TNet: A Reliable System Area Network

A major departure from traditional I/O systems, TNet is a new system area network designed to support current and future needs for reliable, efficient communications among processors and peripherals. It is an extensible hardware-software layer that allows very large configurations by logically and physically isolating processor buses from I/O buses. TNet features wormhole routing, packet-switched transfers, and point-to-point links.

Robert W. Horst

Tandem Computers Incorporated

The rapid pace of CPU performance enhancements has far exceeded performance advances in I/O architecture. As a result, systems cannot take full advantage of available processing power. To close the gap, new system designs must transition from being processor centric to I/O centric. The TNet (Trusted Network) architecture results from taking a fresh look at the requirements for supporting scalable I/O systems in future generations of CPUs.

TNet is not a direct replacement for any existing type of local area network or I/O bus. It is a new interconnection layer, a system area network (SAN), which provides common hardware and software services to processor and I/O nodes. By connecting all devices through this common I/O layer, the devices inherit TNet features that allow on-line service of peripherals; isolation and containment of faults; a common I/O programming model; and common I/O configuration and error management. TNet links extend within a system cabinet through backplane connections, or between cabinets through external cables. This flexibility allows systems with hundreds of processors and thousands of I/O devices.

The TNet SAN also provides inter-CPU communications for distributed memory multicomputing. The links are logically similar to LAN connections, but the low latency in both hardware and software means that multiple-CPU systems behave much more like massively parallel processing systems than LAN-connected clusters. TNet interprocessor communication (IPC) capability can be used for single-system-image parallel processors, such as the Tandem NonStop systems,[1] as well as other loosely coupled processor clusters.

Designers typically structure existing multiple processor clusters around a group of workstations connected through a high-speed LAN, as shown in Figure 1 (next page). This design's main problem is that hardware and software latencies may be so large that the system devotes a high percentage of CPU cycles to servicing the LAN.

To send a message between processors, hardware must arbitrate for two buses on each system and may have to copy data several times at each end. Software requires several context switches and interrupts to set up and perform the transfer. Latency for a single message can grow to many milliseconds.

The TNet cluster (Figure 2) directly addresses latency problems. TNet provides routing and addressing to allow any CPU to communicate with any other CPU or I/O controller in the network. TNet nodes can read and write portions of each other's memories without requiring software execution at the remote node. The cluster uses TNet addresses as virtual I/O addresses for automatic scatter-gather capability when accessing

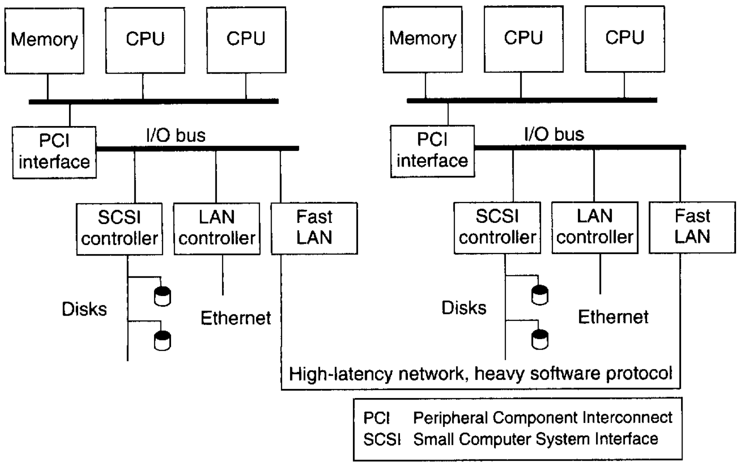


Figure 1. CPU cluster connected with a typical high-speed LAN.

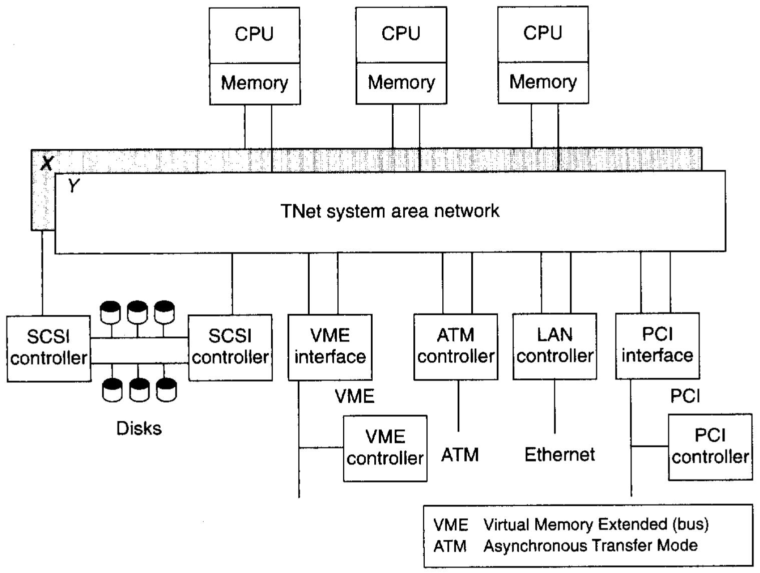


Figure 2. CPU cluster connected with the TNet system area network.

By connecting all I/O devices and CPUs through the same interconnection layer, hardware can support direct connections between CPUs, from any CPU to any I/O device, or between I/O devices. This capability supports either shared-disk or shared-nothing database systems.[2] In a shared-nothing database, such as Tandem NonStop SQL, the networked disk connections are beneficial in allowing load balancing by reassigning disk ownership without requiring disk connection recabling.

Direct peripheral-to-peripheral communication gives TNet-based systems unique capabilities in multimedia applications. A CPU may set up a large transfer between communications and disk controllers, but the data can flow directly between the controllers without passing through main memory. This improves the performance of the transfer, conserves memory bandwidth, and reduces CPU cycles.

TNet supports dual paths to every node through a pair of independent *X* and *Y* subnetworks, making the network fully fault tolerant. In normal operation, TNet spreads traffic across the two networks, but either one can take over the load after a failure. TNet-based systems never pass packets between networks, preventing errors or congestion on one network from affecting the other.

Designing TNet

The architecture of Figure 2 requires the network implementation to have certain characteristics: scalability, high performance, low cost, and reliability. It was natural to look for an existing standard bus or network to fit this architecture, but we found nothing that met all the requirements. This forced us into the difficult job of designing a completely new network.

consecutive virtual addresses, which are scattered in physical memory. Translating TNet addresses to physical addresses also protects main memory from errant I/O devices. The CPU must set permission bits in the TNet address translation table permitting access to a portion of memory.

Scalability is a critical requirement for commercial systems to provide smooth growth for customers as their computing requirements grow. The need for growth dictated a network instead of a bus, because a network can provide additional system bandwidth for each added node. Point-to-point links are also beneficial in containing and isolating hardware and software faults, and networks do not have the severe length restrictions of shared buses.

High performance is always an important consideration in designing system interconnections. Performance analysis must consider the link bandwidth, communications latency, and software overhead. Software overhead often dominates performance, and hardware must facilitate short software path lengths wherever possible.[3] In commercial systems, short messages and I/O transfers are common, and the architecture must not penalize short message latency even if a long transfer is currently in progress. Designing for short latency imposes requirements such as short packet sizes and efficient interrupt handling.

The new interconnection network must be simple and of low cost to be incorporated into all CPU and I/O components of the system. The interface logic must be simple enough to allow several network interfaces on a single application-specific integrated circuit. This leaves enough room for the other logic the ASIC must perform. Also, interface costs must not preclude the attachment of low-cost peripherals.

An interconnection network for systems with hundreds of nodes or more must have built-in reliability. Errors must be detected as they happen, and the network must continue operation in spite of failures. The existing literature on fault-tolerant interconnection networks often fails to consider faults encountered in real implementations, such as clocking and power faults. In addition, many subtle fault modes in the addressing and control logic cannot be modeled by simply considering link failures. For instance, the network must prevent faults from causing misaddressed packets and deadlocks.

A network also must handle the stale packet problem. When an error occurs, packets in transit may be stuck somewhere in the network. If there is no way to remove these stale packets, they can show up later and disrupt operation after the system recovers from the original problem.

During our investigation, we considered many existing and proposed standard networks. Fibre channel[4] has very good performance on long sequential transfers, but its latency on short packet-switched (class 2) transfers could not meet our requirements. Asynchronous Transfer Mode (ATM)[5] was far too costly, given the large number of network attachments required in a system area network. The scalable coherent interface (SCI)[6] had a design center for handling cache coherence traffic, and as a result its cost and complexity exceeded our acceptable limits. Designed as an I/O bus replacement, the P1394 serial link's[7] performance and scalability do not meet high-end system requirements.

Although none of these networks were viable for the system area network itself, SAN-connected controllers may-support them to satisfy open-interconnection requirements.

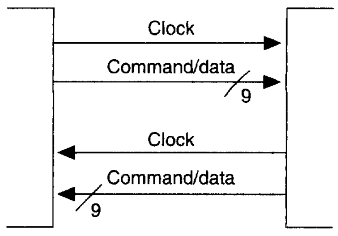


Figure 3. TNet byte-serial data links.

Table 1. Command and data symbol encoding.

CD8	CD7	CD6	Function
0	0	0	Command
0	0	1	Error
0	1	0	Error
1	0	0	Error
0	1	1	Data <7:6> = 00
1	0	1	Data <7:6> = 01
1	1	0	Data <7:6> = 10
1	1	1	Data <7:6> = 11

TNet implementation

In the end, the only reasonable choice was to design a new network to satisfy the unique requirements of the SAN. The requirements dictated a wormhole-routed, packet-switched, and point-to-point network designed with special attention to reducing latency and assuring reliability.

Physical layer. Figure 3 illustrates the physical connections of each TNet link. The links have independent transmit and receive channels, each with a nine-bit command/data (CD) field plus a clock. The CD field provides 256 data symbols plus up to 20 command symbols. The links use command symbols for link-level flow control as well as for initialization and error signaling. Encoding commands and data into the same lines reduces pin count and improves control logic fault detection.

Table 1 shows the 8B/9B code for symbol encoding. The code uses three bits to distinguish between a command and four groups of data symbols. The four data symbol groups, plus the remaining six bits, encode the 256 data symbols. A different code in the upper three CD bits selects the command symbols.

A 3-of-6 code (in which all valid symbols have exactly three 1s and three 0s) over the remaining six CD bits represents the commands. This code detects all unidirectional errors as well as any odd number of errors in the lower six CD bits. The three error codes separate the data symbols from the command symbols by a minimum Hamming distance of two. This prevents any single fault from turning a command symbol into a data symbol, or vice versa.

TNet links use source clocking, with the clock sent through similar delay paths as the data. This makes it easier to scale the system to higher frequencies, and avoids the performance losses of asynchronous networks that slow down as handshake signals become longer.

Figure 4 shows the TNet port implementation. Intercabinet links use differential emitter-coupled logic drivers to drive up to 20 meters of cable. Single-ended CMOS buffers drive local backplane TNet connections. Each TNet interface uses a pair of FIFO buffers for clock synchronization and flow control. The synchronizing FIFO buffer takes in CD symbols from the source clock and sends them out synchronously to the receiver's clock.

TNet port logic uses the elastic FIFO buffer to queue data symbols and support link-level flow control. Congestion in the network may limit the retrieval rate of data symbols from the elastic FIFO buffer. When the FIFO buffer begins to fill, it sends out a signal instructing flow control logic to return a Busy command symbol to the sender. When the sender receives the Busy symbol, it stops sending data symbols and instead sends Fill symbols (that are discarded by the receiver) until it receives a Ready symbol.

When a long cable connects the sender and receiver, there can be a significant time lag between the time the receiver FIFO buffer begins to fill, and the time the sender actually stops sending data symbols. This time lag dictates the minimum size of the elastic FIFO buffer. In the first implementation, a 20-meter cable transferring at 50 Mbytes/second requires less than 32 bytes of elastic FIFO buffer, including internal pipeline delays. The elastic FIFO buffers have at least 64 bytes of storage to satisfy this requirement, as well as to provide extra storage for improved link usage.

Future adapters may extend TNet distances through serial fiber optics. This conversion is relatively straightforward because all control functions, including flow control and interrupts, communicate through the same CD symbols.

Packets and transactions. Figure 5 shows the TNet packet formats. Packets consist of an 8-byte header, optional 32-bit address, variable-size data payload, and a cyclic redundancy check. The header specifies destination and source node identification, data length, and which operation to perform. Operations include Read Request, Read Response, Write Request, Write Response, and Unacknowledged Write. A single packet's maximum payload is 64-bytes to reduce worst-case latencies. Restricting the packet length

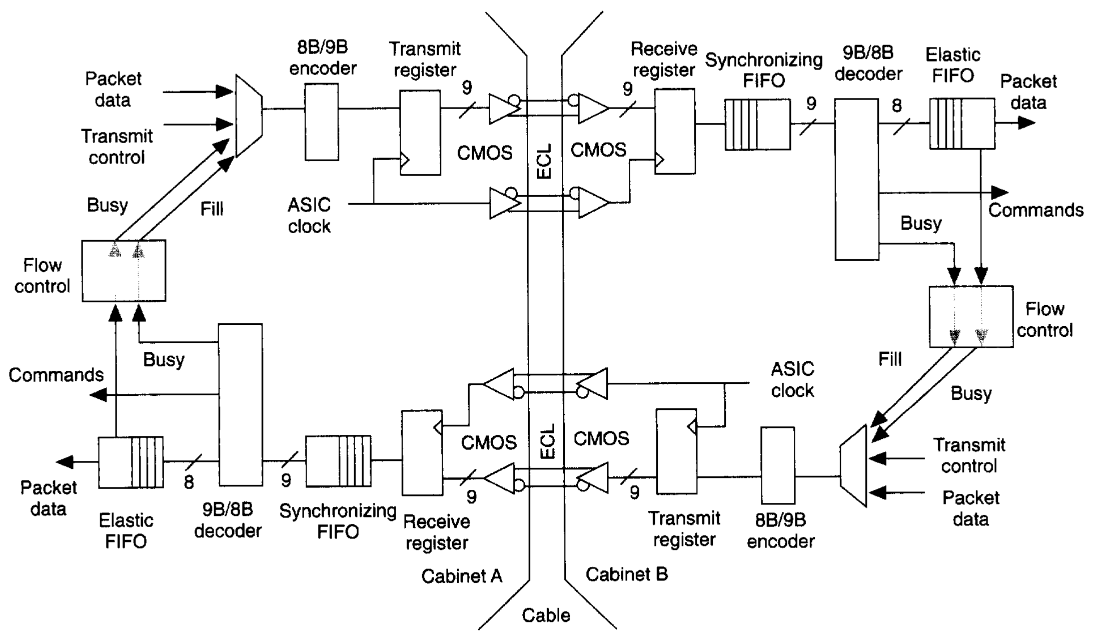
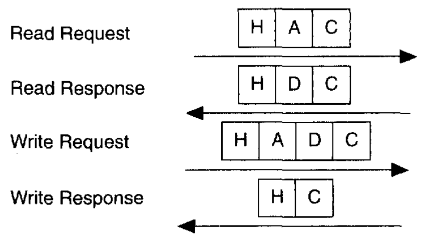


Figure 4. TNet port implementation and flow control.



H	8-byte header
	20-bit destination ID
	20-bit source ID
	4-bit transaction type
	8-bit data length
	4-bit transaction ID
	1-bit primary/alternate path
	1-bit ack/nack
	1-bit request/response
	5 bits reserved
A	4-byte TNet address
D	0-64-byte data
C	4-byte CRC of H; and A and D if the packet has A and D fields

Figure 5. TNet read and write transactions.

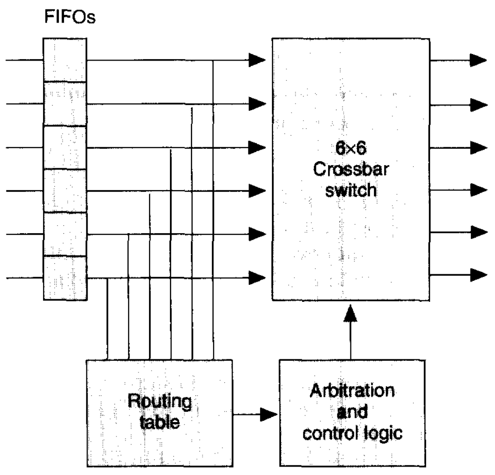


Figure 6. TNet router.

also improves fairness in link usage, and reduces storage requirements.

The TNet address is a 32-bit window into the destination's address space. The interface to the destination CPU or I/O device provides mapping and validation hardware translating this virtual TNet address to the appropriate physical memory address. Typically, I/O devices use one-to-one mapping, but because TNet node identifications differ for each I/O bus, each I/O node has its own independent 32-bit address space. CPUs have an address validation and translation table (AVT) to map the virtual TNet address space to the processor's physical address space.

Performing both read and write functions allows a node to either push or pull data across the network. The pull capability, which is missing from most LANs, reduces the number of handshakes required for buffer management. When reading data from a remote node, the pull capability eliminates the context switch and software execution otherwise required at the remote node. The support of both read and write functions also allows I/O device traffic to be forwarded to host memory; hardware thus handles the TNet transfers transparently.

Packet routing. Figure 6 shows a block diagram of a TNet router ASIC. The first generation uses 6x6-port routers and makes routing decisions using a programmable routing table. The table selects an output port based only on the destination identification of the incoming packet. Routers have FIFO buffers on the inputs, logic for arbitration and flow control, a routing table implemented in RAM, and a crossbar switch. The service processor can modify routing tables to configure or reconfigure the network.

TNet uses wormhole routing to reduce latency to 300 ns or less. As the first bytes of a packet arrive at a router, it uses a portion of the destination identification to address the routing table. The table specifies the output port number of the packet's destination. If that port is busy, or if the input port loses arbitration, additional bytes of the packet continue to accumulate in the elastic FIFO buffer. Link-level flow control prevents packet loss despite momentary congestion in the network.

In addition to link-level flow control, end-to-end flow control prevents nodes from injecting more packets into the network than can be handled efficiently. This is accomplished by requiring a positive acknowledgment for every packet sent. (The Unacknowledged Write is an exception. However, it is only used for maintenance subsystem functions and in special situations where higher level protocols support recovery of dropped packets.) Some nodes support multiple outstanding requests allowing multiple packets in transit to a destination before acknowledging the first one. In all cases, nodes must guarantee enough buffer space to accept the maximum outstanding packets from all possible sources. End-to-end flow control prevents saturating the network to the point where added requests could actually decrease network throughput (similar to virtual memory thrashing). CPU nodes use main memory to buffer incoming packets, effectively giving unlimited buffer storage. I/O node interfaces

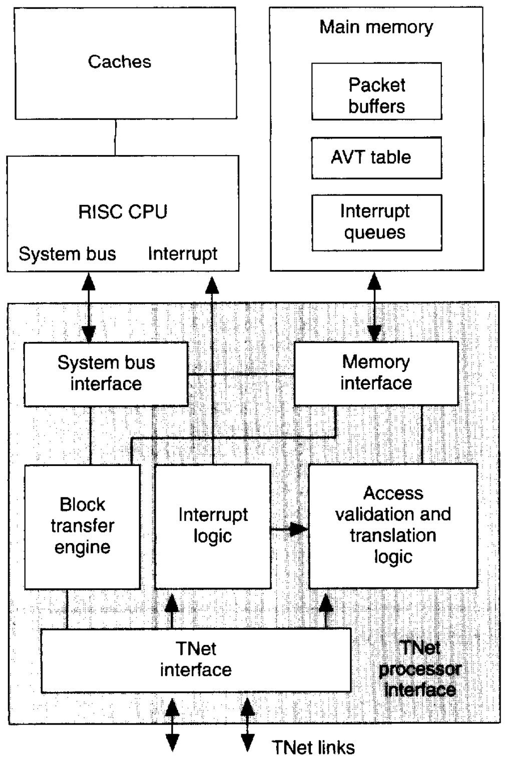


Figure 7. TNet processor interface.

have a small number of hardware packet buffers; a controlling CPU manages their use.

Deadlock avoidance. In wormhole-routed networks, congestion can force a packet to wait until the router sends the packets ahead of it. If the network is configured with loops in the connection graph, congestion may block several packets, with each packet waiting for a link currently used by another blocked packet. A reliable network must either avoid this type of deadlock, or detect and break deadlocks when they occur.

TNets avoid deadlocks by disallowing configurations with circular dependencies and requiring end nodes to accept incoming packets even if their outbound links are busy. Many topologies, such as tree networks, need no modification because they are inherently cycle free. Selectively disabling certain paths in the router will support topologies with cycles.

We have not included virtual channels[8,9] as a way to avoid deadlocks because the small benefit in configuration flexibility did not justify the added complexity and additional buffer space required. Disabling selected paths does not unduly burden the anticipated topologies for TNet networks, and yields design area substantially smaller than a virtual channels design.

Processor interface. Figure 7 shows a diagram of the TNet processor interface (TPI) ASIC. This ASIC provides connections between the CPU and memory, and between the TNet ports and memory. The TPI also performs CPU initialization and controls interrupt queuing and delivery.

The TNet system implementation allocates a portion of main memory for use by the processor interface to reduce the amount of required on-chip memory. Main memory contains packet buffers, the large AVT table, and interrupt queues.

When an inbound read or write request packet arrives at the processor interface, the interface must validate and translate the virtual TNet address by looking it up in the AVT table. Each AVT entry includes the physical address translation, permission bits indicating whether the CPU permits read or write access, and the identification of the TNet node allowed to use this entry. A small hardware cache for AVT entries eliminates memory AVT table access for many inbound packets. Some programmers at first objected to setting up AVT entries for every transfer. However, they soon grew to appreciate the automatic scatter-gather capability that this structure provides. In many cases, this capability can eliminate an entire data copy and the associated latency.

I/O devices generate interrupts by sending standard write packets to CPU addresses that the AVT table translates into interrupt requests. When an interrupt packet arrives, a special AVT entry directs the packet to one of four priority interrupt queues in main memory.

The interrupt packet includes the node number of the device requesting the interrupt, and may contain other interrupt status information. This interrupt payload eliminates the need for several more round-trip transfers between the CPU and I/O controller to acquire status.

To support efficient message passing, the processor interface ASIC has a block transfer engine (BTE) that the processor can program to send short or long messages through a sequence of chained TNet packet transfers. Programmers can prebuild BTE descriptors in main memory and initiate them by writing to a hardware register in the interface.

The TNet implementation does not currently support memory-mapped I/O because we thought it was the wrong model for coping with increasing I/O latency. As CPU speeds increase, both due to faster cycle times and more instruction issues per cycle, each noncached load can easily waste tens to hundreds of instruction cycles even when accessing a "local" I/O bus that may in reality be many microseconds away. The processor interface instead assumes a DMA model where the processor can accomplish useful work while waiting for transfers to complete. Even for short transfers, the added overhead required to program the BTE is small compared to the time when the processor would otherwise stall.

Bus interface. Figure 8 shows a block diagram of the TNet bus interface (TBI) ASIC. The TBI translates transfers on the standard bus into TNet read or write transactions that travel through TNet links to main memory.

Different versions of the bus interface logic support industry standard buses such as VME and the peripheral component interconnect (PCI), or microprocessor buses such as that of the Motorola 68040 chip.

Each TBI chip supports an independent I/O bus. Each system may have many TBIs to support several identical or different I/O buses. Each standard bus typically has from one to four attached peripheral controllers. TNet links eliminate the length and loading restrictions that would occur if all devices connected to the same bus. The individual buses also have much better fault isolation and containment.

Another benefit of this structure is that most standard buses are not split-transaction buses; therefore, once a device requests a memory read, it can initiate no other transactions until the read data returns. By implementing multiple I/O buses (each controlled by a TBI), many reads can simultaneously wait for data to be returned.

I/O programming exploits the DMA capabilities of new peripheral I/O chips. The programming model assumes that the speed of the I/O device itself controls and paces most input and output processing. For inbound data (such as reading from disk), the I/O controller generates TNet write transactions to place the data directly in the requesting CPU's memory. In the outbound direction, the I/O controller requests data using a TNet Read request. The CPU node then responds with the data as part of a Read-Response packet. In both directions, processor-interface hardware handles the transfers between the TNet and main memory without involving the CPU. All devices can simultaneously have DMA transfers in progress without burdening the CPU with supporting a multithreaded DMA controller.

Topologies. Users can configure TNet routers in many topologies including hypercubes, meshes, and trees. System clusters will generally use different topologies in different places. For instance, it may be advantageous to use a tree for I/O connectivity, but connect the I/O trees and CPUs together in a hypercube.

The requirements of a network used to construct large clusters of processors and peripherals may change over time. During a system's lifetime, users may add many devices and add or remove links to help balance network use. We intended to design a network that was not restricted to a fixed topology, but could grow and adapt as needed. Our design's flexible, six-port router ASIC allows use of many different networks to satisfy system requirements.

Error detection, isolation, and recovery. One of the primary distinguishing characteristics of the TNet design is its focus on reliability. No I/O system can work reliably if errors go undetected or if the system cannot isolate the cause of the errors. Point-to-point TNet links make fault isolation much easier than in a shared-bus I/O system.

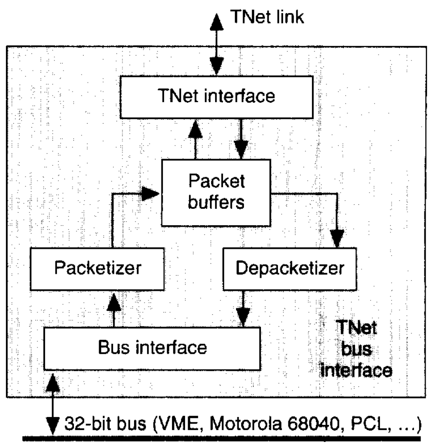


Figure 8. TNet bus interface.

Error checkers detect single-bit errors on command or data bits either through the packet CRC or through command symbol encoding. These codes also detect multiple-bit burst errors. The CRC covers all header bytes, providing protection to address as well as data errors. The CRC gives end-to-end protection of packets, because the routers do not modify or regenerate a packet's CRC as it passes through.

Checking the CRC code at each router crossing enhances fault isolation. The TNet interface appends a command symbol to every packet. Normally the appended symbol is "this packet good," indicating that CRC was good when it left the previous routing level. If a TNet interface detects faulty CRC anywhere along the path, it changes the symbol to "this packet bad." The maintenance subsystem uses this information to isolate exactly which link or router stage introduced the error, even if the error was transient and nonrepeatable.

The AVT table also plays an important role in assuring that faulty I/O controllers do not corrupt memory. AVT entries are programmed to grant access rights for a range of memory pages or for as little as a single byte. This fault containment isolates problems to a single controller instead of the entire I/O system. This is especially important in I/O, because the system vendor has little control over the hardware and software quality of third-party peripheral suppliers.

Fault recovery is the responsibility of higher level software. Software retries I/O errors, and may retry them through an alternate path if available. Systems designed with multi-

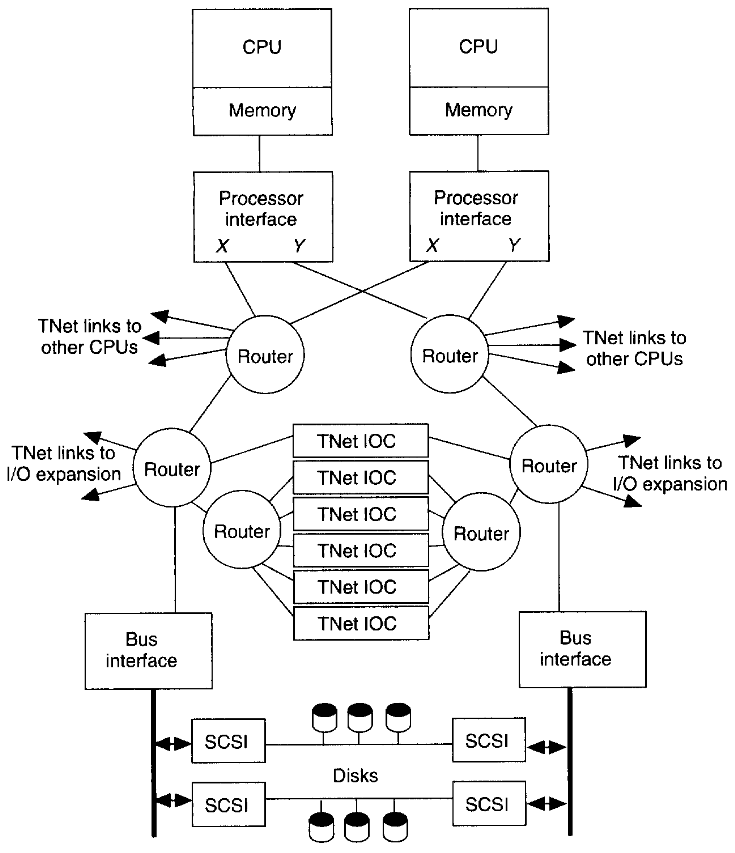


Figure 9. Typical system architecture using TNet.

ple subnetworks provide complete fault tolerance in the I/O system. TNet I/O adapter cards optionally connect to two independent networks allowing continued operation despite the loss of one entire network.

Bounded worst-case latencies. A major design goal was to provide worst-case latencies below a reasonable value (a few milliseconds). This is important for fault-tolerant systems because it allows designers to set low values for timeout counters and thus reduce fault recovery time. Any network with unbounded latencies will occasionally hit an error and cause some form of hardware or software retry. As such a network becomes congested, the chance for those retries increases, and then the retries themselves increase network traffic even more. This can cause a runaway situation that is extremely difficult to model or prove bounded.

We designed TNet explicitly to avoid delays that have no provable upper bounds. TNet does not use asynchronous handshake signals, as distinguishing between faulty links and slow handshakes can be difficult. It also does not use adaptive routing because many of those algorithms can loop indefinitely looking for possible paths. In addition, we designed routers with duplicated and compared state machines to prevent erratic or erroneous operation.

System architecture

Figure 9 shows a typical system architecture that can be implemented using a TNet SAN. Dual-ported processor interfaces connect to redundant *X* and *Y* subnetworks. Spare TNet links can connect many CPUs. The system provides I/O controller (IOC) slots for storage, communications, or TNet router expansion.

I/O expansion is provided by an unbalanced tree connecting local TNet I/O controller slots and TNet cable connections to remote peripheral cabinets. Dual-port connections to the I/O slots assure connectivity during failures or reconfigurations of one network. Storage interfaces provide dual paths to the disks and allow disk mirroring across independent Small Computer System Interface (SCSI) buses.

THE TNET SYSTEM AREA NETWORK is a major departure from traditional I/O systems, yet supports standard I/O buses and high-level programming models. It is noteworthy not only for its wide range of features but also for features that were consciously left out for simplicity and low cost.

Simplicity allows low-cost ASICs with multiple TNet ports for configuration flexibility and fault tolerance. We designed the network without memory-mapped I/O, virtual channels, hardware retry, or adaptive routing to lower costs without sacrificing our primary goals.

TNet represents a new layer of hardware architecture. This layer solves the problems of I/O extensibility and allows a uniform programming model optimized for intelligent peripheral controllers. The network supports message passing within CPU clusters, as well as CPU-I/O and even I/O-

I/O transfers. The emphasis on fault detection, isolation, and repair allows TNet to provide a solid foundation for building reliable systems.

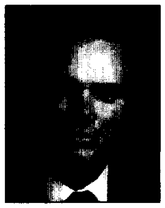
TNet is currently the basis for several future systems development projects. We are continuing to evolve the architecture and programming model for use in different applications. The flexible network topology gives us many options in building parallel systems, and we have begun research projects to study the best ways to optimize large TNet configurations. We expect to build many future generations of TNet-based systems.

Acknowledgments

Many talented engineers codeveloped the TNet architecture, and I thank all the team members for their creativity and hard work. In particular, special thanks go to Dave Garcia, David Sonnier, and William Watson who made key contributions to the architecture, and to Bill Baker, Bill Bunton, Rich Cutts, Dan Fowler, John Krause, and Stephen Low who were instrumental in developing the first TNet components and systems.

References

1. J. Bartlett et al., "Fault Tolerance in Tandem Computer Systems," *Reliable Computer Systems*, D.P. Siewiorek and R.S. Swarz, eds., Digital Press, Bedford, Mass., 1992, pp. 586-648.
2. D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High-Performance Database Systems," *Comm. ACM*, Vol. 35, No. 6, June 1992, pp. 85-98.
3. V. Karamcheti and A.A. Chien, "Do Faster Routers Imply Faster Communication?," *Parallel Computer Routing and Communications*, K. Bolding and L. Snyder, eds., Springer-Verlag, New York, 1994, pp. 1-15.
4. T.M. Anderson and R.S. Cornelius, "High-Performance Switching with Fibre Channel," *Digest of Papers Compcon 1992*, IEEE Computer Society Press, Los Alamitos, Calif., 1992, pp. 261-268.
5. D.J. Greaves et al., "Design and Implementation of an ATM Backbone Ring," *Digest of Papers Compcon 1992*, CS Press, 1992, pp. 255-260.
6. *Scalable Coherent Interface (SCI)*, ANSI and IEEE Std 1596-1992, IEEE, Piscataway, N.J., 1992.
7. M. Teener, "A Bus on a Diet—The Serial Bus Alternative: An Introduction to the P1394 High Performance Bus," *Digest of Papers Compcon 1992*, CS Press, 1992, pp. 316-321.
8. W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553.
9. W.J. Dally, "Virtual-Channel Flow Control," *Proc. 17th Ann. Symp. Computer Architecture*, CS Press, 1990, pp. 60-68.



Robert Horst is a technical director at Tandem Labs where he has contributed to the architecture and design of five generations of fault-tolerant parallel computer systems. His technical interests include interconnection networks, computer architecture, fault-tolerant computing, and wafer scale integration.

Horst received a BS in electrical engineering from Bradley University, Peoria, Illinois, and an MS in electrical engineering and PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE Computer Society and the ACM, and holds 20 US patents.

Readers may reach the author at Tandem Computers Inc., 10555 Ridgeview Court, LOC 100-27, Cupertino, CA 95014; horst_bob@tandem.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162 Medium 163 High 164