The Kiewit Network: A Large AppleTalk Internetwork

Richard E. Brown
Dartmouth College, Kiewit Computer Center, Hanover, NH 03755

Abstract: Dartmouth College's Kiewit Network connects nearly all of the computing resources on the campus: mainframes, minicomputers, personal computers, terminals, printers, and file servers. It is a large internetwork, based on the AppleTalk protocols. There are currently over 2900 AppleTalk outlets in 44 zones on campus. Over 90 minicomputers act as bridges between 177 AppleTalk twisted pair busses. This paper describes the extent and facilities of the current network; the extensions made to the AppleTalk protocols, including a stream protocol and an asynchronous link protocol; and current development projects, including an AppleTalk stream to TCP converter.

The Kiewit Network is a general purpose local data network. It serves as glue between all the host computers and nearly all the personal computers and terminals on campus. The network extends to the academic and administrative buildings, and the student residence halls, with network ports placed in almost all the offices and rooms. From these ports, users may access any of the host computers connected to the network. In addition, the hosts use network connections to communicate between themselves for electronic mail, file transfer, printer sharing, etc. The network also supports Apple Computer's AppleTalk® protocols. Departments and individuals can purchase standard devices which use AppleTalk (such as LaserWriters® and file servers), and connect them directly to the network.

The Kiewit Network is not part of the telephone, energy control, or security systems of the College. While there can be benefits from combining them, separating these systems has simplified our design in many ways. A network failure doesn't interrupt these other services. We can also schedule experimental time for testing new software at our convenience.

Dartmouth College started its campus network with General Electric Datanet 30 computers acting as front ends for the Dartmouth College Time Sharing (DCTS) system. In 1978, as the requirement for ports grew, Honeywell 716 minicomputers replaced the Datanet machines. In the same time period, two factors forced a change in perspective away from a front-end function toward a networked approach. New hosts began to arrive on campus. These new machines required enhancements to the network. Second, off-campus sites demanded more ports than the existing phone lines could supply. We served these remote users by using Prime P200 minicomputers as statistical multiplexors. In 1980, we implemented the DCTS terminal handling protocols on a less expensive minicomputer, and began to place these network nodes in the basements of buildings across the campus.

When the College recommended that the freshmen entering in the fall of 1984 purchase a Macintosh™ computer, funds were appropriated for three major network enhancements: installation of a network port for each undergraduate in the residence halls; conversion of the network to the AppleTalk protocols; and creation of a terminal emulation program to exploit the enhanced network capabilities.

Computers at Dartmouth

Dartmouth supports a variety of computing environments which include the Dartmouth College Time Sharing system (DCTS), Unix® (4.2 and 4.3 bsd), VAX/VMS, IBM VM/CMS, and Primos operating systems. The host computers which support these include two Honeywell DPS-8/49's, one VAX 8500, eight VAX 11/785 and 11/750's, six microVAX'es, one IBM 4381-1, a Convex C1 XP, and a Prime 650. The network also serves Sun and Xerox workstations, many IBM RT PC's, and the Telenet public data network.

Several services are accessible from the network without logging in: the library's on-line card catalog, a computer help system, the campus events calendar, and several student jobs databases.

Owners of Macintosh computers use DarTerminal, a locally-developed terminal emulation program, to connect to any of the host computers on campus. DarTerminal supports multiple simultaneous sessions, each in its own window; cut-and-paste transfer between sessions; VT100 emulation; TEK 4012 graphics; the ability to transfer arbitrary Macintosh documents to and from hosts; and a distributed screen editor called MacAvatar®. This screen editor works with a back-end editor on a host computer to give a Macintosh-style editing interface for each of the hosts. This gives us essentially the same screen editor on four of the popular operating systems on campus: Macintosh, DCTS, Unix, and VAX/VMS.

There are about 4000 Macintosh computers on campus. Over half are owned by students who live in residence halls. About 800 IBM PCs of various models use asynchronous ports to connect to the network. There are over 1200 known RS-232 terminals, including CRT's, dot-matrix printers, and letter-quality printers.

Why choose AppleTalk?

A number of factors led to our choice of AppleTalk as the base protocol within the Kiewit Network: the difficulty of maintaining the existing network; the desire to use higher bandwidth links; the existence of a coaxial cable around the campus; the low cost of an AppleTalk connection; and the College's decision to recommend Macintosh computers to the students.

In early 1984, the Kiewit Network comprised about 40 minicomputers which acted as packet switches and terminal concentrators. The network served about 1500 terminal ports (at 2400 bps) with a normal load of 200-250 simultaneous terminal sessions. Internally, the network was based on X.25 level 2 data links, with the DCTS terminal handling protocols as a transport layer. The links between buildings were 19.2 kbps local area data circuits.

The network met our user's demands, but was quite hard to maintain. The major problem was that all routing information had to be hard-configured: a network engineer had to enter an explicit path from each of the network hub nodes to each host computer. Furthermore, we had to balance traffic flows through the network manually, by reconfiguring these paths. As the number of hosts increased, it was becoming difficult to manage and configure the network.

In 1978, as part of a major project to install cable TV wire around the campus, Kiewit had several runs of coaxial cable laid to the major buildings on campus. We believed that the best way to exploit the coax was to have a datagram-oriented network. We had considered stripping down TCP and IP to make a protocol with adequate performance across the medium speed links on and off campus.

The decision, in 1984, to recommend that freshmen purchase personal computers gave us access to real networking, with enough computing power to support reasonable protocols. After studying Apple's network plans, we saw that AppleTalk gave adequate internetwork facilities, yet was simple enough to be reliable and implementable. In addition, AppleTalk was considerably more efficient than the other contender, the Internet protocols.

AppleTalk requires only a $50 connector to access the network. By using relatively inexpensive bridges, AppleTalk is very cost competitive with other networking schemes, even RS-232 async terminal ports.

All these reasons convinced us that AppleTalk would be a reasonable alternative. AppleTalk

offered low-cost connections, with the dynamic configuration that we desired. We had great hope that it would succeed in the market, so that third party products could enhance the value of the campus network. Even if it failed, though, we were sure that AppleTalk would work here at Dartmouth. We began a major redesign of the network in May 1984. When the freshmen arrived in September, the network was providing AppleTalk services in the residence halls.

Inside the Network

The Kiewit Network is a large AppleTalk internet. The network nodes act as AppleTalk bridges, providing routing, naming and zone services. In addition, the bridges act as terminal servers and as controllers for networked line printers using an AppleTalk stream protocol. Any device which uses the Apple protocols can be connected to one of the 230.4 kbps ports and communicate with other AppleTalk devices anywhere on campus. As commercial products (networked printers, file servers, electronic mail, etc) have come to market, our users have a much wider choice of products than we could have developed internally.

We currently have 95 bridges active in the network. These support 82 AppleTalk busses (at 230.4 kbps). We give fictitious network numbers to the terminal servers, and thus have a total of 177 networks, divided over 44 zones. Typically, one zone serves a single building.

The network supports the full set of AppleTalk protocols including AppleTalk Link Access Protocol (ALAP), Datagram Delivery Protocol (DDP), Routing Table Maintenance Protocol (RTMP), Name Binding Protocol (NBP), and Zone Information Protocol (ZIP). ALAP sends packets between devices on a single bus. DDP is responsible for routing packets between devices on separate AppleTalk busses. RTMP defines how the network nodes keep track of the best route to all other nodes. NBP and ZIP define how a name of a service is converted to a (numeric) network address.

The Kiewit Network also supports the X.25 protocol, connecting us to host computers and public data networks. This X.25 package has been certified by Telenet, Tymnet, and Uninet. The X.25 also provides terminal connections and file transfers to our VMS, Unix, and VM/CMS hosts.

In addition to the standard Apple-defined protocols, we designed and implemented two enhancements. An Async AppleTalk Link Access Protocol can replace the standard (230.4 kbps) ALAP. Async AppleTalk allows a Mac to connect into an AppleTalk network over an async (RS-232) link, even on a 1200 baud dial-up line. Async AppleTalk has been implemented as a Macintosh driver. We use Async AppleTalk for printing to LaserWriters, file serving, electronic mail, terminal access, and several of the public domain AppleTalk packages. Currently, Async AppleTalk only works with the terminal ports of the Kiewit Network; commercial products which convert between the async and 230.4 kbps links are beginning to come to market.

We also designed a data stream protocol (DSP) which we use for terminal access to host computers (see The Data Stream Protocol, below).

Network Hardware

The packet switching computers of the network are 16 bit minicomputers, manufactured by New England Digital (NED) of White River Junction, VT. These nodes act as terminal concentrators, and perform packet switching and resource naming. Nodes contain up to 128 Kb of RAM and a variety of interface boards.

All nodes contain a small ROM bootstrap. The program in the ROM establishes a connection to a central reload server and requests a reload. This ROM bootstrap loads in a larger RAM bootstrap program, which continues the reload procedure.

Each processor contains a 10 second watch dog timer which is reset by the software's main loop. If the software or hardware reaches an error condition and fails to reset the watch dog timer, the processor transfers into the ROM bootstrap.

Each node computer can support a combination of up to 56 asynchronous (RS-232) ports, or up to 12 HDLC interfaces which use X.25 (to 56

kbps) or the 230.4 kbps AppleTalk ALAP. We connect to our Honeywell hosts through disk channels on the I/O multiplexor. The NED nodes also support Centronics and Data Products printer interfaces.

Network Topology

The nodes of the network are connected to form a tree, rooted in the reloading machine located in the computing services building. There are multiple interconnections between the nodes within the computer center. Notable among these links is a cross-tree AppleTalk bus which joins a dozen nodes in the building.

These central nodes fan-out to other nodes which in turn connect to leaf nodes in the basements of buildings around campus. These leaf nodes contain asynchronous ports, AppleTalk interface boards, and printer interfaces.

Links between nodes are either medium speed synchronous (HDLC) lines or standard 230.4 kbps AppleTalk busses. The links between buildings are generally 19.2 kbps LADS (local area data set) circuits.

Within buildings, we place a four-prong telephone outlet (the old style square jack, not a modular jack) in each room or office and run four-conductor shielded station wire from the room to a central point on each floor. Additionally, we run a vertical backbone of 50-pair cable from the network node (generally in the basement) to each floor.

This wiring scheme has the advantage that it handles either asynchronous terminals with RS-232 signaling or AppleTalk devices with an RS-422 driver. We select asynchronous or AppleTalk connections simply by cross-wiring the station wire to the appropriate pair(s) in the backbone cable.

In the residence halls, we have one outlet per bed, for a total of 2600 outlets. In the academic and administrative buildings, there are more than 1650 active asynchronous terminal ports. Currently, a project to convert these RS-232 outlets to AppleTalk has installed about 300 ports.

The Master Node

A single network node (the "master node") watches over the network. It runs standard network software with additional functions, and never carries production traffic; it only monitors the network and reloads nodes which have failed.

The master node is at the top of the network tree. It is the only machine which has mass storage. Its disks hold reload images and configuration information for all the nodes in the network. When a failure occurs, the master node receives a copy of the failed memory image (a "dump"), sends a fresh copy of the program and configuration information to the failed node, and forwards a copy of the dump to the DCTS mainframe for examination. To track down the cause of the failure, a program on DCTS displays a stack trace, identifies and formats packets in a human-readable form, and makes several automatic checks on the binary image.

Configuration and Startup

Configuration information for the network is stored in a (text) file on the master node. The configuration can be changed by editing the file and reloading the affected node (we cannot currently perform on-the-fly configuration). To reload a node, a technician either presses a front panel switch or forces a crash from a privileged terminal. Forcing a crash is simple: the node turns off interrupts, and enters a tight loop. The watch dog timer and ROM bootstrap do the rest. A front panel switch on the remote node, or on the master node, can force the master node to skip the dump procedure, saving time and storage.

New revisions of the software are installed the same way. We update the binary image of the program on the master node's disk, and then force a reload in the particular node. We can reload the entire network by asking the master node to pass a "death packet" to its children. Each recipient passes the packet to its children and then begins its reload procedure. Thus the reload request spreads outward through the network, and it reloads from the center. One node takes about two minutes to load (with a dump); the entire network reloads (without dumps) in 15 minutes.

Master Node Monitoring

The master node polls each of the other nodes, and prepares a display (called the "status display") which shows software response times, the round trip time from the master to the other node, number of connections in a node, errors on a link, and other performance statistics. Any terminal on the network may connect to the master and receive this display.

The status display has several views on the network's state. The default screen shows errors – nodes or links which are in an unusual state. The terminal bell sounds whenever an entity changes to a worse state. A technician can silence the bell by acknowledging the problem from a network privileged terminal; typically, the technician enters information about the trouble, with their actions and initials.

The status display can also show information about the entire network, a specific node or link, or a certain class of link.

Troubleshooting Capabilities

We have invested heavily in the control and monitoring functions for the network. In the standard network software, between 10 and 15 per cent of the total lines of code are devoted to testing and diagnostics.

Each network node has a "control process" which accepts connections (with password protection) from terminals. This process provides a user interface to the monitoring functions in the machine. Support personnel can connect to the control process and request information about the relevant aspects of a node's performance. This includes link information (bytes and packets per second, errors per minute), node information (free storage, per cent busy, per cent interrupt processing), routing tables, the current configuration, terminal port activity, the state of active connections, etc.

Each code module within the node has calls to routines which can produce error messages with detailed information about interesting events. Examples are CRC errors on a link, checksum errors, routing table changes, protocol violations, etc. The various protocol layers can produce lines containing relevant protocol information. A support person can cause these lines to be formatted and displayed by setting the appropriate monitoring flag. We are careful to keep monitoring flags turned off most of the time, since formatting the data is computationally expensive and often increases the node's response time.

Each node maintains a connection to one of our DCTS host computers for archival purposes. There, a log file for each node continually receives information about unusual events from the control process. We perform daily maintenance on these files, watching for files which have grown too much or which show unusual occurrences.

The Data Stream Protocol

In the summer of 1984, we designed and implemented a data stream protocol (DSP) as part of our DarTerminal terminal emulation program. We chose to design a new protocol, rather than use the industry-standard TCP family, for these reasons:

- At the time, none of our host computers used TCP (terminal connections were primarily over X.25 links); we would have had to integrate TCP/IP as well as the AppleTalk protocols into the existing network.
- There was no IP support within a Macintosh at the time.
- The links of the network were slow. Most of the links between buildings were medium speed (19.2 kbps) local area data circuits, we had several links to off-campus sites running at 2400 baud. With the overhead of TCP and IP headers, these links would have given poor service for many applications (most notably host character echoes).

The original design used a two-byte header. This has served well for the last three years. In the summer of 1987, we converted to a four-byte header because: a) The two-byte header had an eight-bit packet sequence number. This was too small to prevent wraparound over fast links. The new protocol has 16 bits in the se-

quence number; b) A cooperative effort with a third-party AppleTalk vendor required additional features that the two-byte protocol could not support. This commercial implementation will allow our users to use any Macintosh terminal emulator across an AppleTalk network. The rest of this document describes the four-byte header.

The DSP is a client of the AppleTalk Datagram Delivery Protocol (DDP) and Name Binding Protocol (NBP). A client may use DSP as its only interface to the AppleTalk network. The DSP establishes a connection between two entities on an AppleTalk network. The client may specify a name or the AppleTalk address of the peer for the connection.

A DSP connection provides full duplex, reliable, flow-controlled delivery of a stream of bytes. In addition to a raw stream of bytes, the DSP provides a record service whereby clients use an end of message (EOM) flag to indicate that one or more datagrams should be treated as a single transaction. A client may also create its own layered protocols, by using a protocol type field in the DSP header which is not interpreted by the DSP.

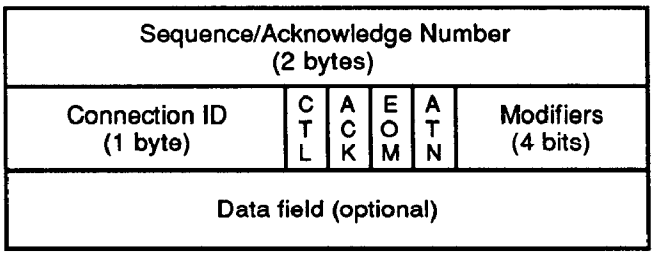
The sending side of the DSP accepts buffers of data from its client and sends that information across the connection. The DSP may fragment a buffer into several datagrams; each will contain a sequence number. Each of the datagrams is held in a retransmit queue, pending receipt of an acknowledge packet. The receiving side of the DSP checks the sequence number of the arriving datagrams, and sends back an acknowledge packet for any which are in-sequence. Acknowledge packets contain the sequence number of the next expected datagram and a receive window which indicates the number of additional maximum-length datagrams which may be sent. Datagrams which arrive ahead of sequence are stored in a short queue, pending arrival of the expected datagram(s).

If a block of data must be fragmented, each fragment retains the protocol type field, and all but the last have the EOM bit cleared. The last datagram has its EOM bit set to the value requested by the client. The receiver must observe the EOM bit and the protocol type field, and only reassemble packets into receive buffers which have the same type.

In addition to a reliable, sequenced stream of messages, DSP provides a means for its clients to exchange out-of-band information in the form of *attention* messages. Attention messages may be sent any time a connection is opened, even if the other end's receive window is closed. Delivery of these messages is best-effort; they are not sequenced, acknowledged or automatically retransmitted. Only one unread attention message will be held in a connection; subsequent attentions will be discarded.

The DSP Header

The DSP packets are sent as AppleTalk DDP datagrams. The DDP header carries several information fields, including the internet source and destination socket addresses. A DSP header follows the DDP header, using DDP protocol type $26 and contains information specific to the DSP protocol.



The DSP Header

Byte ordering is assumed to have the most significant (high-order) byte first, conforming to the AppleTalk conventions.

Sequence/Acknowledge Number (16 bits)

Generally, this field contains the sequence number of the packet. If ACK is set among the control bits, this field contains the sequence number of the packet the sender next expects to receive. If ATN is set, this field must be zero.

Connection ID (8 bits)

The sender's identification of the connection. Coupled with the sender's internet address, this allows unique identification of the sender's client by the receiving DSP.

The DSP uses a unique, incrementing connection ID number (ConnID) for each connection half, which must never be zero. The

connID is derived from a seed value maintained within each DSP implementation (initialized from a random source, such as the low order bits of a system clock); it is incremented at each connection attempt, and must be unique within the set of active connections.

Control bits (4 bits, mutually exclusive)

CTL: This packet transfers information between DSP peers; the CTL bit is not set by the client. The modifier field defines the control function. The data field may carry supplementary control information.

ACK: This is an acknowledge packet; the ACK bit is not set by the client. The modifier field contains the receive window. The sequence field specifies the next requested message. There is no data field.

EOM: This is the last data packet in a "record" message; the EOM bit is set by the client. The modifier field is user definable for implementing layered protocols.

ATN: This is a client supervisory control packet; the ATN bit is set by the client. The modifier and data fields carry user defined supplementary information. The value of the sequence field must be zero.

Modifier Bits (4 bits)

This field expands the meaning of the four control bits defined above (the default case, with no control bits, indicates a data packet).

If CTL is set, this field contains a control code which identifies the type of DSP control packet: note that the values marked as "processed immediately" will be processed even if ahead-of-sequence.

$0	Connection Request (non-privileged)
$1	Connection Request (privileged)
$2	Reserved for future expansion
$3	Connection Deny
$4	Disconnect Request
$5	Abort Advise *
$6	Acknowledge Request *
$7	Forward Reset Request *
$8-$F	Reserved for future expansion

*- processed immediately

If ACK is set, this field contains the sender's receive window, i.e. the number of maximum-length packets, beginning with the one indicated in the sequence field, which the sender of this packet is willing to accept.

In all other cases (ATN, EOM, no control bits), this field is available for client-defined, higher-level layered protocols. Reserved protocol values are:

$0	A data packet, requiring no special interpretation.
$1	Kiewit Network terminal handling protocol (TCFACE).
$2	Infosphere ComServe protocol.
$F	Extended modifier value – escape to next byte.

If the modifier field contains a value of $F, then the following byte (i.e. the first byte of the data field) contains the actual modifier value and the data field begins one byte later. This allows expansion to 256 modifier values. Extended modifier values of $00 to $0E have the same meanings as the basic value range.

Implementation recommendations

There is always a tradeoff between timely transmission of data and the desire to increase network efficiency by sending large packets. Optimal transmission strategies are an area of active research. The DSP specification offers recommendations which are simple (so they are likely to be implemented correctly) and safe (so they will work, with minimal impact, under all network conditions). Implementors may deviate from these recommendations, but must be sure that the differences will actually perform better than these recommendations.

Transmission Strategy

Whenever data is to be transmitted on an idle connection, the sender will foward a window-full of data. This is the number of packets specified in the receiver's receive window. Once a window-full has been sent, the sender will refrain from sending more packets until all outstanding packets have been acknowledged. This

minimizes the number of packets that the network has to carry, while providing near-optimal response for the client.

If the sender receives more data from the client while packets are still outstanding, it may attempt to combine the new data with other packets waiting to be transmitted (if the DSP protocol type and EOM permit). This increases the network efficiency by increasing the amount of data carried in the packet.

If the sender cannot combine the new data, it will queue a separate message (with a new sequence number) to be sent when all outstanding packets have been acknowledged.

Retransmission Strategy

The sender retransmits packets which have not been acknowledged. To ensure that the network is not flooded with retransmissions, the DSP must dynamically adjust its retransmission rate. The algorithm must be robust, and take into account link speeds which can vary by three orders of magnitude (1200 baud to 10Mbps).

The DSP maintains two values: an estimate of the current round trip time (RTT) across the connection, and the current retransmit timeout (RTO). The DSP estimates the RTT by measuring the elapsed time between the transmission of a packet and receipt of its ack message. This estimate should only be used if the packet has been transmitted once. The RTT estimate should be a weighted average of recent samples, with a faster time constant for increasing RTT (to react quickly to congestion).

The DSP will a) set the RTO to twice the current RTT estimate for the initial transmission of a packet; b) only retransmit the packet with the lowest sequence number. Retransmitting all the messages in the retransmit queue generally won't improve performance, since only one packet is likely to have been lost. Subsequent packets would be waiting on the receiver's queue of ahead of sequence messages, and would not need to be retransmitted; c) double the RTO (but not the round trip time estimate) any time a packet is retransmitted. By backing off agressively, the DSP will not flood the network when load suddenly increases.

Window Management

The receiving DSP process sends acknowledge messages to update the sender's notion of the sequence number or receive window. The goal of the DSP is to minimize the number of acknowledge messages sent, and still give timely information about the receiver's state. The DSP will send an acknowledge when:

- An in-sequence message is received;
- The client empties the receive queue;
- The client drains the receive queue such that the receive window is twice the value sent in the previous acknowledge message;
- The 30 second "tickle" timer expires (without sending any other message).

The TCP to DSP Protocol Gateway Project

In parallel with the development of the AppleTalk network described above, local area nets using Ethernet have emerged in several key locations around campus. These include the engineering and computer science buildings and the Kiewit Computation Center. Whereas the typical AppleTalk device is a Macintosh, the devices on the Ethernet tend to be more powerful (and more expensive). They include the VMS and Unix VAX hosts mentioned above, Sun and Xerox workstations, and an increasing number of microVAX, IBM RT PC, and other small computers. All these devices can use the TCP/IP protocols to communicate over the Ethernet.

This division into two worlds, AppleTalk and Ethernet, has a major drawback: it is hard to communicate between the two networks. With AppleTalk wiring everywhere on the campus, and Ethernet connected to (nearly) all our host computers, we needed to find a general scheme for connecting an AppleTalk device with a computer on the Ethernet.

Currently, there are a few solutions which convert between AppleTalk DDP and IP datagrams. The most widespread are based on Stanford's SEAGATE technology, and its commercial implementation from Kinetics, Inc. While this hardware works, there are no off-the-shelf soft-

ware packages which will support the terminal sessions we require. Rather than implement TCP/IP, we decided to design a converter between the stream and terminal protocols of the AppleTalk DSP and TCP/Telnet. There were several reasons:

- The links between buildings are still slow (19.2 kbps). Full TCP/IP headers would give poor performance. Async AppleTalk at 1200 baud would be unusable.
- The network nodes which perform terminal service have very little code space left. A new protocol family probably would not fit in the machine.
- We have three separate implementations of the DSP: the network terminal handlers, the DCTS host, and the Macintosh. All three implementations would have to be changed to a new protocol.
- A third-party AppleTalk vendor is developing an "AppleTalk serial driver" which will replace the current async RS-232 driver in the Macintosh. Any terminal emulator will then work across the AppleTalk network, simply by installing the new driver.
- The Internet protocols still cannot provide several of the services we get from AppleTalk: dynamic network address assignment; simple routing algorithms for a local network; naming algorithms which don't require a system administrator to update name tables.

The Gateway Implementation

We plan to use a Macintosh II computer running A/UX (Unix System V, Release 2) as the initial platform for this gateway. The advantages are: an Ethernet connection is available; the full set of IP protocols with all of the support functions (host name tables, name servers, etc) exist; and it has a strong development environment.

The Macintosh II has enough processor power to handle a large number of connections at once. It can be directly connected to the Ethernet, so that the communication link will not be a bottleneck. Also, the Mac II supports a large memory (up to 8 Mbyte) for message buffering and connection databases. The Mac II is a mass-produced machine; we will not have to build any hardware for this project.

The Mac II used for the gateway will be dedicated to this task. There is no need to share it for other functions, since it is a fairly low cost piece of network equipment. If price becomes an issue, we can reduce costs by porting the code to less expensive hardware after the gateway is functional.

Software

There are several major pieces of software within the gateway. They are: the stream conversion algorithm itself; conversions between the terminal handling modes; naming – how a service on one side becomes known to the other side of the converter; and finally, software to control, monitor, and troubleshoot the gateway.

Several complicating issues arise: the stream protocols are not identical, so some mapping must be made (perhaps losing some information); there needs to be flow control between the two connections; the terminal handling functions of the two protocols are distributed differently between the layers, and must also be mapped to (nearly) equivalent forms.

The conversion between the TCP and DSP is simple. The gateway simply "holds down" the ends of two connections: a DSP connection to an AppleTalk device, and a TCP connection to an Internet device. When a DSP message arrives, it will be converted to an equivalent TCP form, and sent to the ultimate destination. Similarly, an arriving TCP message will be translated and sent to the DSP device.

Telnet and TCFACE (the Kiewit Network terminal modes) are roughly equivalent terminal handling protocols. They specify forwarding conditions (when to send characters which have arrived), no/local/remote echoing, binary/ASCII transmission, and interrupts (breaks) to a process, among others. The conversion between them is relatively simple, since TCFACE terminal handlers have a small repertoire, and map to a simple Telnet device. Similarly, host computers on the TCFACE side of the network expect simple terminal handlers, and so a full Telnet handler should match these expectations.

Name conversions are difficult to do in full generality. Connecting from an AppleTalk device to an Internet service is straightforward. The protocol converter can contain a list of all the host names on the Ethernet side, and AppleTalk's NBP provides an escape hatch for connecting to other names. Connecting the other way, from an Internet host to a dynamically assigned AppleTalk network address, is not currently defined. (This is a current problem in the Internet, too. We plan to watch developments in this area, and will defer to others who are creating standards.)

The gateway will provide a mechanism to monitor and debug the data flowing through it. A monitoring process, accessible only from a network privileged terminal, will display operational modes and data streams within the converter.

Implementation

The initial implementation of the gateway will connect the DSP (and TCFACE) code to the Unix login process. This acts as a proof of concept, and allows us to test our conversion and naming algorithms, and provides a quick login service to the Mac II. Next, the DSP code will connect to an outgoing Telnet connection. It may suffer from low performance, since each packet could require multiple context switches on its way through the machine. The final implementation will probably process individual datagrams, translating and passing them on to the other protocol code on the fly.

A separate control process in the gateway will accept DSP (or Telnet) connections. This process can then display connection state, data streams, etc. within the operating gateway.

Naming will also progress through several stages. Names (and addresses) of services will initially be hard-configured in a file which is read at the gateway's startup. Later on, we can read the Internet host tables, and implement a scheme for registering an AppleTalk (DSP) service with the gateway.

Enhancements

Follow-on projects for the protocol converter, which haven't been well specified, include:

- Incorporation of the AppleTalk-to-Ethernet gateway functions currently performed by the Kinetics hardware;
- Conversions to new protocols (X.25 PAD and transparent service; DEC's LAT; and any other stream protocols which seem useful);
- A Mac-like user interface to the control process, to enhance the ease of use for debugging and troubleshooting;
- Booting from the master node. This draws the gateway closer to the existing network management system, and reduces the training required for network personnel. By reading configuration information from the master node, we get a single point of control.

Experience and Conclusions

AppleTalk has served our users well. We have had terminal service since the fall of 1984. Our users have also purchased a surprisingly wide variety of hardware and software which exploits the cabling that we installed.

Virtually the entire AppleTalk protocol suite has been implemented as specified. The only modification has to been a minor change to the routing algorithms to facilitate starting up a large network. AppleTalk bridges normally exchange routing tables every 10 seconds. This works in an environment where the network configuration is static, or where there are relatively few hops between networks. Our modification is to have a node send routing tables whenever a new network number appears (or an old network disappears). Thus, new routing information spreads quickly.

The dynamic configuration properties of AppleTalk have greatly simplified the network maintainer's life. We no longer worry about routing connections through the network; the routing algorithms take care of the work. Configuration of host names, network numbers, zones, etc. is easy to maintain. The values are entered in a single file, and installed at network initialization time.

The wiring scheme used in the residence halls (a star connection of the wires of a bus) did give us troubles during the initial installation. The reflections induced at the point where the wires tie together made communication erratic at various points on the bus. A 20 to 30 ohm resistor placed across the signal pair at the connection point solves the problem by absorbing much of the reflected energy. (Note that this problem only occurs with our wiring scheme, not with Apple's standard connectors.)

The 19.2 kbps links between buildings have not proven to be a bottleneck for terminal traffic. This will certainly change when people begin to demand file access as a normal service. By that time, we hope to have higher speed links between buildings, so that the network can support the offered load.

We have seen poor performance due to a large number of hops in the network. Worst case distances between an outlying building and a central server can be six hops. The initial release of Apple's file server issues many (up to 20) non-overlapped requests to the file server. With a normal afternoon network round trip time of 0.25 second, there is a built-in delay of five seconds before any result can be displayed on the user's screen. We hope that subsequent versions of the file server and workstation software will improve the situation.

The advantages of our network management scheme are manifold: we have widely dispersed network nodes, not only around the College, but at remote institutions across the East Coast. It is relatively easy to install new software (even if we must do it at 3:00 AM, to avoid interrupting users). A single machine (the master node) checks the health of the network. Consequently, troubleshooting can begin very quickly after a problem is detected. We avoid the expense and inconvenience of a data scope, since each node contains software which displays what we need to see (and if the software isn't good enough, we can fix it so that it is).

One disadvantage of using a network node to monitor itself is that the act of watching a data stream changes its characteristics. Although rare, the formatting of the display can slow the node down enough to mask a problem. A more frequent problem is loss of data, since the monitor cannot format the data fast enough to keep up with the real time data stream.

Dartmouth has invested the following amounts in the network: the hardware is approximately $1M, dispersed over the 95 nodes. There is close to another million dollars worth of wire in the walls, counting both materials and labor. (When we wire a building for data, we try to install an outlet per office, since the incremental cost is small compared to the cost of adding an outlet afterward.)

The effort to implement the AppleTalk protocols (including the DSP and Async AppleTalk) and the DarTerminal application was about 12 years. For on-going maintenance, an engineer acts as a user-services contact for problems and the electronics repair shop assigns one full-time technician.

Acknowledgments:

The DSP described in this document is a joint project of Dartmouth College and Infosphere Incorporated.

Appendix - Kiewit Network as of June 1, 1987