical. As this hub-plus-link loss limit (three) is exceeded, the segment must degrade gracefully. While loss of all of a segment’s hubs will totally disable that segment, a segment will typically survive the loss of many more than three links, degrading slowly, as shown in the “Performance” section.

Rostering cannot in any way depend on information gathered before the triggering glitch, because that information is almost certain to have been invalidated by whatever caused that glitch. Consequently, every rostering process must independently explore the segment and gather all necessary information for itself anew.

Rostering cannot in any way depend on the arrival times and/or the arrival time sequences of the rostering packets (used to explore the segment) because the node-hub cable lengths (and therefore packet propagation delays) vary widely, and also because some but not all hubs may have bypassed a given node, further scrambling packet arrival order and timing, as seen at the various nodes.

Nodes, hubs, and links must be able to enter and exit the segment in any order whatsoever, all without coordination.

Not all systems require or wish to pay for the full measure of damage tolerance. Hubs, needed mainly to allow unavailable nodes to be automatically bypassed, cannot be required for rostering or data transfer, although some or all fault tolerance will be lost in their absence. Segments may be designed with from zero to four hubs, but all rings of a segment must be identical. One cannot have a mix of rings both with and without hubs in the same segment.

6. The Rostering Process

In the following, it is assumed and required that the nodes of a segment are configured on the up to four rings with NodeIDs assigned in strictly ascending (in the direction of packet propagation) numerical order, and also that the configurations of the up to four rings are exactly identical and parallel, except that the link lengths need not be the same. NodeIDs must be unique, but gaps in the sequence are allowed. These assumptions are requirements of the rostering algorithm.

During data transfer mode, each node receives from ("listens to") exactly one of its up to four optical receivers ("inputs"), simultaneously transmits identical traffic on all optical transmitters ("outputs"), and each node removes ("purges") its own data packets from the segment.

If early packet purging is active, unicast packets will be removed from the segment by their target node, and thus will never return to their sender to be removed. Nodes will still remove (purge) any of their own packets that somehow escape being purged by the target.

The rostering algorithm does two main things. First, it decides which input each and every node shall use. This chosen input is called the "active" input, and different nodes will often choose different inputs to be active. The node ignores the "inactive" inputs, except to detect glitches and/or K30.7 symbols. The sum of these input-choice decisions defines the single composite (logical) ring to be used for data transfer. Second, rostering chooses exactly one node to be responsible for purging orphaned data packets from this chosen single composite ring in data-transfer mode.

Purging is required to prevent packets with undetected damage to their NodeID fields from circulating forever, their number ever growing.

Note that all node inputs and node outputs can be manually enabled and disabled, to allow an unreliable input, output, or link to be ignored. So, the following descriptions apply only to enabled inputs and outputs. A link containing a disabled and/or failed input or output, or a broken fiber, is treated as a failed link. The act of disabling (or enabling) an input or output will generally provoke rostering. Afterwards, disabled inputs are silent and so cannot provoke rostering.

6.1 Rostering in a Hubless Segment

A description of the simplest case, a hubless segment having multiple rings with no missing or broken nodes, but some broken links, follows. For clarity, this simple case ignores some practical problems that will be discussed and solved later in the present description. As will be seen, a node in rostering mode has the following basic duties: to explore the segment's rings with the node's own roster packets, to choose an input based on the NodeCount field in its own returned roster packets, to decide if this node shall be the segment's purge node, to increment and forward the roster packets of other nodes, and to remove its own roster packets from the rings. Each node makes its input-choice decision based only on its own returned roster packets. Packets from other nodes may be incremented and forwarded, but are not otherwise acted upon.

Somewhere, a link is suddenly broken or reconnected. That link terminates on a node's input somewhere. That input reports a glitch, causing the detecting node to enter rostering mode. Upon entry, that node emits identical copies of its own roster packet on all outputs. These packets arrive (after various propagation delays) at some or all of the inputs of the next node in sequence, depending on how many of the intervening node-to-node links are currently functional. Arrival of the first roster packet at this next node causes that node to itself enter roster mode and emit its own roster packet on all outputs. In addition, this next node emits identical incremented copies of these pass-through roster packets on all outputs. As the initial roster packet propagates around the rings, the nodes are recruited into roster mode one by one, and the population of roster packets grows. Each node increments and forwards all roster packets (except its own) on all outputs. Each node processes and discards those of its own roster packets that come back. Each node chooses as its data-mode input one of the inputs from which it received a copy of its own roster packet; it doesn't matter which. (Actually, any input from which roster packets were received will do for this simplified system, but bear with me.) The roster packets also contain the NodeID of the last sender, which identifies the receiving node's nearest upstream neighbor. Because the NodeIDs are assigned in strict ascending order around the ring, in a hubless segment, all nodes except one will find that their nearest upstream neighbor's NodeID is numerically less than their own NodeID. (In a system with hubs, a node may have up to four "nearest upstream neighbors", as will be discussed later.) The one node whose nearest upstream neighbor's NodeID is greater than or equal to the present node's NodeID will be chosen to be the purge node, used to eliminate (scrub) orphan data micropackets while in data mode. The "equal" case covers one-node rings, which occur during network startup. In a one-node ring, one is one's own nearest upstream neighbor, and so the NodeIDs are equal.

The simplified rostering algorithm described just above has two serious flaws, both of which cause hardware throughput limits to be grossly exceeded.

First, we are receiving at full rate on up to four inputs, whereas the node is designed to handle exactly one full-rate input, one fourth of the aggregate input data rate. This problem is solved by transmitting roster packets somewhat slower than one-fourth of the full rate, at nominally four roster packets per microsecond, plus one or two microseconds of delay through the node, so that all nodes can read and process all inputs without overflow or packet loss.

Second, and by far the worst, is exponential growth of the roster packet population. In a segment with "N" nodes and "R" rings, each of N nodes will emit R copies of its own roster packets, and will in addition forward R roster packets for every roster packet received, so the total number of packets to be handled will be N(R^N), a very large number, one with extraordinarily unfavorable scaling, sufficient to completely overwhelm the segment, even if (as planned) roster packets are sent at less than one-fourth the full rate. For example, in a typical 30-node, 4-ring segment, there will be (30)(4^30)= 3.5e19 roster packets handled, which would take something like 168,000 years to work off, even at full gigabit speeds. Although there are some obvious methods to solve the population explosion problem in the above simple hubless segment, such population control methods do not solve the problem in segments with hubs, so the problem will be solved later, in the next section.

6.2 Rostering in a Segment Having Hubs

A description of the usual case, a segment having multiple rings and hubs, some missing or broken nodes, and some broken links, follows. The practical problems ignored in the previous case, and some new ones, are now addressed.

The addition of hubs and missing nodes greatly complicates things. First, nodes may be bypassed by some but not all hubs, in some essentially random pattern. There are no forbidden patterns, and the presence of bypassed nodes creates multiple alternate paths of varying length (counted in nodes, not meters of fiber), and can result in semi-isolated nodes (that can talk but not hear, or can hear but not talk), and also can result in totally isolated nodes (that can do neither). Because the pattern of node bypassing is totally unconstrained, only those properties that survive bypassing can be depended on, such as: the strict ascending order of the NodeIDs of the surviving (non-bypassed) nodes, the direction of packet propagation, the fact that bypass paths are always shorter (measured in nodes) than non-bypass paths, and the fact that the segment’s rings are still rings, albeit smaller. With bypassing, a node can have up to four different “nearest upstream neighbors”, requiring care to ensure that a segment ends up with exactly one purge node. In summary, bypass-invariant properties must be the sole basis for any rostering algorithm expected to work correctly in a segment containing hubs.

If there were no bypassing, no wrap node would be needed, as there would be no way for undamaged roster packets to escape being purged by their nodes of origin. With bypassing, it becomes possible for undamaged roster packets to become immortal orphans, circulating forever, preventing termination of rostering, causing nodes to choose exactly the wrong input, etc. There also needs to be a way to scrub damaged roster packets from the segment. The first order of business is therefore to kill the orphans as quickly as possible. What then, precisely, identifies an orphaned roster packet? Because packets are intended to go exactly once around the segment, returning to and being discarded by their source node, and none other, any roster packet that manages to bypass its source node is by definition an orphan, and should be discarded as soon as possible, preferably by the next downstream node to receive it.

A downstream node will normally forward roster packets from all upstream nodes. How then does a downstream node know which of these roster packets are orphans? The key is to have a “wrap node” that inspects and marks all passing roster packets.

The packets are marked using the classic “policeman's algorithm”. The policeman is monitoring a parking lot for cars that have been parked too long. Once an hour, the policeman patrols the parking lot, carrying a piece of chalk. For each car, he marks the right front tire with the chalk. If he finds that he has already marked the tire, he knows that the car has been there at least an hour, so he gives the car a parking ticket, and continues to the next car.

Because of bypassing, there can be up to four wrap nodes while in rostering mode. To be precise, it is the individual inputs of the node that decide to mark a passing roster packet by setting its WrapFlag bit, and we will have a total of four inputs per segment that are marking packets, spread out over up to four adjacent nodes.   A given node may be marking packets at none, some, or all of its four inputs. We somewhat imprecisely call any node where at least one input is marking packets a wrap node. If there are four wrap nodes, each will be marking at only one input. As any specific roster packet will see only one of the up to four wrap nodes, the following is for simplicity and clarity written as if there were only one wrap node.

The basic problem to be solved is to find an algorithm that will pass all rostering packets from other nodes the first time those packets are encountered, but drop any that are seen a second time, in spite of random patterns of bypassing. The present node is called NodeID and the originating node is called OriginID. We want the present node to pass packets on the first encounter, but drop them on the second encounter. In all cases, if NodeID equals OriginID, we have a roster packet returning to its node of origin, so the packet is dropped. If not equal, the following cases are found:

On the first encounter, there are two cases, depending on where the wrap node is in relation to the present node and the originating node (from which the roster packet came). In the first case, where the roster packet encounters the present node before encountering the wrap node, the packet's WrapFlag bit is zero and OriginID is less than NodeID. In the second case, where the roster packet encounters the wrap node before encountering the present node, the packet's wrap bit is one and OriginID is greater than NodeID. In both cases, the packet is forwarded. (In practice, a packet with wrap bit of zero is always incremented and forwarded, without inspection of the other fields.)

On the second encounter, there are also two cases, depending on where the wrap node is in relation to the present node and the originating node, but the packet's WrapFlag bit is always one because the wrap node has already seen this packet. In the first case, where the roster packet encounters the present node before encountering the wrap node (a second time), OriginID is less than NodeID, and the packet is dropped. In the second case, where the roster packet encounters the wrap node (a second time) before encountering the present node, the OriginID is greater than NodeID, and the packet is dropped, if it ever gets this far. Normally, the wrap node will already have dropped this packet based on the packet's WrapFlag bit alone. If the present node or the origin node is also the wrap node, comparison of NodeID with OriginID gives no added information, and the wrap node will scrub using only the packet's WrapFlag bit.

ID logic —> **Pkt's wrap bit:**	**OriginID LT NodeID**	**OriginID EQ NodeID**	**OriginID GT NodeID**	**Wrap Node only**
Was Zero	Forward	Drop: always scrub own packet	Drop	Set packet's wrap bit to one; forward
Was One	Drop	Drop: always scrub own packet	Forward	Drop

The above table recaps the previous few paragraphs. For a more detailed description, including all the various special cases, see the Rostering Flowchart and its associated notes.

The following description of the rostering process includes hubs.

Recall that a node in rostering mode has the following basic duties: to explore the segment's rings with the node's own roster packets, to choose an input based on the NodeCount field in its own returned roster packets, to decide if this node shall be the segment's purge node, to increment and forward the roster packets of other nodes, and to remove its own roster packets from the rings. To this list we add the duty of filtering roster packets from other nodes, to control the population explosion.

Each node makes its input-choice decision based only on its own returned roster packets. Packets from other nodes are filtered and may or may not be incremented and forwarded, but are not otherwise acted upon.

Somewhere, a link is suddenly broken or reconnected. That link terminates somewhere, on either a node's or a hub's input. The input reports a glitch, either causing the detecting node to enter rostering mode, or causing the detecting hub port to emit a K30.7 symbol, receipt of which causes the next non-bypassed node to enter rostering mode. Upon entry to rostering mode (by either route), the node emits identical copies of its roster packet on all outputs. These packets arrive (after various propagation delays) at some or all of the inputs of the next node or nodes in sequence, depending on how many of the intervening node-to-node links are currently functional. Arrival of the first roster packet at a next node causes that next node to itself enter roster mode and emit its own roster packet on all outputs. In addition, this next node increments the "NodeCount" field in each received roster packet, and emits identical copies of these incremented pass-through roster packets on all outputs. As the initial roster packet propagates around the rings, the nodes are recruited into roster mode one by one, and the population of roster packets grows. Each node increments and forwards all roster packets (except its own) on all outputs. Eventually, each node receives back, processes, and discards its own roster packets. These packets each carry the NodeCount, a count of the nodes they traveled through on their way around the ring. Each node chooses as its data-mode input that input from which the node received back a copy of its own roster packet carrying the maximum value of NodeCount field seen. If more than one input received a roster packet with the same maximum value of the NodeCount field, any one of those inputs may be chosen. Typically, the lowest-numbered input is chosen.

The roster packets also contain "LastTxID", the NodeID of the last sender, which identifies the receiving node's nearest upstream neighbor. Because the NodeIDs are assigned in strict ascending order around the ring, all nodes except one will find that their nearest upstream neighbor's NodeID is numerically less than their own NodeID. (With bypassing, up to four nodes may make this discovery.) The one node whose nearest upstream neighbor's NodeID is greater than or equal to the present node, as seen by the selected input, will be chosen to be the purge node, used to eliminate (scrub) orphan data micropackets while in data mode. The "equal" case covers one-node rings, which occur during network startup. In a one-node ring, one is one's own nearest upstream neighbor, and so the NodeIDs are equal.

A node that never receives any of its own roster packets is required to go directly into listen-only mode, as it (by definition) is not in the ring and so cannot participate in the segment, and thus can only cause trouble if it attempts to speak. In theory, a node that cannot hear will be isolated when rostering is complete. However, enforced silence is the safest policy. However, this isolated node shall not turn its transmitters off, as this would cause pointless re-rostering, or, in extreme cases, prevent the completion of rostering.

6.2.1 The Population Explosion is Contained

We still have a population explosion problem, but now we have node counts, and this added information is the key. Each node makes its final choice of which input to listen to based only on the maximum NodeCount it sees in its own returned roster packets. Because all roster packets carrying a lesser count cannot affect the final choice, they need not be forwarded by other nodes. So, each node keeps a 256-element array of MaxCount values, indexed by OriginID, where the maximum NodeCount seen by a node from each of the various other nodes is kept, and a roster packet is forwarded if and only if its NodeCount exceeds the current MaxCount for its node of origin. This node-by-node filtering of the roster packets prevents the population explosion by min-max pruning of the growing tree of roster packets.

Another measure controlling the roster packet population is the time quantization resulting from the nodes' scanning of all four inputs, digesting any received roster packets, and only then emitting a few incremented and forwarded packets, having discarded all roster packets of lesser NodeCount, regardless of their order of arrival. Typically, the node cycles at about one megahertz, simultaneously receiving up to one roster packet per input, processing the roster packets collected in the last cycle, and outputting the results of the cycle before last. In short, the node is acting as a little three-stage pipeline. If necessary, the scan can be done multiple times per output bundle, further compacting and filtering the outputs. Time quantization works because in general the most important roster packets arrive last, having traveled through the greatest number of nodes.

Even with the various population-control measures, the swarm of roster packets can become quite large, large enough that a "thermodynamic" (also known as "statistical mechanics") analytical approach is useful. The essence of thermodynamics theory is to ignore the individual molecules, and instead deal only with the average properties and statistics of large populations of molecules. In the case of rostering, the roster packets are the molecules, the swarm is the large population, and we are in effect using ensemble averages over all possible configurations of link lengths and fault patterns.

Another analytical simplification is to note that each node rosters independently of all other nodes, so the total roster packet population is simply the sum of the individual nodes' own roster packet populations, or on average N times that of the typical node, for an N-node segment. One can therefore analyze the artificial configuration where only one node emits its own roster packets, and the N-1 other nodes only forward those roster packets, without emitting their own roster packets, and then simply multiply the results by N to get the result for the full segment. To be precise, one sums N rotated versions of the results from that one typical node.

After passing through a few nodes, the descendants of the initial roster packet will have grown into a swarm of more or less steady population size touring the segment's rings. This population can be decomposed into two subpopulations: one whose NodeCount values increase monotonically from first packet to last (ascending), and one whose NodeCount values decrease or remain equal from first to last (non-ascending). The relative sizes of these two subpopulations will depend on the details of node bypassing, but can be estimated, and we will assume in the present example that the ascending subpopulation is one-half the size of the nonascending subpopulation. (A worst case is when alternate nodes are bypassed.) Note that it's a property of bypassed rings that the last packet to arrive often has the largest NodeCount, because it traversed the largest ring, so the "ascending" subpopulation often contains the deciding roster packet. In largely undamaged segments, which have very little bypassing, the nonascending subpopulation will instead be by far the largest. The effect of the MaxCount filter in each node is to absorb the nonascending subpopulation at each step, but to forward the ascending subpopulation. For example, in a 4-ring segment, only the ascending subpopulation is quadrupled and passed on, resulting in an effective swarm growth rate of (1/3)*4= 1.33 per node in the present example. In other words, the packet population would grow by one third at each node. This growth rate, if permitted, would be sufficient to saturate the segment, with massive loss of roster packets, after a few nodes. The effects of time quantization have been neglected here. Also, depending on bypassing and fiber lengths, the ordering of packets in the surviving ascending subpopulation becomes somewhat scrambled, regenerating the other subpopulation, at the expense of the ascending subpopulation, further controlling population growth.

If each node waits long enough so that for every two ascending-subpopulation roster packets, only one packet is quadrupled and passed on, the swarm growth rate will then be reduced to ((1/3)(1/2))*4= 0.67, which is small enough to ensure that the population remains within the physical capacity of the segment. How long is long enough? In saturation, each node is handling roster packets at full rate, about four per microsecond. If each node instead accumulates and filters roster packets for two or three microseconds, the growth rate of the ascending subpopulation will be decreased by a like factor.

By the theory of geometric growth, the growth rate must absolutely be less than one, and, for robustness, it would be well for the growth rate to be significantly less than one. As mentioned above, for a N-node segment, the total roster packet population is given by the formula N(R^N), where R is the number of rings in the segment, which led to a population of (33)(4^33)= 2.4e21 roster packets in a 33-node, 4-ring system. With the above population control measures, the N(R^N) formula no longer applies, and the population is reduced to a few hundred or thousand packets, and does not grow exponentially as it travels.

How many roster packets in total will be handled by each node in an undamaged R-hub segment? Each node will independently emit R roster packets. The other nodes will drop (filter out) all but one of these R roster packets upon receipt, and emit R (incremented) copies of this one surviving roster packet. So, as seen on the links, each node will cause a little R-packet bundle to circumnavigate the segment. As there are N nodes, each node will handle a total of N*R packets. In our 33-node, 4-hub example, this is 132 packets. This undamaged-segment case yields the lower bound on roster packet population.

What is the upper bound on roster packet population? Because of the effects of time quantization, it’s hard to compute, but the following observations are relevant. The growth rate depends on the amount of bypassing of nodes, and not on the total number of nodes in the segment. Assuming unit delay in the fibers (the worst case) without bypassing, all paths around the segment take the same amount of time, and the nodes will forward and replicate only one of the packets from its upstream neighbor. This is the undamaged-segment case discussed earlier. With bypassing, shorter paths generally take less time, and so the packets will arrive in precisely the wrong order (that is, biggest NodeCount arrives last), and no filtering will occur (unless the packet rate is sufficiently high that time quantization becomes effective).

If the segment can be divided into distinct bypassed and unbypassed regions, there is a simple approach to computing the growth rate. In each bypassed region, compute how many alternate paths having distinct transit times there are, assuming all fibers, nodes, and hubs have unit delay. (Here, “distinct” means differing by more than one time quantization period.) An unbypassed region has exactly one distinct path, as all physical paths take the same amount of time. A region can have from one to R such distinct alternate paths, but is required to contain at least as many nodes as paths. Notice that the number of distinct paths is not the same as the number of bypassed hub ports. In general, it takes multiple broken fibers (and therefore bypasses) to implement each alternate path. Compute the overall ring packet growth ratio by taking the product of the path counts of all the regions in the segment. Each node in an unbypassed region will emit R roster packets and receive R*<packet growth ratio> packets in return. Nodes in a bypassed region will often receive fewer packets in return, but here we will assume the bounding case, that all nodes receive the same number of their own packets back.

For a numerical example, assume that our 33-node 4-hub segment is divided into 32/4= 8 four-path regions plus one one-path segment, for a packet growth ratio of (4^8)(1^1)= 65536, so each roster packet sent out will yield (4)(65536)= 262,144 packets back worst case. In practice, in a 33-node segment, we would see only a small fraction of the maximum (33)(262144)= 8.651e6 roster packets. Why a fraction? Two reasons. First, because the population arising from a given starting packet grows by a factor in each region, and only the last region sees the full swarm. Second, time quantization will limit the peak rate.

How many broken fibers does this example require? Each of the eight four-path regions requires six broken output fibers to implement, so the entire ring would require at least (6)(8)= 48 well-chosen broken fibers. Most 48-break patterns will yield much smaller populations.

One can model the effects of time quantization by using the Poisson distribution to estimate the average number of ascending subpopulation packets that will land in a time quantization period, and thus what the population growth rate will be, because only one of those arriving packets will be replicated and forwarded. Time quantization (TQ) does not control the total roster packet population so much as it limits the local population growth rate. At low packet densities, TQ has little effect. At high packet densities, TQ will dominate because more and more roster packets will fall within a given TQ period, and so will be dropped rather than replicated and forwarded.

6.3 Exit from Rostering Mode

Exit from rostering and subsequent entry into data-transfer mode is controlled by timers in the nodes, on a node-by-node basis. When rostering starts, the roster packet population grows until limited by the MaxCount filter with time quantization, then declines as more and more roster packets have made it all the away around the segment's rings and have been absorbed by their nodes of origin. This process takes slightly more than two ring tours, one to recruit all other nodes, and one more for the last-recruited node's roster packets to return. After this, the segment falls silent, until the original triggering node, which started first, times out, returns to data mode, and releases its pent-up data packets in a flood. These data packets propagate around the segment, recruiting nodes to return to data mode.

The middleware shall ensure that this flood of pent-up packets is released at a controlled rate, to prevent data packet storms.

7. Entry and Exit of Nodes

In the beginning, the segment is empty. All nodes are bypassed, but listening to the hubs, which emit a continuous stream of FC-PH FC-1 “IDLE” ordered-set 32-bit longwords.

On power-up, each node's RTFC hardware performs self-test, and, if successful, begins to listen, but does not attempt to enter this zero-node segment until so commanded by the host computer's software (middleware). The nodes' receivers instead all acquire, after a hardware-determined delay of up to one millisecond, symbol and longword code phaselock while listening, but transmit nothing. The transmitter logic will also phaselock to the received signal, but the optical output will be dark. If the power-up self-test fails, the

RTFC hardware will refuse subsequent commands to enter, and thus will not join the segment, to protect the segment.

Later, somewhere, a node's host software, having initialized itself, decides that it's time to enter the segment, and commands its RTFC card to enable its optical transmitters, which then begin to emit a copy of what's being received from the hub ports, perhaps only a stream of IDLEs, back to those same hub ports. The hub ports at each of the up to four hubs, after a hardware-determined delay of up to one second, declare that they are receiving a sufficiently strong optical signal, and have acquired both symbol and longword code phaselock on the node's modulated light, and so un-bypass the node at each of its associated hub ports. The hub ports wait until the input has been both of adequate signal strength and the receiver has been in full phaselock continuously for about three-fourths of a second to ensure that chattering inputs are automatically ignored. The act of changing the bypass state of the hub ports (in this case, from bypassed to active) causes those hub ports to emit K30.7 symbols, which in turn trigger the next downstream node to enter rostering mode. As this is the first node to enter the empty ring, the node triggers itself via the hubs. Rostering, as described above, yields a one-node segment.

Even later, a second node's host software decides as before to enter the segment, and commands its RTFC card to enable its optical transmitters, which then begin to emit a copy of what's being received from the hub ports, perhaps only a stream of IDLEs, back to those same hub ports. This proceeds as before, except that this is the second node to enter the previously empty ring, and the entering (second) node triggers the first node (via the hubs) to enter rostering mode, yielding a two-node segment after rostering is complete.

Because rostering is so fast, if a node attempts to enter while rostering is in progress, rostering will simply be re-invoked, allowing correct insertion of all new nodes at once. Anyway, the above process repeats as described, for node after node.

Keeping all receivers, active and inactive alike, in hub and node alike, in phaselock allows the segment disruption time to be measured in microseconds, rather than milliseconds, due to the design of the widely-used Fibre-Channel transmitter and receiver integrated circuit chipsets. Although the node entry delay is about one second, the ring disruption time is measured in microseconds.

A node that wishes to exit simply disables all of its optical transmitters. The up to four corresponding hub ports all detect the loss of modulated light after a few microseconds, bypass the node, and emit the K30.7 symbols, which will trigger the next downstream node or nodes to begin rostering, as described before.

7.1 Detailed Node Insertion and Exit Sequences

7.1.1 Insertion

Each hub port always transmits a copy of the passing traffic to the associated node, bypassed or not. This keeps the hub port's transmitter in phaselock. If the node can hear the hub port, the node's corresponding receiver and also transmitter logic will also lock onto the hub port's modulated light, even if transmitter output is currently disabled. The effect of enabling the transmitter output is thus almost instantaneous, as the node's receiver and transmitter are already in phaselock with the segment's traffic. The hub port, upon hearing modulated light from the node's transmitter, begins the process of acquiring phaselock. As the hub port's receiver has neither bit/word frequency lock nor bit/word phaselock, this takes a random amount of time of up to one millisecond, determined by the internal design of the Fibre Channel ENDEC (Encoder-Decoder) chip. When the hub port's ENDEC chip's announcement of full codelock has lasted for at least 100 µSec and less than one second (exact value being implementation defined), the hub port logic unbypasses the node, thus inserting it into the segment's ring. This switch causes a phase hit (but not a frequency hit), which forces the next ENDEC in line to reacquire phaselock, which takes only a few microseconds, because frequency did not need to be re-acquired, only phase. Switchover itself takes a few 37.65-nanosecond logic clock periods, and is announced by the generation of a K30.7 symbol, which will cause the next downstream node to begin the rostering process.

Insertion of a node can take as long as one second.

7.1.2 Exit

A node desiring to exit simply disables all of its transmitters, causing modulated light to disappear from the associated hub port's receiver, which in turn causes the ENDEC chip to declare loss of codelock within a few microseconds. The loss of the codelock signal triggers the immediate bypassing of the node, which takes a few 37.6-nanosecond clock periods, and is announced by the generation of a K30.7 symbol, which will cause the next node in line to begin the rostering process, as before.

Exit of a node takes microseconds, and is always followed by rostering, which takes less than one millisecond from the last fault to complete, in typical segments.

8. Performance of RTFC segments

These performance calculations assume the standard Fibre Channel [FC-PH] signaling rate of 1.0625 Gbaud and the Fibre Channel FC-0 and (a subset of the) FC-1 signal structure. Matters such as optical carrier wavelength (850 nm versus 1300 nm), fiber type (50-micron multimode, 62.5-micron multimode, or single-mode), fiber bandwidth, and the various optical budgets, all have no significant effect on the present computations.

8.1 RTFC is Random-Access, not Sequential-Access

The central architectural difference between RTFC and sequential packet-based communications systems (SPBCS) such as ethernet, FDDI, ATM, and the like, is that RTFC is random-access, while sequential packet-based comms (SPBCS) are sequential-access. This is precisely the difference between disks and tapes: In practice, disks are much faster than tapes, even if their nominal data transfer rates are identical, because of the gross differences in effective latency.

Even if the end user is message-based, the message-carrying middleware's performance and functionality are much improved if the underlying transport hardware is random-access. This is why realtime intertask communications done within a computer have traditionally been implemented upon shared memory, even if the overall architecture of the applications was loosely (that is, message) coupled.

A classic problem with SPBCS is message (packet) priority inversion: small, high-priority messages get stuck behind large, low-priority messages. In overloaded systems, the transmit queues grow, and one can get some advantage by moving high-priority messages to the front of the line, but the SPBCS transport media are effectively FIFO. (Priority has no effect unless the media is oversubscribed.) With RTFC, with its random access to the words comprising the packets, the messages simply cannot get in each other's way. Thus, RTFC-based networks (and thus systems built upon them) will be more responsive than SPBCS networks, even if the nominal throughputs are identical.

In addition, the aggregate network capacity of RTFC is much greater than most SPBCS systems, even if they have nominally equal point-to-point throughputs, because in RTFC, everybody can talk to everybody at once.

Switched SPBCS networks can in theory match RTFC's large aggregate (compared to point-to-point) bandwidth, but cannot match RTFC in either latency or random-access, and multicast/broadcast may not be all that well supported, because switching doesn't improve the fundamental bandwidth of the individual channels from the switch (hub) to the nodes, and because available switched-media LANs do not directly support multicast/broadcast, due to the fundamental difficulty of implementing multicast/broadcast in switched media. Typically, multicast/broadcast is implemented as repeated unicast in such LANs, making performance and scaling critical issues.

The non-priority-inversion advantages of RTFC’s random access apply to the endpoints, and not just to the (perhaps switched) communications media.

The Scalable Coherent Interface [SCI] standard makes many of these same points. SCI is a random access communications standard similar in general objectives with RTFC. However, SCI is more complex than required to implement RTFC’s network cache (a form of distributed shared memory), is not damage tolerant, does not support required standard busses such as VMEbus (instead defining its own bus right down to the connectors), is not widely implemented or supported by computer vendors, and thus is expensive.

Both RTFC and SCI implement a distributed cache memory within which the middleware keeps its various data structures.

Nor do these SPBCS, as currently implemented, have built into them anything like the level of damage (vice fault) tolerance provided by RTFC, forcing the host software to implement "active redundancy" (double send, double receive, discard duplicate) over multiple parallel SPBCS systems or the like. Experience teaches us that damage tolerance cannot be added on, it must be part of the original design to be effective. Nor can one achieve damage tolerance by use of parallel redundant fault-tolerant communications systems, because a single damage event will provoke all parallel systems into simultaneous fault recovery, causing a total communications outage for however long it takes the fastest system to recover, and because the individual fault tolerant systems may in any case fail totally to cope with massive damage, for which they were never designed.

Every low-level communications media has a more or less fixed data carriage capacity (the “bit budget”), which can be spent in a number of ways, depending on which protocol is chosen. There are a number of different communication system protocol families in wide use, each optimized for a different desired characteristic. Protocols having large variable length packets (and thus packet headers) are optimized for very large networks and sustained throughput, at the expense of latency and predictability of timing. Protocols having very short packets (and thus headers) are optimized for low latency and reasonable predictability of timing, at the expense of throughput. Protocols having fixed packet schedules, such as the slotted-ring protocols, are optimized for predictability of timing and reasonable throughput, at the expense of latency. Applications choose protocols according to how well the protocols and available hardware and software meets their needs.

RTFC uses a register insertion ring protocol (RIP), which inherently has twice the capacity of a token ring protocol on the same communications links, because of greater protocol efficiency [Liu]. RIP is most useful for short, fixed-length packets, which is one reason that FDDI (ANSI X3T9) and Token Ring (IEEE Std 802.5) use token ring protocols. The greater capacity of RIP compensates to a great degree for the overhead of the short packets.

8.2 Ring Tour Time

A “ring tour”, the time it takes a micropacket to circumnavigate a segment, is the central determinant of both update latency and damage recovery time of that segment. Throughput and ring tour time are unrelated. Ring tour time is reported as the maximum possible value for the segment, which is seen when no nodes are bypassed. The ring tour time does not depend on how many hubs there are, except that the fiber links to various hubs may vary in length, so we consider only the maximum length links, and, for the purpose of computing ring tour time, speak as if all systems consisted of one hub and a number of nodes and links. What is computed in the following paragraphs is the ring tour time under light traffic while in data transfer mode. Under heavy ring traffic, the various queues (hardware FIFOs) will grow, raising the effective ring tour time by a large factor. In rostering mode, the ring tour time may be slightly longer, as the per-node delays are somewhat larger than in data transfer mode.

The elements of a segment are nodes, links, hub ports, and hub cards. Each hub card contains four optical hub ports plus one copper hub port for connections between the hub cards within a hub. A port contains two paths, in and out, which may be either optical or electrical. Each node connects to an optical hub port using two glass-fiber links, one in each direction. The overall physical topology is that of a star, with the hub in the center and the nodes at the points.

8.2.1 Node Delay

There are two components of net delay through a node, these being the optical receiver plus input FIFO path, and the output FIFO plus optical transmitter path respectively. Under light traffic (little queuing in the FIFOs), the net average delay is (2)(135.6)= 271.2 nanoseconds for SRAM-based RTFC nodes, each FIFO accounting for 135.6 nanoseconds of delay.

8.2.1.1 Queuing-Theory Model of RTFC Node Delay

Ring traffic is the aggregate of traffic from multiple, independent, and uncorrelated sources, the software in the nodes. This fits the classic definition of a Poisson process. All micropackets, from whatever source, make one circuit of the ring. (The exceptions are unicast and cut-through traffic.) The traffic in all links is therefore equal, so we need speak only of the “ring traffic”, and the micropacket inter-arrival time observed at any node will be exponentially distributed. This Poisson-process approximation breaks down only when flow control, triggered by node output queue (FIFO) growth, forces the source (host software) to slow down, thus violating the Poisson-process assumptions of independence and non-correlation.

Each node can handle the full ring traffic, unless that node is adding too much of its own traffic. It takes a fixed amount of time (135.6 nS) for a node to send a micropacket on to the next node, and there is a queue (output FIFO) between the node and its output, handling the sum of incoming ring traffic and the node's added traffic. We therefore have an example of a classic single-server queue with fixed service time and an exponential arrival time distribution. All this implies that the ring traffic will from time to time randomly pile up and fill the FIFO (and thus potentially lose packets) at a fairly predictable rate, given the average ring packet arrival rate and the depth of the FIFO. In fact, flow control will intervene to prevent micropacket loss, so queuing theory will instead predict how often flow control will be invoked. The rate at which flow control is invoked can as a matter of design be made arbitrarily low, as discussed in the section on flow control.

As traffic increases, the output FIFO delay will grow somewhat, but the effect isn't all that great until traffic exceeds about 80% of the theoretical throughput, and the overall behavior is well-modeled by a single-server queue (the output FIFO) with an exponential inter-arrival time distribution (between data micropackets) and a constant 135.6-nS service time, until flow control takes effect. This is in addition to the 135.6-nS delay incurred by the node's input FIFO.

As a matter of design of the embedding system, the average throughput should be kept below 50% of the theoretical throughput, to ensure low and predictable latencies, even when peaks exceed this design average throughput. The following analyses assume that this limit has been observed.

8.2.2 Link Delay

The links consist of 62.5-micron graded-index FDDI-grade "62.5/125" multimode Silica optical fiber, whose index of refraction is 1.5014 at 850 nm, and 1.4966 at 1300 nm, as computed from optical time delay measurements [Corning]. The two refractive indices differ by only 1.5014/1.4966= 1.0032, or 0.3%, so we will use their average. Manufacturing variation is likely to be larger. The speed of light is 299,792,500 meters per second in vacuo [CRC, page F-191].

The average group delay is therefore ((1.5014+1.4966)/2)/(299,792,500)= 5.0001e-9 seconds per meter, or almost exactly five nanoseconds per meter of fiber, or 1.52 nanoseconds per foot of fiber. This is almost exactly two-thirds of the speed of light, and is about the same speed as for electrical signals in coax or twinax. This propagation delay is incurred twice per node, once in the hub-to-node fiber, and once again in the node-to-hub fiber.

The link delay will vary slightly with ambient temperature. Although no temperature coefficient is specified in the fiber datasheets, it is about ten parts per million per degree centigrade, and is thus negligible [Carr]. A more significant effect is that the transmitter lasers' optical wavelength varies with temperature, and the propagation delay of light in fiber varies with the wavelength, so the effective delay varies with laser temperature. A typical room-temperature variation in laser output wavelength is 0.3 nm per degree centigrade at 850 nm, or 0.3/850= 350 parts per million per degree centigrade. However, neither of these effects is significant in RTFC, as there are elasticity FIFOs in both nodes and hubs, and each link runs independently of the others.

8.2.2.1 Links Considered as FIFOs

As discussed in section 8.3, a micropacket is on average (130)(10/8)= 162.5 code bits long, taking 162.5÷(1.0625e9)= 153 nS and occupying (1.529e-7)÷(5e-9)= 30.59 meters of fiber, per micropacket. So, a 300-meter length of fiber can store about ten micropackets.

8.2.3 Hub Delay

There are three components of hub delay. A hub port may be either active or bypassed, with different values of delay. We will have some bypassed hub ports, even if no node is bypassed, because each hub card has four optical hub ports, and the number of nodes will not in general be an exact multiple of four. Each hub card incurs an added fixed per-card delay overhead to cover the card-to-card copper connection ports. So, the three components of hub delay are: active hub port, bypassed hub port, and hub card overhead. The hub cards are connected into a ring using copper intrahub (coax or twinax) jumpers between the copper connection ports, the choice between coax and twinax varying by manufacturer.

Hub ports also contain FIFOs, but these queues never overflow, because hubs add no traffic. The queuing effects are included in the above Hub Delay. The Hub port FIFOs are intended to decouple hub port inputs from outputs, allowing the hub port ENDECs to totally regenerate the FC-PH signals, thus preventing the accumulation of jitter. The issue is that each transmitter is controlled by its own transmit oscillator crystal, and each receiver phaselocks to its upstream (not local) transmitter, and received data must be re-clocked for transmission, because each transmitter-media-receiver link is independent of all others. The FIFOs between receiver and downstream transmitter act as elasticity buffers to absorb the slight differences in link operating frequency, allowing the links to be joined into a ring without overruns or underruns.

There is an elasticity FIFO associated with each optical or intra-hub copper link transmitter. The elasticity buffer delay is buried in delay block t2 and/or t4 (details depending on implementation). Putting the FIFO in t2 allows ports to be looped back on themselves. These delay blocks are discussed below. The minimum theoretical FIFO size is 4 or 5 longwords, but implementations use at least 16 longwords. The theoretical minimum depends on the maximum transmit crystal frequency tolerance plus worst-case timing between receiver clock and next transmitter clock phases.

In short, the segment is designed as a plesiochronous ring. The tolerance on these transmit crystals is required to be plus or minus 100 parts per million total (not per degree) or less, over the entire Fibre-Channel temperature range of 0° to 70° C.

In the following computations, the hub has a 26.56-MHz logic clock at a 1.0625-Gbaud signalling rate, so the fundamental clock period “Tclock” equals 37.65 nanoseconds.

8.2.3.1 Active Hub Port Delay

There are four components of delay in an active (that is, non-bypassed) hub port, denoted t1, t2, t3, and t4 respectively. The path of a micropacket through an active hub port is t1, t2, hub->node fiber, node logic, node->hub fiber, t3, and t4, and then t1 of the next hub port, in that order. The actual logic delays are t1 equals Tclock, t2 and t3 both equal twice Tclock, and t4 equals either three or four Tclock periods, randomly with a uniform distribution. So, on average, in numbers, t1= 37.65 nS, t2=t3= 75.30 nS, t4= (3.5)(37.65)= 131.78 nS, and their sum is 320 nanoseconds. This delay is incurred once for each active node in the segment.

8.2.3.2 Bypassed Hub Port Delay

There are two components of delay in an inactive (that is, bypassed) hub port, denoted t1 and t4 respectively. The path of a micropacket through an active hub port is t1 and t4, in that order. The actual logic delays are as described for "Active Hub Port Delay", above. So, on average, in numbers, t1= 37.65 nS, t4= 131.78 nS, and their sum is 169.4 nanoseconds. This delay is incurred once for each bypassed (or absent) node in the segment.

8.2.3.3 Hub Card Delay

There are four components of delay in each hub card, denoted t1, t2, t3, and t4 respectively. The path of a micropacket through a hub card is t1, t2, the four identical hub ports, t3, and t4, in that order. The actual logic delays are as described for "Active Hub Port Delay", above. So, on average, in numbers, t1= 37.65 nS, t2=t3= 75.30 nS, t4= 131.78 nS, and their sum is 320 nanoseconds. This delay is incurred once for each hub card in the segment, regardless of the bypass states of the associated hub ports.

8.2.3.4 Ring Tour Time — Rule of Thumb

By assuming a random "typical" segment, we can average over many possible segment configurations to arrive at a much simplified ring tour time computation involving only the total number of nodes and the average distance from nodes to hub. For each node, add (320 + 271.2 + 320/4)= 671.2= 670 nS per node. For the node-hub cables, add 10 nS per meter of dual-fiber cable (total for all nodes).

8.2.3.5 Ring Tour Time — Scaling Law using Network Diameter

To compute the ring tour time "Trt" (in nanoseconds) given a N-node segment of physical diameter "D" (in meters), the above rule of thumb becomes: Trt = N(670 + 5D). Note that the network diameter is twice the maximum link length.

8.2.3.6 Example: Exact Computation of Ring Tour Time

Consider a typical segment containing 33 nodes, a hub, with an average link length of 125 meters. Because 33/4= 8.25, the hub will contain 33 active hub ports, one for each of 33 nodes, plus 3 bypassed hub ports, in (33+3)/4= 9 hub cards.

The total node delay will be (33)(271.2)= 8,953 nS. (14%)

The total fiber delay will be (33)(2)(125)(5)= 41,250 nS. (64%)

The total active hub port delay will be (33)(320)= 10,560 nS. (16%)

The total bypassed hub port delay will be (3)(169.4)= 508 nS. (1%)

The total hub card delay will be (9)(320)= 2,880 nS. (5%)

Total of the above is 64,141 nS, or about 64 µS. (100%)

The total ring tour time for this example 33-node system is therefore about sixty microseconds, and is dominated (two-thirds) by propagation time in the glass fiber. These numerical results are typical for shipboard applications.

For comparison, recompute the above example using the rule of thumb of 670 nS per node plus 10 nS per meter of total in-plus-out node-hub cable: (33)(670)+(33)(125)(10)= 63,360 nS, which is within 64.14/63.36= 1.0123, or within about 1% of the exact computation.

The network-diameter approach will yield the same answer as the rule of thumb approach.

8.2.3.7 Example: Maximum Ring Tour Time

In a 256-node segment with 300-meter links, the ring tour time will be 256(670+5*2*300)= 939,520 nS, or 0.94 mS, or about one millisecond.

8.3 Theoretical Throughput of RTFC Variants

This section computes the precise theoretical throughputs of RTFC at the 1.0625-Gbaud signaling rate, for various combinations of memory type (SRAM, DRAM), and micropacket type ("Data" or "D64 Block-Mode"). Data packets are used for "normal" longword data transfers, one at a time. D64 Block-Mode packets are used for "DMA64" data transfer bursts.

The FC-PH FC-0 and FC-1 codes used to carry RTFC data around the segment are generated by use of standard Fibre-Channel chips, such as the TriQuint "TQ9309" ENDEC in so-called "Proprietary Link Mode" (PLM), where "proprietary" means simply that one is not using all of FC-PH. In PLM, after each block of user data, the ENDEC chip emits one FC-PH "EOFa" ordered set, followed by zero or more FC-PH "IDLE" ordered sets. These ordered sets are all 32-bit longwords [FC-PH].

For SRAM (Static RAM), each 96-bit micropacket is followed by one 32-bit EOF longword, and there is one added 32-bit IDLE longword inserted after every sixteen such (96+32= 128-bit) padded micropackets, for a total of 16 micropackets, with 16 EOFs plus one IDLE per sequence.

For DRAM (Dynamic RAM), each 96-bit micropacket is followed by one EOF longword and perhaps one IDLE longword, and there is one added 32-bit IDLE longword inserted after every sixteen such (96+2*32=160-bit) double-padded micropackets, for a total of 16 micropackets, followed by 16 EOFs plus 17 IDLEs per sequence, assuming the added IDLEs are needed.

The extra IDLE after 16 padded or double-padded micropackets prevents manufacturing tolerances on transmit signaling rate crystal oscillators, $\pm$ 100 parts per million, from causing overruns and lost data, by slightly diluting the output data stream, thus ensuring that a node's transmitter is always able to send at a slightly higher rate than data can be received. This dilution is actually done only to traffic being added to the segment, and not to traffic that is just being passed through a node. The dilution factor is 1/(4*16)= 2% maximum. For conservatism, these throughput estimates assume that all traffic is diluted.

For DMA64, a maximum of 64 bytes of data can be transferred per burst. This requires one address-only micropacket plus 8 data-only micropackets, for a total of 9 micropackets to transfer 64 bytes of data.

These micropacket and EOF/IDLE padding longword databits are converted to optical code bits at the FC-PH code rate of 8 data bits yields 10 code bits, and the resulting codebit rate is divided by the signaling rate, 1.0625 Gbaud, to yield the time to send the item. A "baud" is one code bit per second. The item's payload in bytes is then divided by this (very short) time to get the throughput in bytes per second, which is rounded to three significant digits.

So, to tie it all together, the Normal-SRAM throughput is computed as an example:

The 16 micropackets plus padding occupies 16*(96+32)+32= 2,080 data bits. This corresponds to 2080*(10/8)= 2,600 optical code bits, which takes 2600/(1.0625e9)= 2.45 µS for the padded 16-micropacket sequence. Each micropacket carries 4 bytes of payload, for a total payload of 4*16= 64 bytes for the 16-micropacket sequence, so the throughput is 64/(2.45e-6)= 26.1538e6 bytes per second, which rounds to 26.2 MB/sec.

Throughputs in megabytes/second, rounded to three significant digits:

Transfer type -->	<u>Normal</u>	<u>DMA64</u>
Memory Type:		
<u>SRAM</u> -->	26.2 MB/s	46.5 MB/s
<u>DRAM</u> -->	21.0 MB/s	37.3 MB/s

Throughputs as a percentage of the Normal-SRAM case:

Transfer type -->	<u>Normal</u>	<u>DMA64</u>
Memory Type:		
<u>SRAM</u> -->	100%	178%
<u>DRAM</u> -->	80%	143%

Use of parallel logic in the node hardware logic design should allow a combination of SRAM speed and DRAM memory capacity, all at DRAM prices, by allowing use of standard fast DRAM in place of SRAM. Currently (in 1997), SRAM costs at least ten times what DRAM costs, is physically larger for a given memory capacity, and takes a great deal more power.

Although these are computed (vice measured) values, current reflective-memory hardware (upon which RTFC is based) does achieve its predicted throughput of 29.5 MB/sec at a signalling rate of 1.20 Gbaud, so these new values will probably prove accurate.

8.3.1 Theoretical Throughput using Large Block-Mode Data Micropackets

A new kind of DMA data micropacket, carrying four 64-bit doublewords of data, was added to RTFC-2002 in October 1997. What is the maximum performance we can expect from this? The following analysis initially ignores the needed block-mode control micropackets (which carry the starting address), and concentrates on the data micropackets.

Each such large micropacket carries four 64-bit doublewords of data plus a 32-bit longword of header plus trailer, for a total length of nine longwords. To this must be added one trailing FC-PH "EOFa" ordered set, also a longword, for a total of ten longwords per "padded large micropacket". After sixteen such padded large micropackets, one extra "IDLE" FC-PH ordered set (a longword) must be added, to allow for transmitter crystal frequency tolerances, yielding a dilution factor of 1/(9*16)= 0.7%.

So, the average length of a padded large micropacket is (10 + 1/16)(32)= 322 bits, which becomes (322)(10/8)= 402.5 code bits per padded large micropacket. This takes (402.5)/(1.0625e9)= 3.7882e-7 seconds per padded large micropacket to send, which implies that 2.640e6 such packets can be sent per second. The throughput is therefore (2.640e6)(4*8)= 84.472e6 bytes per second, or 84.5 MBytes/sec.

The maximum theoretical throughput of RTFC at 1.0625 Gbaud is (1.0625e9)/(10)= 106.25 MBytes/sec, so this represents 84.5/106.25= 0.7950, or 80% of the famed "bit budget", which is as high a fraction as seen in any widely-used LAN or comm-link technology.

For the record, standard Fibre Channel has a reported maximum practical throughput of about 80 Mbytes/sec on 1.0625-Gbaud links.

When the block-mode control micropackets and announcing interrupt micropackets are included, the throughput will drop a bit, depending on our assumptions as to the average transfer length. If we assume that only our traditional 200-byte messages are being sent, the dilution will be one block-mode control micropacket plus one interrupt micropacket for every (200+16)/(4*8)= 6.75, or about seven padded large micropackets sent. The "+16" covers the middleware message header that comes with each message. These messages are actually 208 bytes long, plus the 16-byte message header, for a total of 208+16= 224 bytes, which exactly fills seven large micropackets.

Ordinary padded micropackets are four longwords (three plus the EOFa) in length, while padded long micropackets are ten longwords (nine plus the EOFa) in length, ignoring the extra IDLE after sixteen micropackets. The fraction of the above 80% that delivers message text is therefore 10(7)/(4(1+1)+10(7))= 0.8974, or 90% of the 80%, which is (0.7950)(0.8974)= 0.7135, or about 70% of the link bit budget.

What is the maximum message throughput of the glass, assuming 200-byte messages? Each such message requires seven large micropackets plus either one or two interrupt micropackets. Each of the seven large micropackets takes 3.7882e-7 seconds on the glass, for a total of (7)(3.7882e-7)= 2.6517e-6 seconds per message text and header. The one or two interrupt packets take (4 + 1/16)(32)(10/8)/(1.0625e9)= 1.5294e-7 seconds each, so the total time per message is either (2.6517e-6 + (1)(1.5294e-7))= 2.8047e-6 seconds (for one interrupt packet) or (2.6517e-6 + (2)(1.5294e-7))= 2.9576e-6 seconds (for two interrupt packets per message). These correspond to 1/(2.8047e-6)= 356,547 messages per second and 1/(2.9576e-6)= 338,109 messages per second, respectively.

So, in round numbers, the message capacity of the glass is between 340 thousand and 360 thousand messages per second, or about 350,000 messages per second.

As each message carries 208 bytes of user data, the delivered user throughput on the glass is (208)(350000)= 72.8 MBytes/sec, or about 73 Mbytes/sec of user data.

This somewhat exceeds the capacity of any current-design single node to filter (ignore) passing messages, let alone send or receive the messages. To filter messages at such a rate, the middleware's capture interrupt service routine (ISR) can take no more than 2.8 µS per message (under heavy load) to decide if a message is to be captured. This isn't impossible, although it probably isn't currently (in 1997) achieved.

The "under heavy load" qualification is based on the observation that under sufficient load to keep a node's interrupt FIFOs largely non-empty, we rarely have to pay the full hardware interrupt-fielding time, which is typically a factor larger than the above 2.8 µS. The effect of this is that as the load increases, the efficiency of handling that load improves, a very desirable kind of behavior.

The above message-rate limitation is due to handling interrupts, and therefore does not apply to the use of RTFC as a blackboard.

8.4 Segment Transmission Error Model

In this section, the loss and damage rates are computed for an improbably worst-case segment, one with 33 nodes with all links at maximum length, running at maximum theoretical throughput. Most applications don't go anywhere near that fast, the links are not really that lossy, and the middleware uses 32-bit checksums on messages and critical data structures to detect and re-send lost and/or damaged micropackets.

It's worthwhile to keep in mind during the following discussion that in practice, the dominant cause of packet loss and/or mangling in any system is likely to be the software, not the hardware, by many orders of magnitude, even if the software is bug-free. In practical networks, buffer overflow in endpoints and/or routers is the leading cause of packet loss.

There are two kinds of micropacket error to be handled, both caused by random bit errors: micropacket loss, and wrongful acceptance of damaged micropackets. These are discussed in the following paragraphs.

8.4.1 Micropacket Loss

The following micropacket loss rate analyses are the bounding absolute worst cases, assuming that all links are simultaneously of maximum length (greater than 300 meters for multimode fiber) and yet run at the maximum datarate. In fact, most links are much shorter, the datarate is far from maximum, and the optical bit error rate (BER) is two or three orders of magnitude less than the 10e-12 rate guaranteed by the very conservative Fibre Channel standard. In practice, optical links are either essentially perfect or essentially useless; they either work or they don't work. The actual error rates in practical systems are so low that it's quite difficult to measure them. No other media is better, especially at gigabit rates. In practice, the bit error rate of RTFC hardware will be dominated by the electrical design of the RTFC node and hub cards, and not by the optics.

So, in practice, the observed error rates will be many orders of magnitude less than those computed here. Also note that the middleware has its own 32-bit checksums protecting data and critical structures. A following subsection gives a numerical example.

An RTFC micropacket, containing one 32-bit longword update, is 96 bits long. On transmission, this is expanded to (10/8)(96)= 120 bits. The guaranteed maximum per-link bit error rate is less than or equal to one in 10^12 bits sent; most links are at least a factor of 100 or 1000 better. The maximum probability of a packet error is thus (120)/(10^12)= 1.2e-10. Most (99.9%) such damaged packets are discarded partway around the ring. The observed effect is that some nodes see an update, while some do not, and so continue to use the prior value of the longword that was to be updated.

If a micropacket is lost, it's lost. In a 33-node segment, there are 33 links, and these per-link loss rates are additive, so the net probability of loss is (33)(1.2e-10)= 3.96e-9 of micropackets (each containing one 32-bit data longword) sent. The theoretical maximum throughput of the RTFC at a signaling rate of 1.0625 Gbaud is 26.2 MB/s in single-transfer mode, or 26.2/4= 6.55e6 micropackets per second. At this rate, updates will be lost at a maximum theoretical rate of (6.55e6)(3.96e-9)= 0.0259 longwords per second, or one longword lost every forty seconds (38.6 seconds) at the full hardware throughput.

If one updates each (critical) item twice, each micropacket is sent twice, and the probability of both packets failing to make it all the way around the 33-node ring is (3.96e-9)^2= 1.57e-17. The theoretical maximum RTFC data transfer rate is 26.2/2= 13.1 MB/s in double-update mode, or 13.1/4= 3.275e6 longwords per second. At this rate, updates will be lost at a maximum theoretical rate of (3.275e6)(1.57e-17)= 5.14e-11 micropackets per second, or one lost micropacket every 1.95e10 seconds at full hardware speed. This is once every 618 years of full-rate dual-update operation.

8.4.2 Wrongful acceptance of damaged micropackets

Packets containing bit errors are detected by means of eight parity bits, which are included in the 96 bits of micropacket length, and by detection of 8B/10B code violations. There is therefore a one in 2^8 (or 2^-8= 3.91e-3) probability that a random pattern of databit errors will fool the parity error detection logic, causing a damaged packet to be wrongly accepted as being without error. The quoted “one in 10^12 bits” maximum bit error rate applies to the 10-bit symbols, and not directly to the 8-bit data bytes, and the Fibre Channel FC-0 8B/10B code by itself detects about (1-2^8/2^10)= 3/4 of the bit errors in the 10B symbols, so the net probability of wrongful acceptance of a damaged micropacket is ((96)(10/8)(1/4)/10^12)/(2^8)= 1.17e-13 per micropacket sent per link. In a 33-node segment, there are 33 links, and these per-link loss rates are additive, so the net probability of wrongful acceptance is (33)(1.17e-13)= 3.87e-12 of micropackets sent. The theoretical maximum throughput of the RTFC at a signaling rate of 1.0625 Gbaud is 26.2 MB/s, or 26.2/4= 6.55e6 micropackets per second. At this rate, damaged micropackets will be wrongly accepted at a maximum theoretical rate of (6.55e6) (3.87e-12)= 2.53e-5 micropackets per second, or one every 39,479 seconds at the full hardware speed. This is once every 11 hours at full rate.

8.4.3 Control-micropacket spoofing by damaged micropackets

How often are ordinary data packets transmuted by undetected bit errors into interrupt or roster packets, causing uncommanded control actions? It takes two very specific databit errors to convert a data packet into an interrupt packet, and four very specific databit errors to convert a data packet into a roster packet. The VME-reset and silence-node packets are both specific kinds of interrupt packet. We will analyze only the interrupt-packet case because conversion to roster packets will be much less common.

The probability of conversion of a data packet to an interrupt packet is the product of the probability of getting exactly two undetected databits in error, out of 96 bits available, and the probability that a specific two-error pattern would happen, out of the 4,560 possible patterns of 96 things taken 2 at a time. The probability of getting two undetected bits in error in a packet is approximately (1.17e-13)^2= 1.37e-26 per packet per link. Of this, only one in 4,560 will be changed into an interrupt packet, or one in (1.37e-26)/(4560)= 3.00e-30 of packets sent, per link. In a 33-node ring, this is (33)(3.00e-30)= 9.91e-29 of packets sent around the ring. At the full throughput of 6.55e6 update packets per second, it will be 1/((9.91e-29)(6.55e6))= 1.54e21 seconds between conversions, or one in 50 trillion years.

8.4.4 Message Damage

A message is a collection of related update micropackets. Damage to a message is almost always due to a missing update packet leaving a hole in the message, and is not often due to wrongful acceptance of a damaged packet.

Take a 200-byte (54 longwords including 16-byte header) message being sent as a collection of longword updates, all protected by a common 32-bit longword checksum (included in the header). Each longword travels in its own update micropacket.

Each message, being composed of 54 longwords, has a (54)(1.2e-10)= 6.48e-9 probability of being damaged in a link. In our 33-node example, each such message must traverse 33 such links, and so has a (33)(6.48e-9)= 2.14e-7 probability of being lost in transit somewhere in the ring, so the net probability that at least one of the 54 longwords (including the checksum) is lost is 2.14e-7 per 200-byte message sent.

The maximum rate such messages can be sent is (26.2e6)/(54*4)= 121,296 messages per second, so the probability of message damage is (121296)(2.14e-7)= 0.02594 per second, or one every 38.6 seconds at full rate. This is as expected, because we are assuming that the network is handling the maximum data rate, so the loss rate is independent of the message size, because lost packets are so rare that the rate of damaged messages is identical to the rate of lost packets, regardless of the message size. In short, each lost micropacket damages just one message.

Each message comes with a 32-bit checksum. The probability of a damaged message (one missing at least one update) being accepted as correct by this checksum is 1/(2^32)= 2.33e-10 of damaged messages, so the net rate of acceptance of damaged messages is (0.02594)(2.33e-10)= 6.04e-12 per second at max rate, or one every 1.66e11 seconds, or once every 5,247 years. Definitely, tomorrow's problem. In practice, the middleware simply requests a retransmission of any message whose checksum fails to pass.

8.4.5 Effect of Cut-Through Routing

With cut-through routing, unicast (point to point) updates travel through exactly two links per segment, regardless of the size of the segment. So, in our 33-node example, all loss rates are cut to 2/33= 6.06% of their full-ring rates. Broadcast packet traffic is not helped by cut-through routing, so the loss figures computed above still apply to broadcast packets.

8.4.6 Forward Error Control Considered

ATM uses a one-bit-correcting, two-bit-detecting adaptive BCH code to handle errors in their optical links [dePrycker]. The basic theory is that field tests on multi-kilometer optical links show that 99.64% of the errored seconds contain one error, 0.20% contain two errors, 0.04% contain three errors, and 0.12% contain runs of four or more errors, so if one can correct a single error, one gets a 100:1 improvement in effective error rate, and there is little value in trying to correct more than one bit. So, 53-byte cells containing one error are corrected, while cells containing more than one error are discarded. By contrast, RTFC discards all micropackets containing an error, making no attempt at error correction.

How useful would forward error control (FEC) be in RTFC? There is a suitable 127-bit BCH code carrying up to 120 bits of data and correcting one error, which would easily map onto the current 96-bit micropackets carrying 88 bits of data, so much the same approach as is used in ATM could be used, and would have the theoretical effect of decreasing the link bit error rate from one in 10^12 bits to one in 10^14, thereby increasing all the mean times between micropacket loss or damage computed above by a factor of one hundred.

Software and hardware design foibles are likely to dominate the effective loss rates, not the optical links. And, a factor of one hundred isn't really that much. FEC is of no help during a ring glitch, as all data in transit is dumped. FEC hardware is complex and slow, and may be difficult to implement at gigabit speeds. In RTFC, a better place to spend the resources would therefore be hardware design, to drive the effective loss rates down directly, rather than trying to repair the damage after the fact. In the end, we would still require the 32-bit checksums in normal operation.

8.5 RTFC Flow Control

Why do we need flow control? First, at system startup, caused by power-up of a major part of the ship, everybody will try to talk at once, and they all will have a lot to say. The transient overload could lead to lost messages, retransmissions, more lost messages, etc, slowing or even stalling system startup. Second, most systems don't really have <u>one</u> chief engineer, able to administratively limit aggregate traffic. At best they have one chief engineer per major system, and these systems are added/upgraded incrementally over the years and are not often in sync. Transient overloads are to be expected, and must be handled. Thus, flow control is required, and must be built in right from the beginning.

8.5.1 Flow Control Objectives

What are the objectives of flow control? In an overloaded network, something must give. What will be protected, and what will be allowed to fall to the floor?

In commercial data processing networks, the objective is to gradually throttle the traffic such that the lossless throughput (and thus efficiency of hardware use) is maximized and the network is both stable and fair in spite of gross overload, at the expense of greatly increased latency and especially latency variation. This kind of latency behavior would not be acceptable in a realtime embedded system, where the latency must remain bounded and predictable, even at the expense of everything else except perhaps packet loss rate.

Stability and fairness are formal properties of protocols. A stable protocol is one whose throughput does not decline with increasing offered load; gradual saturation is the usual goal. A fair protocol does not allow any one node to be squeezed to zero throughput when the network is overloaded; the nodes have more-or-less equal potential throughput.

In realtime, to ensure essentially lossless communications with low and predictable latency, as a matter of design we typically expect to use only a fraction of the physical capacity of communications systems hardware, usually less than one-half. In short, we buy predictability with excess capacity.

In RTFC, the objective of flow control is therefore lossless transmission with low and predictable latency, from and to all nodes, in spite of offered loads occasionally exceeding the physical capacity of the network. The overloads may have abrupt onset, but will not last long. Maximizing throughput is less important than controlling latency and preventing packet loss.

This implies that the flow control mechanism must ensure stability under overload, and some reasonable degree of fairness (to prevent squeezeout of some nodes).

Network-wide distributed flow control mechanisms cannot even in theory respond more quickly than two ring tour times; the first to gather utilization data, the second to act on that data, limited largely by the speed of light in fiber. In practice, distributed network flow control algorithms that try to achieve control response times more rapid than perhaps ten ring tour times are likely to prove hopelessly unstable. This is an example of the classic servo-with-transport-delay problem.

Various distributed flow control mechanisms were considered. Credit-based mechanisms don’t work well in RTFC, which is basically a broadcast system. Rate-based mechanisms would work to control average traffic, but cannot react fast enough to prevent transient overloads. Thus, one always requires a fast-acting local flow control mechanism, and the need for the distributed mechanism is thus unclear, as is the practicality.

Because all nodes see the same ring traffic, all nodes have equally good information about the global state of network load, so an instantaneous flow-control algorithm based only on locally available information can work. Addition ally, nodes have information about their own locally-generated contribution to the ring traffic. (The exception is cut-through and/or unicast traffic, which do not circumnavigate the ring, unlike broadcast traffic.)

The ring hardware is designed to handle the full physically-possible ring traffic without accumulation in the node FIFOs. Accumulation therefore happens only when the incoming ring traffic plus the node's own outbound traffic together exceeds the capacity of the outbound fiber. Packet loss can occur only when the accumulation finally exceeds the depth of the node's FIFO. Packet accumulation and subsequent loss is therefore a local problem, and can be observed and solved locally.

The very simple local flow control mechanism mentioned above and described in the next section is known to be sufficient, and no distributed flow control mechanism is required. Observability is the key to this simplicity.

8.5.2 Flow Control Algorithm

By design, each node can carry the full segment traffic as it passes through from optical input to optical outputs, if the associated host adds no traffic of its own. Because each broadcast data packet makes exactly one entire circuit of the segment, the average traffic is the same everywhere on the segment, and this circulating data cannot by itself cause overload and data loss. The whole issue of flow control thus reduces to throttling the traffic added to the passing segment traffic by each node to ensure that the aggregate traffic does not exceed the average capacity of each node's outputs to carry the traffic to the next node without loss. Therefore, each node must monitor its input and output FIFOs, and increasingly stall the associated host or hosts as those FIFOs fill. This is typically done by progressively slowing down or even stopping the RTFC card's end of the VMEbus handshake, and by sending extreme-overrun error interrupts to the host or hosts. The FIFOs are large enough to make random overrun rare if the average traffic is less than about 80% of the segment's theoretical physical capacity, a matter of design of the embedding system.

One widely-used approach also used on RTFC has three levels of flow control: none, VMEbus slowdown, and VMEbus stall. This applies to all forms of transfer via the VMEbus, including DMA. The node transitions from no flow control to slowdown mode if the input FIFO is not empty and/or the output FIFO is more than half full. In slowdown mode, VMEbus cycles are extended until either the slowdown signal disappears, or 3.2 µS have elapsed, whichever comes first. In practice, about 4 µS is the limit. VMEbus stall, where the bus cycle is allowed to time out, is triggered if the input FIFO is more than half full. The stall usually causes a VMEbus error interrupt to the host, which will spend many tens of microseconds responding to that interrupt, unable to talk, allowing RTFC to work the node's backlog off.

There are also proportional VMEbus slowdown schemes, where the bus cycle extension varies more or less linearly with the number of longwords in the FIFO, up to a 4-µS limit, followed by a VMEbus stall, as described above.

8.5.3 Flow control with Cut-Through Routers and Early Packet Discard

The entire basis for the success of the simple flow control algorithm discussed above is the observation that because all packets go exactly once around the ring, measurements of local traffic give an accurate estimate of global traffic. Cut-through routing has the potential to undermine this, because local measurements would no longer necessarily be a reliable proxy for traffic elsewhere. With cut-through routing, unicast (point to point) updates travel through exactly two links per segment, origin to hub, and then hub to target, and do not travel through any other nodes. In RTFC, the effects of this on flow control is largely solved by having the hub send "filler" packets to all other nodes, so that their local measurements of traffic continue to be an accurate estimator of global traffic. Filler packets are sent by a hub to a node, which forwards them back to that hub, which drops them. In a typical system, a packet cutting through a hub spends so little time in the hub that the original packet and all the resulting filler packets emerge from the hub almost simultaneously, travel to the nodes in parallel, and the filler packets "bounce" from the nodes and return to the hub all at once. Because filler packets carry no data, and do not tour the ring, no causality issues arise.

Early Packet Discard is just what it sounds like — unicast packets (but not broadcast packets) are dropped when they have performed their mission, and so are not returned to their node of origin for purging, an optional performance enhancement that also can cause the flow control mechanism at some nodes to be unaware of traffic at other nodes.

In both cases, cut-through routing and early packet discard, the flow-control issue is that because not all packets make the entire tour of the segment, some local measurements of ring traffic will underestimate peak global traffic, and so will allow too much traffic to be injected, causing some other node to be squeezed out, a violation of the desired fairness property. Packet loss will however not occur, because each node can carry the entire ring traffic without loss.

9. Damage Tolerance of RTFC Segments Quantified

This analysis results in a formula that computes the average fraction of nodes dropped from the segment as a function of the average link failure rate and of the number of rings (hubs, really). The present analytical model is split into three parts. The first part treats the behavior of a single representative node as its input and output fiber links are broken. The second part computes the distribution of link faults as a function of the average link failure rate and of the number of rings. The third part combines the first two parts to yield the fraction of nodes that will be dropped from the segment as a function of link failure rate and of the number of rings/hubs. This analysis also works for the non-damage-tolerant, one-ring case.

The most fundamental assumption of the present analysis is that the per-link average failure rate is low, say 10% or less. This allows some significant simplifications at no loss in accuracy for cases of practical interest. Comparison of the present analysis with the results of a computer-based exhaustive enumeration of fault patterns shows that the present simplified analysis in fact perfectly matches the exhaustive enumeration analysis for average link failure rates of 33% or less. With low link failure rates, we need consider only a very simple “segment” containing exactly one node (as multi-node fault patterns are by comparison so improbable). This node is connected to itself by from one to “r” hubs. (This use of a one-node segment to represent the typical node in a much larger segment is an example of a “periodic boundary condition” approximation.) A node has r inputs plus r outputs, for a total of 2r breakable links. If there are less than r faults, the node does not drop out, regardless of the fault pattern. If the probability of a link breaking is 10% or less, as assumed, the probability of having (r+1) faults is no more than 10% of the probability of having r faults, for (r+2) faults it's less than (10%)^2= 1%, and so on. So, the r-fault case dominates the overall behavior, and, in practice, the only significant case is when we have exactly as many faults as we have hubs (rings), which is exactly r faults.

The first part of the analysis therefore treats the behavior of a representative one-node ring with exactly “r” of its r input plus r output fiber links broken, for all possible patterns of r faults in 2r links. A node is “isolated” if no live output connects to a live input in this little one-node ring. A “path” (also called a “dual-fiber cable”) is a link from node to hub plus the link from that hub back to the node. For the path to work, both of its links must work. The essential observation is that the only way for r faults to isolate the node is if no faults are “wasted” by breaking any path twice, which would force some other path to remain unbroken, because we would have run out of faults. So, each path will have exactly one of its two links broken, and there are exactly 2^r ways to do this, out of the total number of ways 2r things can be taken r at a time. So, Pnd[r] = (2^r)/Binomial[2r, r], where Pnd[r] is the fraction of r-fault cases that lead to the node being isolated from the segment.

The second part of the analysis is to find the distribution of number of link faults at a randomly chosen representative node in the segment, given the number of rings “r” and the per-link average link failure rate “p”. A link is either good or bad, and each link is independent of all other links, so the Binomial Distribution is appropriate. Specifically, each node has r inputs plus r outputs, so there are 2r things that could break, and we want the probability that exactly r of them will break, so the Binomial Distribution parameters “n” (number of trials) and “x” (number of “successes”) are set to n = 2r and x = r respectively, where a “success” is defined as a failed link.

So, lfd[r,p] = BinomialDistributionPDF[2r,p,r], where the function lfd[r,p] gives the probability that there are exactly r failures in the 2r links connected to a typical node in an r-ring network, given “p”, the average probability of any link failing, and the arguments of the Binomial Distribution are the number of trials (2r), the probability of a link failing (p), and the number of links required to fail (r), respectively.

Now, we put it all together. The product of the probability of getting exactly r faults and the fraction of the r-fault cases that lead to the node dropping out yields the overall probability of node dropout, as a function of the number of hubs (rings) “r”, and “p” the average probability of link loss. Note that the Binomial coefficients cancel, leaving a fairly simple function: Pnd[p,r] = ((2p)(1-p))^r, where Pnd[p,r] is the probability of node drop, as a function of the number of hubs/rings “r” and the average probability of link failure “p”. This result is essentially exact, for p < 0.33 or so.

For probabilities of 1% or less, an even simpler function may be found, by approximating the (1-p) term as simply “1”, yielding Pnd[p,r] = (2p)^r, which is as simple as it gets. This result is essentially exact, being within 1% for p < 0.01 or so.

For an example, if less than 15.8% of the links are failed, then ((2*0.158)(1-0.158))^4= 0.0050, or 0.5% (or fewer) of the nodes will drop out, which is one-tenth of the 5% node unavailability rate corresponding to 95% availability. The typical segment of an above example contains 33 nodes, so this 16% rate would imply that 42 out of (8)(33)= 264 links total would have to be broken to cause 0.5% of the nodes to drop out.

9.1 How Many Ringlets can a Segment Have?

Although RTFC is designed to avoid the formation of ringlets (isolated rings within a segment), there are some severe fiber break patterns that will cause the segment to partition into some number of ringlets. What is the maximum number of ringlets that can be generated?

If the fibers are carefully pruned such that each of R hubs has its own isolated subset of the available nodes, we will end up with R ringlets, one per hub. To exceed this number of ringlets, at least one hub must support more than one isolated subset of the nodes, so it suffices to determine if it’s possible for a hub to support more than one isolated subset of nodes.

In short, the design of the hub hardware precludes this, for a very simple reason: traffic cannot short-cut across the ring within a hub, so there is no physical way to close two rings within one hub. It doesn’t matter if the traffic is broadcast or cut-through, because all traffic within a hub must circumnavigate at least part of the small copper ring within the hub, and there is no way provided to cut the one ring into two complete rings. What “cut-through” means here is that unicast messages need not visit all nodes; they still must circulate within the hubs.

So, in an R-hub segment, there can be no more than R ringlets. To partition a N-node segment into R ringlets requires a minimum of N*(R-1) fiber breaks, as each node must have all but one output fiber broken to ensure that the node joins the correct ringlet.

How likely is this? Continuing the example of the previous section, in a 33-node 4-hub segment, at least (33)(4-1)= 99 very well chosen output fiber breaks would be required to partition this segment into four ringlets. To partition the same segment into two ringlets, which need not be equal, but each with two hubs, requires (33)(2-1)= 33 equally well chosen fiber breaks.

A tighter upper bound on the number of ringlets may be found as follows [Gordon]. Each hub supports at most one logical ring, as discussed above. Each node supports at most one logical ring, because nodes can listen to at most one input. Each logical ring must either visit or bypass each node, so the number of rings at a node cannot exceed one (ring threads the node) plus one per bypass (rings bypass the node). As all the nodes are connected to each other nose to tail via the hubs, the node with the fewest outputs bypassed will control how many logical rings there can be in the segment. So, the number of ringlets is the minimum of the number of hubs and one plus the number of bypassed outputs on the node with the fewest bypassed outputs.

10. Fault Isolation and Resilience Features of RTFC Segments

In this section, faults are organized first by root cause, and then by means of detection. In practice, software faults are by far the bigger danger than hardware faults, but hardware helps to detect and contain software faults.

10.1 Detection and isolation of faults in the RTFC hardware itself

10.1.1 The first line of defense — built-in power-up diagnostics

The first line of defense against broken hardware is the RTFC hardware's own power-up self-test diagnostics, which are intended to ensure that each node (or hub) is sufficiently healthy to not disrupt a ring into which it inserts itself (or generates, if a hub). These self-test diagnostics include an automatic offline loopback test of all RTFC hardware except the final gigabit analog serial optical transmitters and receivers, thus testing essentially all the logic; if this passes, it is very likely that the RTFC card is healthy. A failed analog serial module cannot spoof the hub or node logic, instead appearing as a failed link, which rostering is specifically designed to handle.

10.1.2 Handling Chattering Inputs

A chattering input, which can be caused by a weak and/or noisy link, in turn due to a cracked or marginal optical fiber, a noisy transmitter or receiver, or the like, will cause continuous rostering and thus will cripple the segment, unless dealt with. Note that ordinary mechanical vibration, typically ten to thirty cycles per second, is one or two orders of magnitude too slow to cause this kind of chattering; the segment would simply roster twice per cycle as the rattling connector periodically opened and closed. The kind of chattering this is intended to handle is typically caused by a marginal optical signal causing the receiver's phaselock loops to dither between locked and unlocked, generating glitch after glitch. Two solutions are provided, one manual, the other automatic. The manual solution is to simply command the RTFC hardware to disable the offending input; this will terminate all input and noise from that input. The built-in automatic solution is that the RTFC card keeps track of the rate at which glitches and/or K30.7 symbols are received at each and every input. If the rate exceeds a hardware-set nominal rate of about one every ten milliseconds (100 Hz), the input is disabled, and must be manually re-enabled.

Hubs lack the ability to have their inputs manually enabled and disabled, so the inputs are instead made fast to disable (milliseconds) and slow to enable (about one second but retriggerable), automatically controlled by the signal-detect function discussed below. In this way, a chattering input will never be enabled.

10.1.2.1 Signal Detect

Changes in segment topology are detected by use of the signal-detect function of the receivers. The fundamental definition of a "glitch", which triggers rostering, is a change in the signal-detect output from the receiver. Both loss and acquisition of signal-detect are considered glitches, and will trigger rostering.

There are three basic approaches that a receiver can use to implement the signal-detect function, which will decide if a good signal is being received: DC, code phaselock, and AC.

In the DC approach, the signal is declared to be present if the average received optical power exceeds some threshold, and is declared absent if the average optical power falls below some lower threshold, the modulation being ignored. The difference between the thresholds (hysteresis) is intended to prevent chattering.

The problem with a DC-coupled signal detect function is that it won't notice that modal distortion in the fiber has wiped all the modulation off the optical signal, leaving only the DC average (plus lots of noise). This effect can be quite significant in long gigabit links, which are generally limited by distortion, not optical flux.

In the code phaselock approach, assertion of the receiver's clock-recovery-unit (CRU) code-phaselock-acquired hardware signal is taken to mean that sufficient optical signal is present.

The problem with the phaselock approach is that a phaselock receiver is quite capable of acquiring code phaselock even when the signal is completely lost in noise. Even if the receiver manages to achieve stable phaselock, the received data will be so noisy as to be useless, and may in fact tease the receiver logic with white noise until it misbehaves, or even locks up, requiring a manual reset.

This problem with the phaselock approach is prevented by declaring the link signal as lost if either phaselock is lost or the optical receiver declares that it is not receiving a sufficiently strong average optical signal. This also prevents locking to internal electromagnetic interference and crosstalk in the absence of a true signal. In short, both sufficient average optical power and full phaselock must be present for the link to declare that it is receiving data.

In the case of RTFC, this DC plus phaselock combined approach would generally work except when somebody tried to use too long a link, or used a poor grade of fiber. A broken fiber cannot cause loss of modulation without also causing loss of light.

In the AC approach, the signal is declared to be present if the average received modulation power exceeds some threshold, and is declared absent if the average modulation power falls below some lower threshold, the difference (hysteresis) being to prevent chattering. The average optical power (used in the DC approach) is irrelevant.

The AC approach is more robust than the other two approaches, or their combination. An AC plus phaselock combined approach would be sufficiently robust to accept only correctly formatted Fibre-Channel signals.

The AC approach is typically implemented as a RF coupling capacitor feeding a fast one-diode or two-diode peak (envelope) detector with RC filter feeding a schmitt trigger, the RC time constant being in the milliseconds. The diodes, which must be able to follow gigahertz signals, must be a small-signal schottky type, and the capacitors must be RF grade (low inductance). A small amplifier or comparitor may be useful. Everything else is ordinary.

AC signal detect could also be used in the intra-hub copper jumpers (coax or twinax) to detect overly long or poorly made cables.

10.1.3 Verification of correct configuration

Given that the hardware works, we must ensure that the RTFC segment is correctly configured. This is done only as a maintenance activity, rather than operationally, as a correctly configured segment will not rearrange itself without human intervention, even in the event of battle damage. Various components may vanish, but the order and topology of the survivors will endure. The procedure to validate the segment is simple. After maintenance activities are done and the segment is brought up and rostering is complete, the RTFC hardware in each node is read by diagnostic software, yielding the node’s own NodeID, the NodeIDs of the up to four abutting upstream neighbors, and if the current node thinks itself the purge node. Diagnostic software then analyzes these readouts and complains if the segment is not correctly configured.

10.1.4 Recourses against misbehaving nodes

The host software, upon detecting that a node and/or its RTFC hardware is misbehaving, has a number of recourses against that node. The RTFC hardware provides two special interrupt packets, one which abruptly disables the specified node’s optical transmitters, silencing it; the other which summarily resets the specified node’s VMEbus backplane, forcing the node to reboot. The reboot process will typically re-enable the disabled transmitters. In practice, one would first silence the jabbering node, then reset it, to further protect the rest of the segment from the final flailing of the resetting node. If the node comes up babbling yet again, even after having been reset, it can be silenced without resetting it, allowing it to babble on harmlessly, in solitude.

A less draconian approach would be to disable one’s own reception of updates from that node, using the 2-D (256-by-256) write-protect filter , as discussed in “Protection of network cache from confused other nodes” (section 10.2.3).

10.1.5 Detection and isolation of faults in data transmission

The RTFC hardware has two basic hardware defenses against hardware-induced data errors. First, the Fibre Channel 8B/10B line code itself detects about three-fourths of the bit errors. Second, the RTFC micropacket is small (96 bits) and yet contains eight parity bits. This is discussed in greater detail in “Segment Transmission Error Model” (section 8.4). The host software, specifically the middleware, adds its own 32-bit checksum protection to all messages, file pages, and critical data structures. This also allows software problems, such as random scribbling in RTFC network cache, a form of distributed shared memory (DSM), to be caught and remedied. In practice, insane software scribbling at random is by far the dominant cause of memory corruption, and checksums covering packets only while on the glass protect the only least dangerous part of the total communications path.

10.1.6 Maintenance Diagnostics Support

The RTFC hardware is designed to allow substantial maintenance diagnostics to be run without disturbing the rest of the nodes on a segment, and without requiring any recabling. This is implemented with a set of isolation switches that can be operated by special host software. These switches allow the RTFC network cache memory diagnostics to be run on just one node in the segment without interference either to or from the other nodes. These switches also allow RTFC to be initialized by the middleware without also initializing the rest of the segment's nodes' network cache memory, thus supporting the middleware in its handling of random entry and exit of nodes.

10.1.6.1 Node enable and disable points

There are four specific node enable and disable points. First, the analog portions of both transmitters and receivers may be enabled and disabled. Second, the two logic paths, one from receiver to network cache memory (Switch A), the other from network cache memory to transmitter (Switch B), may be individually enabled and disabled, thus interrupting traffic between node and the rest of the segment. The path from receiver directly to transmitter is unaffected. Third, the 2-D (256 by 256) write-protect filter (described in the next section) can be set to prevent writing by some or all other nodes to the present node's network cache memory. This filter is in the receiver to network cache path. If only Switch B is opened (disabled), the node becomes listen-only. This is used to record passing traffic, to allow network cache memory to be initialized without disruption of the rest of the segment, or to silence a jabbering or isolated node. If both switches are opened, the node becomes isolated from the segment, allowing network cache memory to be tested, even as the rest of the segment carries on unaffected, with segment traffic passing through the node under test.

The selective enabling and disabling of inputs and outputs can be used to implement diagnostics for the isolation of problems.

10.2 Detection and isolation of faults in the host software

There are two major sources of software faults, the node's own host software (with access to network cache only via the VMEbus), and the software of all other nodes (with access only via the segment). The handling of these two kinds of software fault varies because the varied nature of the two hardware paths.

10.2.1 Watchdog Timer

Simplest and most basic is the watchdog timer in each node's RTFC hardware. If the host software doesn't touch this timer periodically via the VMEbus, the node hardware will automatically withdraw itself from the segment, thus protecting the other nodes from a sick and possibly flailing node.

10.2.2 Protection of network cache from confused host software

Application software on the host is prevented from accessing the RTFC network cache by use of VMEbus address modifier “keys”; only the middleware and operating system are allowed direct access, and the application code is required to do all RTFC accesses via the middleware, enforced by the SBC and RTFC hardware working together. Improper access attempts will cause the offending application to suffer a bus error. This allows RTFC, a common and precious resource, to be protected from crazy application software anywhere in the segment, and the crazy application to immediately be identified.

A similar but parallel protection is to use the host SBC’s memory-management unit to make the RTFC area of VMEbus memory invisible to unauthorized applications code.

10.2.3 Protection of network cache from confused other nodes

Each node has the ability to control which other nodes can write to which part of the current node’s RTFC, called “2-D Filtering”. There are 256 nodes, and network cache is divided into 256 pages, so the granularity of control is 256*256= 65,536 cells. Nodes may be configured to power up with all cells enabled, or disabled, by default. In practice, as a matter of design of the embedding system, each node knows the NodeIDs of those nodes that have some reason and need to write to the current node’s network cache; all others should by default be blocked. The identity of nodes attempting to write to a blocked cell is captured and optionally announced by an interrupt to the target node, allowing the sources of erroneous accesses to be detected and tracked.

10.2.4 Isolation by segmentation

The overall RTFC network is divided into some number of segments, thus further isolating and compartmentalizing problems to the segment of their occurrence. All data passing from one segment to another is handled and verified by the middleware in the intervening bridge nodes.

10.2.5 Listen-only mode

RTFC nodes can be put into a listen-only mode, as discussed in “Node enable and disable points”.

10.2.6 Segment loop-around tests

RTFC nodes can verify the integrity of the segment by performing loop-around tests on the entire segment, including detection of duplicates of the present node’s NodeID. These loop-around tests can be done at any time except during rostering, and do not interfere with other traffic, if done at a reasonable rate.

11. Notes on the RTFC Packet-Handling Flowchart

The purpose of this section is to describe and explain the details of the accompanying multi-page “RTFC Rostering Algorithm Flowchart” (first issued 2 December 1996 and last revised on 27 December 1996). In general, rather than describing the flowchart box by box, the underlying rationale is provided, plus notes on perhaps unobvious details.

Each and every node in an RTFC segment independently and in parallel follows this one flowchart. The nodes are not synchronized or connected in any way, except by the exchange of roster packets via the links and hubs of the segment.

This flowchart actually describes the behavior of a piece of hardware, containing multiple interacting parallel state machines and hardware receiver and transmitter modules, coordinated by the accompanying microcode and firmware. Consequently, the constraints of Structured Programming (such as one entry, one exit) are not necessarily observed. This flowchart does not describe software.

Note that the manual or automatic enabling and disabling of the present node's receivers and transmitters is done at a lower level (closer to the hardware) than anything described in these flowcharts. For simplicity, the fact that inputs and outputs may or may not be enabled is understood, but not explicitly mentioned. The effect of a disabled input or output is to cause the attached link to appear to be broken, a situation which rostering is designed to handle.

The overall "RTFC Rostering Algorithm Flowchart" contains the following items: the "Per-Node and Roster-Packet Data Structures", and the flowcharts "RTFC Packet Handling", "Rostering Process", "Roster Packet Processing", and "Data/Interrupt Packet Handling". They will be described in that order.

11.1 Per-Node and Roster-Packet Data Structures

11.1.1 Roster-packet data

The detailed format of the roster packets is documented in "RTFC Micro-packet Format", RTFC-2002, section titled "Roster Micro-packet". The roster micropacket fields relevant to rostering are the "wrap/purge flag" (called the WrapFlag bit in the flowcharts), the "SenderID" (called the OriginID), the "LastTxID" (called the LTXID or LTX), and the "Node Count" (called the NodeCount). In the flowcharts and description, (data and roster) micropackets are called simply "packets". In the present section, "NodeID" identifies the present node, which is executing the algorithm independently of all other nodes, which are in turn doing the same.

Sometimes, the Node Count is called simply but confusingly the Count; the same name applying to both the field in the roster packet, and to a count kept in each node, differentiated only by context.

11.1.2 Per-Node Data

Each node has its own independent set of these variables.

11.1.2.1 Per-Node Booleans

The Per-Node Booleans (called flags and/or bits) are: T1_Timeout, which is true if the roster-mode T1 timer has expired; false otherwise. ScrubNode, which is true if this node either is or will be the data-mode purge node; false otherwise. WrapInput[1..4], an array of four flags which are true if the corresponding input received a roster packet whose wrap bit was set by the present node; false otherwise. A node with at least one of these WrapInput flags set is called a "wrap node". DataMode, which is true if the present node is in data mode, and false if the present node is in roster mode. Note that nodes enter and exit roster mode independently of one another, and that the entire segment is said to be in roster mode if at least one of the segment's nodes is in roster mode.

11.1.2.2 Per-Node Integers

The Per-Node Integers are: MaxCnt[0..255], an array of 256 unsigned 8-bit integers, indexed by OriginID. The present node's own maximum NodeCount value, derived from its own returned roster packets, is for convenience also kept in this array, in element MaxCnt[NodeID]. NodeID, a network-unique hardwired 8-bit unsigned integer constant identifying the present node. InputSelector, an integer ranging from zero to four, indicating which of the present node's four inputs is either currently selected as or to become the one active data-mode input, or zero if none is selected.

Some smart hubs have the ability to assign "PortID" values to nodes designed to accept such assignments. In this case, the assigned PortIDs take the role of NodeIDs, overriding the actual NodeIDs of the various nodes. In the flowcharts, no distinction is made between NodeIDs and PortIDs, and the term NodeID is used for both. PortIDs are generated in the hubs in hub-port-number order starting with the hub card that is closest to the VMEbus system controller, and working down the VMEbus in daisy-chain-grant order. If all hubs are configured and wired identically, all hubs will assign the same PortIDs to each node. If a node detects a mismatch, it will by design balk, because such a misconfigured segment cannot operate.

11.2 RTFC Packet Handling Flowchart

RTFC Packet Handling is the top-level flowchart. There are three zones on this flowchart, corresponding to the three possible states of a node: roster, transitioning, and data-transfer.

11.2.1 Roster Mode

On power-up or VMEbus reset, the card executes a self-test diagnostic. If this fails, the RTFC card halts, accepting only status and diagnostic requests from the Host, and will not attempt to enter the segment. If this test succeeds, the Rostering Process, described on the flowchart of that name, is invoked, and either succeeds or fails (because a fault was detected). If rostering failed, the node will re-roster, perhaps after a very short delay (a few microseconds) to allow the FC-PH FC-0 hardware to settle, plus some milliseconds to allow the physical link hardware to settle mechanically. When rostering finally succeeds, the node leaves roster mode and enters data-transfer mode via the transition mode. Roster mode will generally be re-entered upon detection of a glitch or receipt of a roster packet while in data mode.

11.2.2 Transition Mode

This mode is very short-lived, and is used to ensure that the node's mode transitions are always correctly signaled to the associated Host, with status flags always updated before issuing the freeze and unfreeze interrupts upon which the middleware depends for synchronization.

11.2.3 Data-Transfer Mode

This mode, usually called simply “data mode”, is the normal operating mode of the RTFC hardware, and endures until a change in segment topology forces entry into rostering mode. In data mode, packets are received and routed according to type. Bad packets (those with bad parity and/or a FC-0 code violation) are simply counted but otherwise ignored. Bad packets cannot be processed further because no bit in the packet is immune to error, so one cannot know what to make of a bad packet. Good packets are either roster packets or data packets (which include interrupt packets). The receiver hardware also detects some exceptions: glitches, K30.7 symbols, and stray roster-mode timeout events (which are simply ignored). Receipt of any of a roster packet, a glitch, or a K30.7 will trigger the node to leave data mode and enter roster mode via transition mode. Data packets are used to update network cache, or to generate interrupts to the Host, or to reset and/or silence the Host, as specified in RTFC-2002 and described in the flowchart “Data/Interrupt Packet Handling”.

11.3 Rostering Process Flowchart

After the customary initializations, the node sends out its own roster packet, and handles incoming roster and data packets. At a node’s entry to rostering, the segment generally contains a mix of data and roster packets. As one cannot know what data did and did not update all nodes before the triggering glitch occurred, all data in transit must be re-sent after the conclusion of rostering in the segment. The generation of orphans must also be prevented. Consequently, all data in transit, be it stored in a node’s FIFOs or still traveling in fiber, is summarily discarded by nodes in rostering mode.

As the nodes one-by-one exit rostering mode and signal the middleware in their Hosts that rostering is complete (by updating a CSR status bit and optionally then sending the unfreeze interrupt), the middleware will start to re-send data presumed lost in transit, and it would be self-defeating if this re-sent data were to be discarded. So, arrival of this re-sent data should instead trigger nodes to return to data mode, but only after rostering is complete.

These conflicting responses to arrival of a data packet are mediated by timer “T1”, a copy of which each node starts upon entry to roster mode. Until T1 expires, an arriving data packet is discarded; afterwards, the arriving data packet instead triggers the node to enter data mode, and this triggering data packet is then handled in data mode. Timer T2, which expires after timer T1, causes the node to unconditionally enter data mode, if it isn’t already there.

Rostering algorithms lacking a T1 timer do not permit nodes to enter rostering mode cleanly, or to exit rostering, as the random arrival of leftover data packets will forcibly return the node to data mode, only to restart rostering mode when another roster packet arrives. The effect is to cause the network to chatter until all stray data packets have been absorbed by their originating nodes. In a heavily-loaded network, sufficient new data traffic is continuously injected into the ring (by nodes periodically returning to data mode) to forever prevent completion of rostering. The solution is for nodes to discard and otherwise ignore incoming data packets for a minimum of one and a maximum of two ring-tour times, thus ensuring that all old data traffic is absorbed and no new traffic is injected, until rostering is truly complete.

The rostering algorithm assumes that the segment’s topology is absolutely constant during rostering. Any evidence that the topology has changed, such as receipt of a glitch, K30.7, or a bad packet anywhere in the segment, will cause rostering to be repeated at the conclusion of the current rostering process. Receipt of some bad packets (those that fail parity) can be tolerated, but even one glitch or K30.7 must cause re-rostering.

Selection of the data-mode "purge node" is critical. There must be exactly one purge (scrubber) node. If there are no scrubbers, orphaned packets will accumulate in the segment’s fiber-optic ring, making some memory locations in network cache impossible to change. If there are multiple scrubbers, packets will be removed before making their full ring tours, making some nodes inaccessible. The issue is that when certain fiber links are bypassed, the rostering algorithm can choose up to four wrap nodes, which is needed in rostering mode but devastating in data mode. The solution is for each node to remember the identity of the input from which the rostering packet that caused that node to decide that it is a wrap node was received. Later, when rostering mode is exited and a final decision is required on which input is to be used, the node concludes that it is the segment’s sole purge node if and only if the node’s chosen data-mode input was also marked as having been used to become a wrap node. It’s the combination of largest NodeCount and reversed NodeID order that, in a ring, is guaranteed to be unique.

If a node never receives any of its own roster packets, the node is clearly either isolated (can neither talk nor listen) or at least semi-isolated (can talk or listen, but not both). In either case, the node is not permitted to speak, to prevent possible interference to the other nodes. This is done by making the node listen-only, but not disabling its transmitters, as the resulting stream of FC-PH IDLEs is harmless. Disabling the transmitters would only provoke another, pointless round of rostering. Upon a subsequent rostering, the node may enable itself and try again, except for transmitters disabled automatically or manually by a lower level of control than the rostering process. These must be manually reenabled. It is unspecified if the enable/disable state is retained over a power cycle.

11.4 Roster Packet Processing Flowchart

This is where the core logic of rostering is handled. The logic, which is quite dense, is based on those properties of the segment that are invariant under the bypassing of nodes by hubs, as discussed in the section “Rostering Requirements”. A decision table, which exactly parallels the code, is provided immediately below. This table was derived by working out the combined logic of orphan elimination, NodeCount incrementing and filtering, etc. There is no easy way to explain this logic. One must simply grind through it case by case, working examples on scrap paper. Mercifully, it’s not large.

NodeID :: OriginID	NodeID ≤ LTXID	WRAP BIT SET?	NodeCnt > MaxCnt	SET Wrap Flag	UPDATE MaxCnt	Forward Roster Packet	Discard Roster Packet
=	x	x	NO	NO	NO	NO	YES
=	x	x	YES	NO	YES	NO	YES
<	NO	x	NO	NO	NO	NO	YES
<	NO	x	YES	NO	YES	YES	NO
<	YES	NO	NO	YES	NO	NO	YES
<	YES	NO	YES	YES	YES	YES	NO
<	YES	YES	x	NO	NO	NO	YES
>	NO	NO	NO	NO	NO	NO	YES
>	NO	NO	YES	NO	YES	YES	NO
>	NO	YES	x	NO	NO	NO	YES

>	YES	x	x	NO	NO	NO	YES

In the above decision table, "x" means "don't care". Conditions are on the left of the heavy line down the middle, and actions are on the right. See "Per-Node and Roster-Packet Data Structures" for the definitions of the variables.

11.5 Data/Interrupt Packet Handling Flowchart

There are two kinds of packet ("Data Micropacket" and "Interrupt Micropacket") and two kinds of target node address (unicast and broadcast) to be handled, as defined in the relevant sections of "RTFC Micro-packet Format" (RTFC-2002). In addition, the 2-D Filtering and Early Packet Discard functions are performed here. Early Packet Discard is just what it sounds like — unicast packets (but not broadcast packets) are dropped when they have performed their mission, and so are not returned to their node of origin for purging, an optional performance enhancement.

Received —> Address Type:	A Data Packet	An Interrupt Packet
Broadcast	If data address passes the 2-D Filter, update Network cache; forward the packet.	Queue the contents of the interrupt packet in the appropriate interrupt FIFO; generate an VMEbus interrupt if FIFO was empty; forward the packet.
Unicast, for me	If data address passes the 2-D Filter, update Network cache; drop the packet if the Early Packet Discard option is enabled; otherwise, forward the packet.	Queue the contents of the interrupt packet in the appropriate interrupt FIFO; generate an VMEbus interrupt if FIFO was empty; drop the packet if the Early Packet Discard option is enabled; otherwise, forward the packet.
Unicast, but **not** for me	Forward the packet without action; do not update Network Cache.	Forward the packet without action; do not add to any interrupt FIFO.

The above table summarizes the flowchart.

12. The annotated original requirements for RTFC

The following are the combined surviving requirements (and notes) for RTFC, generated and discussed in January and February 1996, with the final version dated 20 February 1996. Not all items in this white paper are requirements, and the term "requirements" may be a bit strong, as all were subject to debate and tradeoff by the RTFC Team, more often than not driven by the realities of hardware design and compatibility with some existing designs. Some are rationale, others are observations, and still others are simply advice to the implementor. Editorial comments are in square brackets. The original text has been edited for clarity, and to bring its nomenclature into conformance with the rest of the RTFC documents. Totally obsolete items have been deleted, unless necessary to understand a rejected alternative. These original requirements are provided to give the reader an idea of what problems were being solved when RTFC was developed, and as a guide to implementors.

The following is the original introduction to the then requirements list:

What we want is both a nailed-down set of requirements, suitable as the basis for hardware design, the corresponding rationale; and a top-level design verifying that the requirements are implementable, to the extent that such a committee can determine such things. We also want a list of rejected requirements, and the reasons they were rejected.

It is not required that the going-in list be self-consistent or even practical. One cannot really sort such things out without some resort to attempts at top-level design, to clarify the consequences of the various requirements. That said, it's pointless to do much more than collect a master wish list before the details can be debated and consequences examined. The going-in lists should contain what is known of consequences and implications, both good and bad, so we don't lose track of them and spend time recreating the thinking, reliving the past.

1. Networks with up to 256 nodes must be supported, although 32 to 64 nodes are more likely network sizes, if only for performance reasons. Time was that ten nodes was a large network. Having so many nodes has many implications. The performance of the network, in all aspects, must scale no worse than linearly with the number of nodes in the network. [In pre-RTFC reflective memory implementations, part of reflective memory does scale as N^2, to allow N-way multicast. In RTFC implementations, which carry 24 bits of middleware data in the interrupt packets, the N^2 array is not required.] Overloaded networks must degrade gracefully.

It may well be too much to have all 256 nodes on a single ring, so we may also in the future require some way to bridge between multiple independent rings, forming some kind of store-and-forward ring fabric. This can be handled in the software (middleware). [The current plan is to divide networks up into a set of interconnected segments, using the middleware software and RTFC node hardware to implement the needed bridges.]

2. The new set of on-the-glass packet formats must be good for at least ten years, and must have room for upgrade of both systems and products. A packet format outlives multiple board designs, so there must be ample room and provision for expansion in the design of the formats, and for growth in system sizes and RTFC memory sizes, and for operation in systems containing nodes of varying hardware generation. The node hardware

designs must allow for future upward compatibility, and the packets must be somewhat self-describing in that they must at least identify their own packet type, and perhaps version, so newer stuff knows what to do. Older stuff must pass newer packet types through unchanged, even if the newer packet types are longer than the original packet types. [It turned out to be too difficult to support variable-length packets, but the packet format and hardware design do support mixing of old and new. The hubs can handle variable-length packets, up to some limit like 256 micropackets or 256 longwords.]

3. The units of atomicity in RTFC shall be the naturally-aligned 32-bit longword, 16-bit word, and 8-bit byte. Overlapping updates of different units shall not interfere with each other, or any other unit. SRAM is expensive, and an N-squared array for N=256 is large, especially if everything must be kept in a longword, so the various sizes are needed. [Use of segments allows considerable savings of memory, as two half-size segments together require one half the N^2 memory that one big segment requires. Use of DRAM also helps.]

Although current implementations use SRAM, consideration should be given to possible use of DRAM, [perhaps]with a FIFO up front to handle bursts, as DRAM is a factor of ten cheaper than SRAM. The performance difference is that pure SRAM can handle the full network bandwidth continuously, while the DRAM+FIFO approach can only handle bursts of full network bandwidth. When the FIFO fills, data will be lost. [DRAM versions will be available. It turns out that DRAM designs can be fast enough to match SRAM performance in RTFC applications.]

4. Naval damage and failure scenario: In normal operation, cards and links from time to time fail, and are repaired, so at any time, most of the network works normally. During a fight, the ship may sustain battle damage. No repairs are undertaken during battle. Battles rarely last more than a few hours, so random failures during battle are very unlikely, simply because the battle is short and random failures are rare.

5. Fault tolerance and damage tolerance are not the same thing. Faults (failures) occur infrequently, and are independent in time and place. In Naval systems, (battle) damage most often happens catastrophically, with multiple things failing simultaneously and all in one area. [MIL-STD-2036 describes various kinds of damage.] The "unit of failure" is different from the "unit of damage", as discussed below.

For the following reasons, Byzantine (ie, malicious) faults are not expected and need not be considered in the design of RTFC: Byzantine faults due to battle damage need not be considered, as battle damage is seldom subtle. Likewise, Byzantine faults due to random failures need not be considered, as these failures are so rare that the probability of multiple failures may be neglected as remote, and the system is under continuous repair, so failures cannot accumulate unduly.

6. The unit of damage (vice failure) is the ship compartment. We assume that all equipment and cabling within a damaged compartment are destroyed. Multiple adjacent compartments may be damaged simultaneously, the number depending on the size of the weapon in question. Thus, there is a high probability of multiple simultaneous casualties. In battle, one may lose and regain prime power, and power may chatter. [Power loss and subsequent restoration can appear like battle damage, in that whole areas vanish and reappear, all at once. Power system chatter occurs at a maximum frequency of about 3 Hz. Loose connectors rattle at about 10 Hz to 30 Hz. Vibration testing is done at 16 Hz, as noted below.]

7. The unit of failure (vice damage) shall be the fiber link and the node (containing RTFC card, host SBC, all in a VMEbus crate). [Also, the hub.] By "unit of failure" it is meant that if any component of such a unit fails, the entire unit is considered to have failed. Most failures are due to human handling of the fibers, such as during maintenance, and most such fiber breaks will occur at or near a connector. [Breakage of the ceramic ferrule on ST connectors is common. ST connectors with stainless steel ferrules are quite robust, but they have higher optical loss.]

8. There was an open issue: Is the damage tolerance requirement to survive loss of three fibers, or three (multi-fiber) cables? The notion “cable” is not well-defined, in that it can contain any number of fibers, up to all fibers in the system, and there is no approach that will survive loss of all fibers. If the requirement is to survive three fiber breaks, then some cable breaks could lead to isolated nodes and/or splitting the network into isolated subnets. The RTFC approach satisfies the three-cable-breaks criteria, assuming dual-fiber cables between hub and each node, so the issue is moot.

9. Although we would much prefer the network to seamlessly operate through faults, we can accept short "glitches", where data updates and interrupts are lost at random, just so long as all glitches are announced to all nodes by a special interrupt. Polling a hardware Control and Status Register (CSR) bit isn't sufficient; an interrupt is required, to trigger the refreshing of recently-written areas of RTFC. [The freeze and unfreeze interrupts, plus pollable CSR status bits, allow the middleware to know when the segment is in the process of healing itself. During rostering, writing to RTFC memory does not cause data packets to be sent, so applications can continue during rostering, so long as backlogged data is sent (“refreshed”) after rostering completes. The whole “blink” takes less than one millisecond.]

10. RTFC (and/or shared) memory has no responsibility for Software Safety, except for best-effort transport of messages. [Software Safety is ensured by end-to-end application-level checksums, transaction IDs, sequence numbers, etc.]

11. Support for maintenance of microsecond-accurate synchronized time at all nodes, in spite of ring-tour-time effects, is a goal. Current systems require about one millisecond synchronization accuracy, but future systems will and some current systems do require much better. One approach is to provide hardware support for a very fast and predictable way to exchange NTP (Network Time Protocol) messages between nodes, with messages timestamped immediately upon receipt or upon sending, in the relevant RTFC interrupt handlers. If vectored RTFC interrupts contain sufficient user data, they could be used for such NTP timesync messages. [This has been done, in the RTFC “Microsecond Accurate Tunable Clock” and “Optional Network Time Protocol Micro-packet”, documented in RTFC-2002. The front end of NTP will have to be slightly modified, to account for the fact that the out and back propagation delays between master and slave clock node are not equal (due to the ring topology of the segment), and are many times larger than the desired timesync accuracy. Note that cut-through routing of NTP packets both sharply reduces the propagation delays and makes them equal. NTP currently assumes equal out and back times.]

12. Interrupts shall contain user-supplied data, which data shall be queued along with the interrupt itself, at all receivers. The issue is that in a 256-node system with ten systems and as many system contractors, four interrupt types are simply not enough. There must be a way to send coordinating information around. It would be very nice if 32 bits of user data could be carried, 24 bits would be nice, 16 bits would be OK, and 8 bits would be painfully minimal. A major threshold is that the interrupts should carry sufficient user data to hold a pointer spanning all of RTFC; this currently implies a 28-bit data field. To span

all of the host's memory requires a 32-bit pointer on most modern platforms, and will come to require up to 64 bits in ten year's time. However, aside from some planned full-scale databases, no current software requires more than 32-bit pointers. [Interrupts are still a precious resource, but they now carry 24 bits of user data, and all nodes are expected to use compatible middleware.]

13. Over the life of a ship, 30 or 40 years, the RTFC network will grow. Thus, the ship's RTFC memory network must accommodate a changing mixture of RTFC cards of varying ages and sizes. All manner of partial and complete overlaps must be accommodated, and addition of a new and larger card must not force the upgrade or reconfiguration of existing cards. Otherwise, all systems on the network must be the same, forcing the old stuff to be upgraded when a new system is added, even if the old systems are unchanged. [Contracts typically call for 15-year to 20-year support. The long term plan is to formally standardize the on-glass packet formats, and everything else required to allow multiple interoperable implementations to exist. Memory sizes will increase over the decades, so current hardware must fully decode the data packet’s address field, even if the hardware today cannot today handle anywhere near that much memory, to not preclude the future addition to the segment of hardware carrying much more memory than current hardware permits, without being forced to upgrade uninvolved nodes on that segment.]

14. At any moment, some number of hubs, nodes, and links are out of service. The RTFC network must therefore run normally with much less than a full complement of equipment, and no one item of equipment, active or passive, can be critical to the network. (A piece of equipment may be critical to the larger system using RTFC for communications, but this isn't a problem for RTFC to solve.)

15. In the life of a ship, nodes power up and down in the course of normal maintenance thousands of times, but are damaged or simply fail very rarely. So, the users' perception of network reliability will be shaped largely by the day-to-day handling of orderly exit and entry of nodes, hubs, and links. Thus, orderly exit and entry must be effectively glitchless, and must certainly be lossless, as seen from outside the network. A short "blink", where all network traffic stops for an instant, is acceptable. [RTFC blinks are typically less than one millisecond.] Nodes and hubs must be able to power up and down in any order and combination whatsoever, without any uncontrolled disturbance to operations in the rest of the network. Groups of nodes may well enter and exit in unison, with groups defined both functionally (by system) and also geographically (by ship compartment). The plan is to use the power-fail interrupts to trigger the orderly-exit procedure; this works even if people forget to warn the system by manually taking something offline before powering it down.

16. Nodes must be able to cold-boot their host processors via RTFC, from power-up. [Typically, using the Internet “bootp” protocol.] Therefore, RTFC must be able to come up without any host software assistance.

17. Robustness. The system must automatically recover from reconfiguration mistakes, especially split decisions leading to isolated subnets. [The current RTFC approach will not yield isolated subnets, unless the damage is so severe that practically no other result is physically possible. A more common consequence of a heavily-damaged (shredded) segment is that no network forms, and all surviving nodes are isolated.]

18. Per-node insertion delay must be kept as small as possible, to retain reasonable ring-tour times in 256-node systems. Ring-tour time directly affects how much data is lost in a ring glitch, and how fast reconfiguration can happen, as one ring-tour time is the lower bound on how long it takes for the global state to propagate to all nodes. [In RTFC, ring

tour time is dominated by the speed of packet propagation in fiber, and not by the nodes and hubs.] Note that in an overloaded segment, the ring-tour time may become much larger than normal, by a factor of ten or more, as traffic piles up in the FIFOs. The speed of the segment makes such overloads quite unlikely. [The segment's flow-control provisions prevent data loss. The ring tour time is unaffected by traffic; what varies is how long it takes a given collection of data packets to get to all nodes. These effects are not significant if the average traffic accounts for less than 80% of the segment's theoretical throughput. The embedding system should observe this constraint, to ensure realtime performance. A good design limit is 50% average occupancy.]

19. RTFC must use the same VMEbus [or PCI bus] address for network cache memory in all nodes, to allow the host software [and middleware] to use absolute pointers in its data structures, a great simplification.

20. Jitter must be kept in bounds, especially in large (256-node) systems with most nodes, hubs, and ports bypassed. [In heavily-bypassed segments, one could have any number of links strung end-to-end, allowing jitter to grow without bound. All RTFC nodes and hubs have their own ENDECs and FIFOs, allowing the datastream to be fully regenerated at each step, so jitter will not accumulate. Catalog FIFOs can typically store a few hundred longwords, although a few tens of longwords will suffice.]

21. Orphaned packets must die, and quickly. Orphans are typically created either by undetected bit errors in a packet's NodeID field, or at the ragged edges of a ring glitch, and will accumulate and circulate forever unless actively scrubbed from the ring. In the current approach to recovery from a ring glitch, all packets in transit are deleted, as the software must re-send these packets anyway, thus eliminating the ring-glitch orphans. [Still true. The purge node scrubs orphans in data-transfer mode, while the wrap nodes scrub orphans in roster mode.]

22. Node exits and especially entrances can cause data-update packets and also interrupt packets to be delivered twice. How will this be handled? We can probably simply ignore duplicated data-update packets. Can we also ignore interrupt packets? A small (4 bits?) sequence number field in interrupt packets would solve the problem, or software can be designed to simply ride over multiple deliveries of an interrupt. The probability of multiple delivery is not high, and so the software solution has been chosen. [The chosen solution is to discard and later resend all data that was in transit at the moment that rostering was triggered by node entrance or exit. The interrupt packets carry 24 bits of user data. The middleware is designed to simply ignore multiple receipts of the same interrupt. In fact, the middleware normally sends many interrupts twice, for absolute reliability in the face of lost packets.]

23. In a large, complex system, at any moment, somebody somewhere is fumbling. Thus, there must be partitioning and permissions to control the scope of damage such accidents can cause. As a practical matter, each node must be able to block out all but the players that this node is designed to deal with, without any aid from other nodes. There can be no dependence on some central ship-wide coordination authority, even if such an administrative mechanism exists. In RTFC, two kinds of protection are required, against other nodes, and against software in the present node's host computer. Note that this partitioning is neither required by nor used for software safety, which is instead ensured by use of end-to-end interlocked message protocols.

The host also has its own built-in memory management hardware and doctrine, so the network cache hardware need not attempt to protect network cache from the associated host's software. [This is available. In addition, the ability to completely prevent application access to the RTFC hardware using VMEbus address modifier keys is available, and in being taken advantage of by the middleware.]

Protection against other nodes can be implemented by a 2-D permission bitmap residing in the RTFC hardware of each node. These bitmaps are indexed by NodeID and by network cache memory page number, allowing the network cache space to be finely partitioned. With 256 nodes and say 256 pages, there are 256*256= 65,536 cells. [This has been implemented as the 2-D Filter.]

The default is for a node to power-up with the entire 2-D permission bitmap set to "write not permitted", but with the global bits controlling the node's ability to send and receive packets set to disabled (needed in any case to support power-up diagnostics and the initialization of the node's own network cache memory), thus allowing the node to atomically set its 2-D permission bitmap before attempting to join the network. [This is supported.]

24. Blocked access attempts from other nodes should cause an interrupt in the node detecting the attempt (and reporting who tried to make the access, and what they tried to write and where it was to go), so that miscreants can be caught and eventually punished. This interrupt must be maskable, to allow a node to protect itself against flooding by a crazy node. As the intended use of this detection interrupt is system debugging, it is OK if during floods not all attempted accesses are announced by interrupt, or all data provided. If say as few as 50% of the access attempts were to be logged, we would still have no difficulty finding the miscreant. This is not a trace facility. [The first attempt is latched in the hardware; all others are ignored until the latch is read.]

25. Also very useful during system integration is a local (to the associated node) interrupt announcing that the RTFC is being overrun, that is, that the RTFC hardware cannot keep up with aggregate traffic from both network and the associated host, causing one or more internal RTFC hardware FIFOs to lose data. These interrupts would eventually be recorded in the host's own "flight recorder" for later analysis. [This information is available, in the form of the “Tx FIFO half full” status bit and interrupt. The issue is detection and remedy of accidental pile-ups in large systems, an overall embedding system performance issue. Flow control will prevent data loss. These FIFO-overflow interrupts, or rather, their handling by the host processor, are part of the last-ditch flow-control algorithm.]

26. The RTFC card's own CSRs must be in VME "A24" CSR space, where the CSRs can be protected from host software more easily. A24 is chosen because A16 is too crowded, especially in systems heavy with legacy hardware. Another alternative is to put the CSRs, or at least the various 2-D arrays and bitmaps, in a private address space. [The RTFC hardware CSRs and their location will be documented in RTFC-2004 “Control and Status Registers”, to ensure portability of the middleware over the various implementations of the RTFC hardware. A32 is also supported.]

27. There must be a way for other nodes to forcibly silence a jabbering (babbling) node. This is in addition to the ability to reset a specified node. It's OK if the silenced node can still listen. [Both are implemented, as special kinds of interrupt packet. Local host software can re-enable itself, if it is sufficiently sane to realize that it has been silenced, and also remembers how to reenable itself.] Other useful things to do from afar include generation of a specified VMEbus backplane interrupt. [Not implemented.]

28. Observability. The hardware design must ensure that all nodes (and hubs) can accurately deduce the precise global state of the network, given only locally available data

from the RTFC hardware. Dependence on added software chit-chat is forbidden. Note that as data takes one ring tour to visit all nodes, one must listen for at least this long to ensure that the network's global state is accurately reflected locally. [An absolute requirement for the design of the rostering process and for flow control.]

29. No stale data. Observability shall not depend on memory of events or condition measurements more than one millisecond old, or prior to a glitch. [The segment is re-explored after each glitch. No prior data is used.]

30. Switchover/healing/reconfiguration cannot depend on any pre-fault data. The correct global state, used to direct reconfiguration, must be determined after the fault. This requirement prevents chicken-and-egg and/or reconfiguration synchronization problems.

31. System configuration control. In a large and complex system, getting all the pieces to agree is a job all by itself. The hardware must cooperate by allowing the settings of all straps, jumpers, and settings to be read by the host software, so that mis-configurations can be automatically detected at system startup or node entry time, or by maintenance activities. Mis-configurations generally only occur after maintenance actions, so checking for mis-configurations need not be done very often.

32. The correct operation of the entire RTFC network depends directly upon the uniqueness and correct order of the NodeIDs. Thus, there must be an ironclad and simple way to detect the presence of misordered and/or duplicate NodeIDs directly, and to post the erroneous NodeID, without resort to manual comparison with (hardcopy) configuration data. Given the timely warning that a problem exists, maintenance personnel will then know to study the configuration data to find the specific problem.

Configuration problems happen only during maintenance activities, so it's OK to test for misconfiguration only as a test required to return to operational modes. However, the node hardware must provide the information required for a maintenance diagnostic to automatically determine that the network is in fact correctly configured, or to make specific complaints about misconfigured parts of the network. [Duplicate NodeIDs can be detected with a loop-around test, as the other node of the same NodeID will scrub the test packet before it can return to its sender. The RTFC hardware makes available sufficient connectivity data for diagnostic software to determine if the segment is correctly configured.]

33. The network must be observable from a SNMP-compliant COTS Network Management product. This implies that some or all nodes will harbor a SNMP Client and associated MIB (Management Information Base, a specialized small database), and that the RTFC hardware must make available the data needed to fill that MIB. [The needed SNMP Agent will be provided by the middleware. The RTFC hardware does make the needed information available.]

34. There needs to be a ring-around test mode, where a node can verify that it can both talk and listen, without affecting any other node. Recall that nodes enter and exit at random, and one cannot put the entire ship in test mode to test some random node. Likewise, the RTFC card itself must be able to do local (electrical, not optical) looparound tests without requiring any fiber path changes, and without disturbing the rest of the network. There must be a RTFC hardware mode that allows both initial fill of RTFC during node power-up, and also local memory diagnostics to be run, all without disruption of other nodes. The initial-fill issue is that after power-up, the RTFC memory contains random garbage, so a node must initialize all of its own network cache memory before commencing normal operation. However, we don't want the other nodes' network cache memory to also be

initialized, so there must be a way to prevent a RTFC node from sending update packets, and yet allow packets from other nodes to pass through. Probably, the RTFC hardware should power up in this bypassed mode, with writing to the segment also disabled, and only switch into operational mode at specific host software command. [All implemented.]

A major purpose of these power-up tests is for each node (and hub port) to verify that it is sane enough that insertion will not bring the entire network down. The combination of testing node and hub cards to the 1-Gbit serial electrical level, one step shy of the optical interface, plus using Fiber Channel IDLE patterns or the like as evidence of correct link operation during operational use, provides a largely chinkless testing approach, because the test domains overlap. A node or hub port that fails this power-up self-test is required to balk, refusing normal (non-maintenance) commands to insert itself into the network.

35. In a 256-node system, provisions must be made to allow maintenance of parts of the system without extended disturbance to other parts, which continue normal operation. One must test in place, without disturbing other nodes, or rearranging the fiber optic cable plant.

36. RTFC Fiber Optic Network (FON) Analyzer. For system debugging, it would be useful if one could configure a RTFC card to take all (or selected by a NodeID bitmap) passing network cache traffic and DMA it (in plain uninterpreted on-glass format) into the RTFC card's own memory in order of arrival, perhaps with timestamps. This FON Analyzer mode must be able to handle packets of any type and length, not being restricted to those packet types and lengths current when a specific card was manufactured. [The intended uses are system debugging, and analysis of performance and traffic loads.]

Note that there are two levels of RTFC FON Analyzer implementation, full-capability and best-effort. The idea is to build a best-effort capability into all RTFC node cards, where "best-effort" effectively means "insignificant added cost given the RTFC card design", and to in addition build special-purpose full-capability RTFC FON Analyzer cards. [It proved too difficult to build this FON Analyzer mode into the standard RTFC hardware. The FON Analyzer will become an independent piece of hardware.]

Software could then later decode this trace of RTFC activity into humanly-readable form. (In the ethernet/fddi world, one can buy software-only "LAN Analyzers" that work by putting the ethernet or FDDI MAC into "promiscuous" mode and feeding the resulting stream of packets to the software.)

37. One must be able to put a RTFC card in a passive listen-only mode, to support data extraction. [Supported.] This is not the same thing as the " RTFC FON Analyzer".

38. How are marginal links to be detected at sea? What is the definition of "marginal"? Intermittent? What of errors and noisy links? Arguments claiming that this newfangled fiber-optic stuff will not have marginal and/or noisy links are not likely to be successful in the Navy community, until some years of fleet experience accumulates. For detecting marginal links during maintenance activities, on-board analog fiber-optic received-power (and analog ground) test points can be provided, with access to a voltmeter probe only when the RTFC card is on a bus extender. [There isn't sufficient slack in the fiber optic cables for board extenders to work, so the test points need to be little wire-wrap pins, so wires can be brought out as needed. The analog test points should be protected from ESD and excessive load with 10,000-ohm series resistors and little zener diodes in parallel. Analog ground-reference test points are also needed.] Maintenance personnel can be given a test procedure with specific voltage limits: if voltage is less than some value, declare the link marginal. Modal and chromatic dispersion, which, for a given operating wavelength and fiber type, depend only on fiber length, do not grow much with age, unlike connector

loss, and so do not need to be tested at sea. Connectors can become dirty, requiring cleaning. [Modern optical transmitters do not age very much, but fibers can still become cracked and connectors can still become dirty.]

39. How do we determine the health of currently-inactive links? For SNMP, we need to keep error counts, for reporting, to trigger maintenance actions. Error counts need not be kept in RTFC hardware, as the error rates are expected to be quite low, and most often zero, unless the link is broken. In practice, the bit error rate of RTFC will be dominated by the electrical design of the RTFC node and hub cards, and not by the optics. [All enabled inputs are monitored, active or not.]

40. It must be possible to manually disable and re-enable selected links, ports, nodes, rings, and hubs. This is used to bypass a marginal link until it can be repaired a month or two later. [Implemented. Disabled inputs tell no lies.]

41. For flexibility, it must be possible to operate without a hub, although some or all fault and damage tolerance may be lost. Hubless configurations could use optical relays, which are fine for shore-based installations. [Implemented.] Note that there is no such thing as electrical “ground” on a ship, and the EMI environment is very severe. [There can be several volts between ground points located in different ship compartments, in spite of heavy shields and ground braids connecting those points. Beware of ground loops.]

42. The links between nodes and hubs may be as long as 300 meters (984 feet). [Using fddi-grade 62.5-micron multimode fiber.] Full specified RTFC performance (speed, update loss rate, and wrongful-acceptance rate) must be achieved even if all links are 300 meters long. The required fiber optical cable type is FDDI-grade 62.5-micron graded-index multimode shipboard cable. ST fiber-optic connectors are required. [The operating wavelength is 850 nanometers, with operation at 1300 nanometers an extra-cost option. Using 1300-nm single-mode fiber, one can go as far as 30,000 meters, using only catalog optical components.]

43. Transmitter control. To prevent disruption of the network, all nodes must follow the following rules: A node with all inputs inactive ("dark") must respond by turning all of its own optical transmitters off, thus forcing the associated hub ports to bypass the node. In addition, a node having only one active input must turn one optical transmitter off (but not the transmitter corresponding to that one active input), thus forcing at least one hub port to bypass the node. The purpose of this turn-one-off rule is to prevent a rare variety of node hardware problem from disrupting the network. The purpose of the added rule that the transmitter corresponding to the one active receiver cannot be the one turned off is to ensure that we don't disconnect the node from its last surviving cable, under the assumption that there are dual-fiber (duplex) cables from each node to each of the associated hub ports, and that the fibers within these duplex cables are likely to fail together, such as when the cable is cut.

44. The RTFC hardware (RTFC cards and hub cards both) must operate continuously over a temperature range of +10 to +50 degrees Celsius, must permit storage over a temperature range of -25 to +70 degrees Celsius, and a non-operating temperature range of 0 to 50 degrees Celsius, and survive levels of shock and vibration typical of heavy-industrial electronics. Typical shock specifications are 30 to 35 gravities perpendicular to the plane of the board. Typical vibration specifications are 1.5 gravities at 16 Hz, in three mutually orthogonal directions. The RTFC hardware must meet all specifications while exposed to a relative humidity range of 0% to 95% non-condensing both intermittently and continuously. Note that the zero-percent humidity condition implies a severe electro-static discharge (ESD) environment. Although no explicit storage altitude range is mentioned,

hardware must survive shipping in military and commercial transport aircraft, not all of which are pressurized. The above requirements are largely based on the Navy-standard Q-70 Console specs, flowed down to the boards within, although all testing is performed at the box level, and not on the boards individually. [Note that these mechanical requirements can be met by use of good industrial-duty design rules, including use of metal hardware (such as component holddowns, board separator standoffs, screws, etc), and soldering most components down. Surface-mount components generally pass without any difficulty. Sometimes, large components such as capacitors must be glued or strapped down, and large IC packages must be strapped down to prevent them from bouncing out of their sockets. Anyway, these specs are far from full military specs. However, be aware that vibration that causes working (relative motion of pin and socket) of tin plated connectors can cause fretting corrosion and subsequent unreliability of connection. Gold plated contacts do not have this problem. Connectors require positive latches or screwdowns to prevent accidental disconnections caused by vibration and unrelated maintenance activities tugging on cables.]

45. The RTFC hardware (RTFC cards and hub cards both) shall be a single VME card, occupying no more than two VME slots. There shall be no jumpers, although a NodeID-selection switch is OK. The design shall accommodate a spectrum of designs ranging from one to four rings. Operation at a wavelength of 850 nanometers shall be the default, with either 1300 nanometers or 1500 nanometers available as special-order options. [The requirement for the 1500-nm option, used only in telecommunications (vice datacomms) applications, has been dropped.]

13. References

[Carr]	"Refractive Index Measurements on Single-Mode Fiber as Functions of Product Parameters, Tensile Stress, and Temperature", J.J. Carr et al, *Fiber and Integrated Optics*, 1990, Volume 9, pages 393-396.
[Corning]	"Corning 62.5/125 CPC6 Multimode Optical Fiber" datasheet, PI360, first issued 10/94, which is current as of August 1997.
[CRC]	"CRC Handbook of Chemistry and Physics", 58th edition, CRC Press 1977.
[dePrycker]	"Asynchronous Transfer Mode", third edition, M. de Prycker, Prentice-Hall 1995.
[FC-PH]	"Fibre Channel — Physical and Signalling Interface (FC-PH)", ANSI X3.230-1994. RTFC uses layer FC-0 (physical) and FC-1 (code), but does not use layers FC-2 and above.
[Gordon]	Karen Gordon, private communication, 9 December 1997.
[Liu]	"A Study of Ring Networks", M.T. Liu and D.M. Rouse, in *Ring Technology Local Area Networks*, I.N. Dallas and E.B. Spratt (editors), Elsevier, IFIP 1984.
[SCI]	"Scalable Coherent Interface (SCI)", IEEE Std 1596-1992.