1990

A Perspective on the Cypress Internetworking Technology

Thomas Narten

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Rajendra Yavatkar

Report Number:
90-1038

A PRESPECTIVE ON THE CYPRESS INTERNETWORKING TECHNOLOGY

Thomas Narten
Douglas E. Comer
Rajendra Yavatkar

CSD-TR-1038
November 1990

A Perspective on the Cypress Internetworking Technology*

Thomas Narten†
Douglas E. Comer
Rajendra Yavatkar‡

CSD-TR-1138
November, 1990

Computer Science Department
Purdue University
West Lafayette, IN 47907

November 8, 1990

Abstract

The Cypress Project at Purdue University started in early 1985 to explore ways of providing low-cost Internet access to the scientific community. The central goal of the Cypress project was to develop a low-cost internetworking technology that provides all the functionality of Internet Protocol at a cost as inexpensive as CSNET's Phonenet service. The Cypress network has been operational since November 1985 and has succeeded in meeting that goal.

This paper describes the design goals, protocols, implementation, and performance of the Cypress network.

*Supported by AT&T, CSNET, Digital Equipment Corporation, National Science Foundation (ASC-8518369), and Purdue University. Participants in the experimental prototype have purchased some of the implet hardware and paid some of the leased line costs.

†Present Address: Department of Computer Science, State University of New York At Albany, Albany, NY 12222

‡Present Address: Department of Computer Science, University of Kentucky, Lexington, KY 40506

1 Introduction

The Cypress Project at Purdue University started in early 1985 to explore ways of providing low-cost Internet access to the scientific community. The state of internetworking was far different then [16]. NSFNET did not yet exist and access to wide-area networks such as the ARPANET [11] or X25net [5] was expensive. CSNET operated Phonenet, which provided a limited service for exchanging and forwarding mail to Internet sites, but did not allow sites to develop network-based applications or exploit full-fledged network services such as remote login and file transfers. At the same time, the advent of personal computers and inexpensive network interfaces made it possible to connect machines at a site with a local area network such as Ethernet. The research community soon realized the need for ubiquitous network-based services in addition to electronic mail. The central goal of the Cypress project was to develop a low-cost internetworking technology to provide all the functionality of TCP/IP Internet Protocol suite at a cost as inexpensive as the Phonenet service. The Cypress network has been operational since November 1985 and has succeeded in meeting that goal.

The significant features of Cypress include good performance, consolidation of several functions into a multi-function packet switch [3], an efficient but flexible link-level protocol [4], an incrementally expandable interconnection strategy using an inexpensive coaxial packet switch [6], and the use of off-the-shelf, general-purpose hardware and software for packet switches and communication interfaces.

This paper describes the design goals, basic architecture, implementation, and performance of the Cypress Network.

2 Design Goals

The architecture and implementation of the Cypress network was influenced by the following design goals:

2.1 On-Demand, Low-Cost IP Connectivity

From the beginning, our goal was to provide Internet connections at a fraction of the cost of conventional connections. Connecting to the Internet means having full functionality, IP-based network connections and not just having some of the services that use IP (e.g., electronic mail). The reason for insisting on IP is that it provides the foundation on which many applications rest, and makes it possible for users to create their own applications that use the Internet. Thus, we intended low cost connections to mean low capacity but not limited functionality.

One subtle consequence of our decision to support IP was that we needed a technology that supplies connections to the Internet at any time. In particular, whenever a program wishes to communicate, it manufactures and transmits IP datagrams. Furthermore, processes that supply services (called *servers*) must accept incoming packets from other

sites asynchronously. Although previous experience using X.25 to send IP datagrams shows that it is possible to make connections on demand [5], existing tariffs in the United States made public data networks unattractive compared to dedicated leased lines. In addition, per-packet charges associated with public data networks have the negative effect of discouraging network use to its fullest capability. Permanent leased lines keep communications charges fixed independent of network use.

To keep costs to a minimum, we decided to investigate use of 9600 baud lines in developing an internetworking technology.

2.2 Incremental Expandability

At the time of Cypress design, many sites that housed a local area network of personal computers needed Internet access. For instance, the Department of Atmospheric Sciences at Purdue University needed Internet access to remote data and computing repositories at the National Center for Atmospheric Sciences in Boulder, Colorado. Such sites were willing to lease a dedicated line and devote a slower, inexpensive machine (e.g., a PC) as a gateway between their local net and the Internet. Such sites wanted to begin with minimum cost equipment and data circuit; they could expand capacity later as use warranted. Therefore, we concentrated on designing topology and protocols that allow a site to incrementally expand its Internet access.

2.3 Consolidation of Functionality

To reduce cost, we wanted to consolidate link-level packet switching, IP-level gateway routing, and high-level network monitoring and control into a single machine. From a subscriber's point of view, such consolidation of functions is valuable. Instead of having two computers, one for packet switching and one for IP gateway routing, the subscriber needs only one machine. Moreover, because the interface between Cypress and IP can be performed in software, the coupling is less expensive than the configuration used by the ARPANET or public data networks.

2.4 Protocol Efficiency

Consolidation of functionality into a single packet switch and use of slow speed links meant designing protocols that do not need much processing power. For example, although our protocols are flexible enough to add data compression, we decided to omit link-level data compression in the prototype network. Also, the link-level protocol was optimized for the transport of IP datagrams because the network was designed primarily for that purpose. Because TCP/IP handles retransmissions, we decided to use best-effort delivery at the link level with no attempt to detect or recover from transmission errors.

2.5 Flexibility

To allow experimentation and to accommodate new and multiple network-level protocols, we wanted a flexible link-level protocol. To keep Cypress flexible and amenable to change, we separate the notion of packet *handling* from packet *type*. Cypress frames have separate *frame handling* and *frame type* fields. As with most protocols, the type field specifies what the frame contains. For example, the type field might specify that a frame carries an IP datagram or that it carries a Cypress control message.

The handling field specifies how an implet routes and forwards the frame. For instance, the handling field specifies:

- whether the frame should be routed directly to a destination using routing tables (used for conventional transmission),
- sent using reverse-path forwarding (used to efficiently broadcast packets generated by IP),
- sent using selective flooding (used to propagate Cypress control messages across the network), or
- sent across exactly one link to a neighboring switch (used to build routing tables and detect link failures).

In most networks, packet handling is determined by the packet type, and special handling of packets is restricted to network control messages. The ARPANET, for instance, broadcasts special routing update packets to all packet switching nodes, but restricts the use of broadcast to network control messages. By contrast, the Cypress protocol specifies that each implet examine and act on the handling field first, and only examine the type field if a copy of the packet is to be delivered locally. Separating packet handling from packet type gives us the flexibility of experimenting with new types (e.g., new protocols) without requiring each packet switch in the network understand them. In addition, Cypress allows higher-level protocols to determine which handling is appropriate when sending a packet. For example, it is possible for an application to use IP broadcasting.

2.6 Off-The-Shelf Hardware and Software

To keep costs low, we wanted to use available technology wherever possible. In particular, we decided to restrict ourselves to conventional computer systems and communication interfaces. While it is tempting to imagine that an implet could be built from scratch for less than the cost of a commercially available processor, the economy of scale obtained with commercial products usually makes them less expensive. Thus, we chose to use commercially available computers for packet switches and adopted conventional communication interfaces, typically character-oriented serial devices. To allow consolidation of

many functions in a single switch and to reduce development time and effort, we decided to modify an existing operating system (Unix*) instead of building packet-switch software from scratch.

Starting with a conventional operating system had several important benefits. For instance, the Unix support of user-level processes facilitated the development of network monitoring and control software. Algorithms and protocols could be debugged and modified without rebooting the system and without halting packet-switching activity. In addition, Unix provides a framework for easily adding device drivers to the operating system, allowing straightforward insertion of the Cypress protocols between existing network software and physical devices.

3 Cypress Architecture

A Cypress network consists of a set of packet switches, called *implets*†, interconnected by dedicated, leased, data circuits. Implets exchange frames with adjacent implets by sending them over the interconnecting leased lines. Each implet functions as a store-and-forward packet switch, accepting incoming packets and routing them on toward their destination based on header information found in each frame. Like most packet switches, implets enqueue packets for transmission while waiting for idle time on one of the leased lines.

Viewed from above, a Cypress network is a *physical network*. Any Cypress node can send Cypress frames to any other Cypress node, by simply specifying the implet identifier of the implet to which the frame should be delivered. The processing and routing of such frames is handled transparently by Cypress, whether the ultimate destination is an adjacent neighbor or several hops away.

One implet resides at each subscriber's site, where it attaches to the user's local area network as shown in Figure 1. One of the most important ideas in Cypress is that implets communicate with subscriber machines over a local area network such as an Ethernet, making the system loosely coupled. The subscriber's machines think of the implet as their pathway to the rest of the world and route outgoing packets to it over the local area network. That is, the subscriber machines use the implet as an IP gateway that leads to the Internet. The implet accepts IP datagrams from subscriber machines and routes them over the leased line(s) toward their final destination. It accepts Cypress frames arriving over the leased lines and routes those destined for the subscriber's machine over the local area network.

Initially, we expected Cypress network topology to be vine-like with a single node attached to the Internet and other nodes attached in a long chain‡. The origin of the

*Unix is a registered trademark of AT&T Bell Laboratories

†The name *implet* comes from the ARPANET packet switch, originally called an *Interface Message Processor* or *IMP*.

‡Indeed, the name *Cypress* was chosen because it refers to a fast-growing American vine.

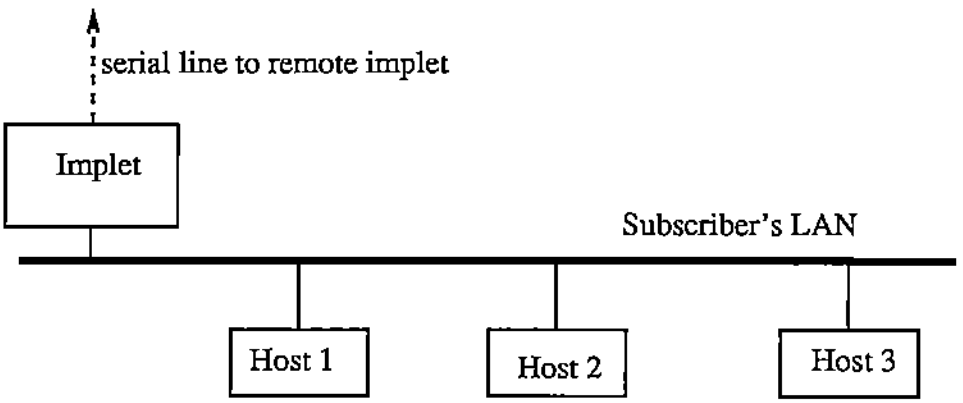

Figure 1: Connections at a subscriber's site.

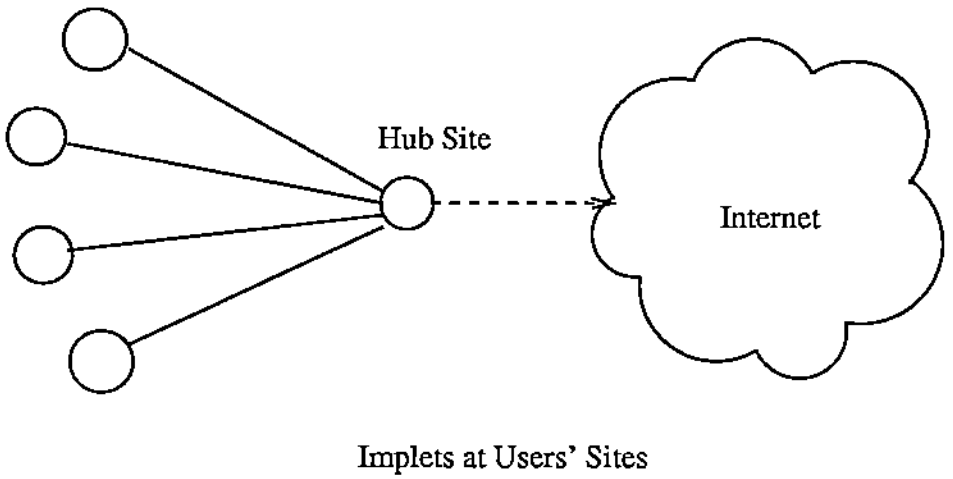

Figure 2: Connections at a hub site form spokes of a wheel.

growing-vine topology comes from Bitnet [8]. The idea is simple: each new site leases a line to the nearest existing site, keeping costs to a minimum.

Two things changed since the early days of Cypress, however, that made the long vine approach less appealing. First, as existing vine-structured networks grew and became more heavily used, their end-to-end delay increased and their performance grew worse. Second, with the advent of NSFNET and NSF-sponsored regional networks, it became apparent that instead of a single Cypress network, it would be more economical to have many, small Cypress nets each covering a geographical area. In geographic terms, it means having many *roots* of Cypress vines at NSFNET backbone sites or sites on the regional networks. To accommodate multiple roots, we developed a *hub-and-spoke* topology that supports wide fan-out at low cost. Conceptually, each hub site has a single implet that connects to the Internet locally, and to a set of n implets at n other sites. Thus, these connections form the spokes of a wheel as shown in Figure 2. The roots of these multiple networks connect to a higher speed backbone (such as an Ethernet),

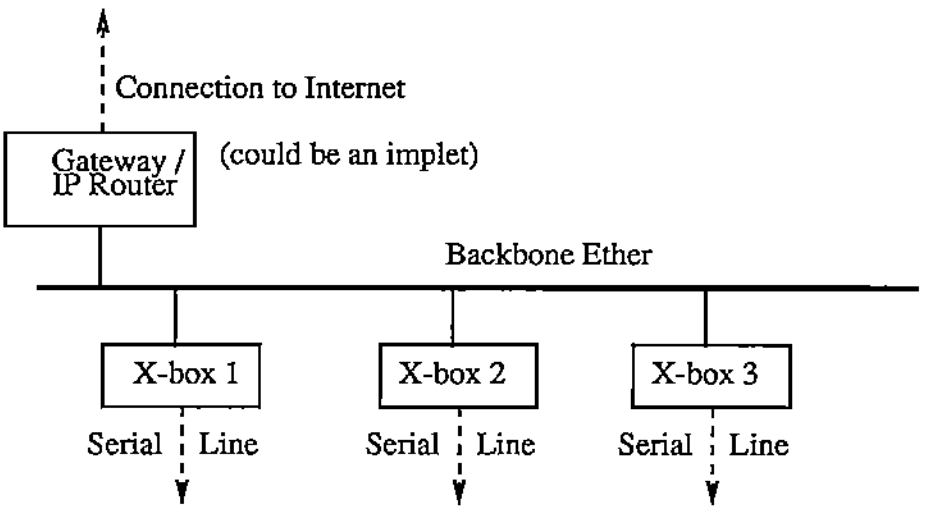

Figure 3: The connections of X-boxes at a hub site.

which in turn connects to the Internet through another gateway connected to the same backbone. The network can now be incrementally expanded by adding an implet (a root) at the hub whenever necessary.

Although a hub site could be expanded by adding additional implets, the economics of doing so are not attractive. Each implet can handle only a small number of serial lines before becoming overwhelmed by the overhead of handling serial line interrupts. To reduce costs, we developed a new packet switch architecture [6] in which the function of routing is separated from the function of data transfer to and from the network interface. The new architecture uses small, inexpensive machines called *X-boxes*. Conceptually, the technology uses an Ethernet as an extension to an implet's local bus to connect one gateway to many serial line connections as shown in Figure 3. The X-box design is described in Section 5.

3.1 Cypress Link-Level Protocol

Cypress uses its own link-level protocol and encapsulates IP datagrams inside Cypress frames. The link-level protocol design is optimized for transmission of IP datagrams. Figure 4 shows the protocol hierarchy in Cypress.

At the link level, Cypress provides best-effort delivery with no error detection or recovery from transmission errors. Indeed, Cypress uses no link-level checksum for frames, relying on higher-level protocols such as TCP to detect errors and retransmit packets. Our motivation to dispense with error detection and recovery stems from two considerations. First, error detection and recovery would have required additional bandwidth in the frame headers and would have increased the complexity of the protocol and its implementation. Second, the quality of terrestrial phone lines had increased sufficiently to consider relying entirely on recovery mechanisms at the end-to-end level. In retrospect,

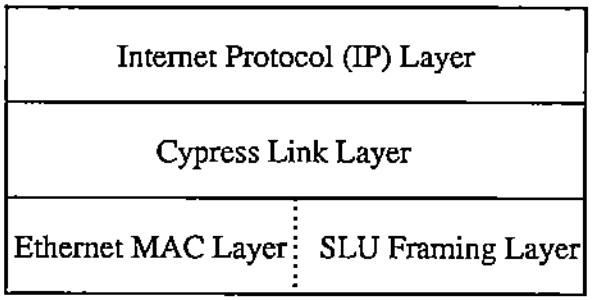

Figure 4: Protocol Hierarchy in Cypress.

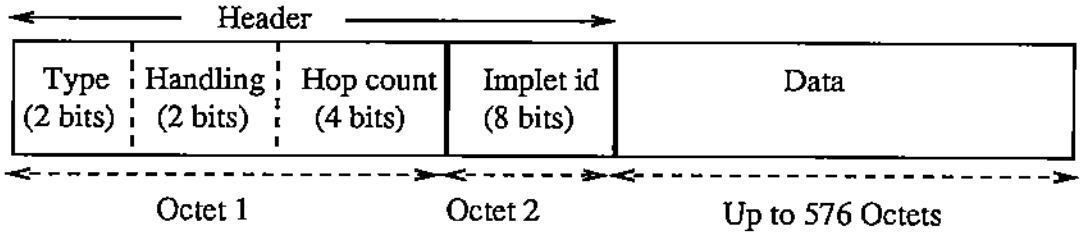

Figure 5: Cypress frame format showing the 2 octet header and data portion.

our decision was a significant factor in getting a prototype network operational quickly and did not lead to problems with data corruption.

The unit of transmission in the network is a Cypress frame that contains two fields: a short, fixed-length *header* followed by an optional *data* field. The standard header contains two octets of information, although the protocol allows longer headers under certain circumstances. The data field contains from zero up to 576 octets. The exact format of Cypress frames is shown in Figure 5.

Cypress frames contain a 2-bit *type* field that specifies what kind of data the frame contains. Table 1 shows the possible frame types. Type *IP* frames carry IP datagrams, while type *Cypress* contain Cypress control messages. Cypress control messages are used within Cypress for debugging and control. Type *extended* indicates that the frame header is extended by one octet, with the frame's actual type found in the 8-bit extension. The extended type allows us to extend the protocol to carry new types of packets (e.g., ISO).

An 8-bit *implet identifier* specifies the frame's sender or intended recipient, and a

IP	Frame carries an IP datagram
Control	Frame carries a control message
Extended	Actual type value in extended header
Reserved	Future use

Table 1: Cypress frame types.

2-bit *handling* field specifies how the frame should be processed. The remaining four bits of the frame header are used as a hop count. The hop count is initialized upon a frame's entry into Cypress, and each implet decrements the count when it processes the frame. If the hop count reaches zero, the implet discards the frame. As Cypress evolved from a vine to a hub-and-spoke topology, it became clear that limiting the diameter of a Cypress network to sixteen nodes would not be a significant limitation.

Cypress defines four handling types:

Direct In direct handling, the frame's identifier field indicates the destination implet, and the Cypress routing function uses the identifier to index into a routing table that maps implet identifiers to outgoing Cypress links. If the frame identifier indicates that the frame has arrived at its destination, the implet delivers the frame to the higher layer protocol indicated by the type field.

Reverse Path Forwarding (RPF) In RPF, the frame's identifier specifies the originating implet, and an implet forwards the frame to its neighbors if it believes that the frame is a new one rather than a copy of a frame it processed earlier. Specifically, the processing implet inspects the implet identifier in the frame's header, and if the frame arrived on the line over which frames addressed to the originating implet would normally be sent, the frame is assumed to be a new frame — it arrived on the link included on a shortest path to the originator — and is delivered locally as well as forwarded out over all other lines. If the frame arrived on a link not on the shortest path, it is assumed to be a duplicate and is discarded without further processing.

Selective Flooding For selective flooding, an implet sends a copy of the frame on each Cypress line except the one on which it arrived, and accepts a copy of the the frame for local delivery. Both selective flooding and RPF can result in looping packets. The hop count field prevents them from circulating endlessly.

Nearest Neighbor With nearest neighbor handling, an implet sends the frame across exactly one link to a neighboring implet. Nearest Neighbor handling is especially useful for communicating with adjacent neighbors *without* knowing who the neighbor is.

In most networks, packet handling is determined by the packet type. In Cypress, the handling and type fields are purposefully kept disjoint to reflect the orthogonality of contents and handling. Handling determines how the packet is to be routed irrespective of the packet's content. Each Cypress implet first examines the frame handling field (and possibly the implet identifier); the type field is examined only if the frame is to be delivered to the implet itself. Thus, implets can forward frames that contain an unknown type field defined by an experimental protocol. Different types of handling also allow the implets to exchange frames in the presence of incomplete information or errors such as software malfunction and routing inconsistencies. For example, while proper delivery of

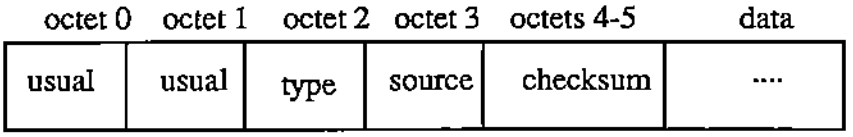

Figure 6: The format of Cypress control message frames.

RPF frames relies on correct routing tables, flooding and nearest neighbor handling do not.

3.2 Control Messages

Cypress packet switches exchange control messages for network maintenance and debugging purposes. The format of a Cypress control packet is shown in Figure 6. To ensure integrity and reliability, all control packets carry a 16-bit checksum computed in software using the IP checksum algorithm [1]. The types of control messages defined in the protocol include *echo requests* and *echo replies* for testing reachability and connectivity, a *reboot request* and a *Flush IP cache* request to allow remote control over an implet's operation, and a *route request* to query an implet about its current route for a specified destination.

3.3 Efficient Routing of IP packets

When IP hands a datagram to Cypress for delivery, IP routing software specifies the IP address of the destination implet to which the datagram should be delivered. Cypress takes the datagram, encapsulates it within a Cypress frame, using the destination IP implet address to fill in the Cypress frame header fields. The implet identifier is extracted from the least significant eight bits of the destination IP address. As the frame is carried through the network, each implet routes the frame using a simple table lookup that maps destination implets to outgoing lines. While a frame travels through Cypress, the IP layer in intermediate implets does not examine the forwarded frames at all. Isolating packet switching from IP header processing makes packet switching efficient; the implet does not recompute IP routes or checksums before forwarding the packet.

If the IP destination address is a broadcast address, Cypress software inserts the sending implet's identifier into the Cypress frame header and sets the handling type to reverse path forwarding. Mapping IP-level broadcasts into Cypress frames with reverse path forwarding handling is an important advantage of the Cypress protocols. It allows IP-level routing protocols to run directly on top of Cypress. In contrast, technologies such as Serial Line IP (SLIP) [18, 17] force the IP layer to be aware of each link, making it difficult to propagate routing updates to all nodes quickly. In addition, each SLIP line requires its own IP network number. In Cypress, the details of the link interconnections are completely hidden from the IP level.

4 Multi-Function Packet Switches

Cypress consolidates multiple functions into a single hardware package. Multi-function implets function as packet switches, IP gateways, and also run user-level processes.

At the lowest level, implets function as store-and-forward packet switches, receiving frames over leased lines, queueing them temporarily, and forwarding them on towards their final destination. At the second level, each implet functions as an Internet gateway, accepting packets from the LAN and routing them onto Cypress and vice versa. An implet must correctly route packets to any destination in the Internet, and it must propagate routing information to the Internet core gateways. At the third level, each implet functions like a host computer capable of executing user processes. It executes processes that monitor network traffic, log errors, and maintain routing tables.

4.1 User Level Functions

User level programs executing at low priority construct the routing tables used for packet switching, and modified kernel system calls install the tables into the kernel. Constructing tables outside of the kernel results in several advantages. First, packet switching activity takes precedence over the computation of routing tables, preventing the computation from delaying switching. Second, the building of routing tables uses spare CPU cycles, instead of competing for CPU cycles needed by the functions handling packet switching. Finally, user-level processes are easier to write and debug, and run in a private, protected address space. They can be stopped and restarted without requiring the implet to be rebooted, an operation that would stop packet switching activity for a significant amount of time.

Another advantage of building tables in user processes is that kernel routing tables can be updated in a single atomic operation. While the kernel is switching packets using its own table, the user process constructs a new or updated table at its leisure. Once the new table is complete, a single atomic update installs it into the kernel. Installing new tables in an atomic update prevents the kernel from using inconsistent routing information in which some entries are based on old information, and others on new.

In the Cypress network based at Purdue, a route server running on the well known implet *Cypress1*, located at the hub site, builds routing tables. The server maintains network topology and constructs routing tables using a shortest-path-first (SPF) algorithm. One interesting aspect of routing is the method implets use to gain initial tables. At the time implets are initially installed at a site, line zero is always configured to lie on the shortest path to *Cypress1*. The route server resides at a well known TCP port on *Cypress1*, waiting for route requests. At boot time, a process running on the implet installs a single Cypress-level route for *Cypress1* through line zero. It then opens a TCP connection to the route server and retrieves a current table. Upon successful retrieval, a copy of the routing table is cached in permanent storage. Thus, if the route server cannot be contacted for any reason, implets have a backup copy of recent routes. The

simple route server approach has worked well in the prototype network because network topology has been restricted to a tree for economic reasons.

Each implet monitors connectivity with each of its neighbors, with the goal of detecting implet or line failures. When testing the liveness of an implet, the testing procedure should not depend on Cypress routing tables. Otherwise, once a line stops functioning, and has been removed from the Cypress routing tables, it would be impossible to discover when the line starts working again. Cypress control messages with neighbor handling solve this problem. Each implet sends periodic Cypress echo request messages to each of its neighbors, using nearest neighbor handling to avoid reliance on routing tables.

A user-level Cypress line monitoring process sends Cypress control echo messages on each line every 15 seconds. A k out of n reachability algorithm [12] determines whether neighboring implets are currently reachable. If an implet receives k responses to the n most recent requests, the line is declared up, and Cypress routes frames over it. Whenever k of n responses are missed, the line is marked down, and the line in question is deleted from local routing tables. Use of a k out of n algorithm prevents rapid fluctuations in link status, yet ensures that a line that is not functioning would not be used. We adopted the algorithm after experience showed that sometimes it is possible to have a burst of line errors lasting a few seconds or less causing a momentary loss of carrier. Discarding packets during the short burst is preferable to marking the line down, because higher layers may overreact to notification that a link is not functioning properly and take action unnecessarily, such as terminating a connection. Cypress currently uses 3 and 4 for values of k and n respectively.

Monitoring line status using Cypress control echo packets provides a reliable mechanism for testing communications lines and the implets at both ends. When a link no longer functions properly, Cypress notifies higher-level protocols that attempt to send datagrams that travel across those links. For instance, IP returns ICMP destination unreachable [15] messages to the originator of a datagram. When Cypress declares a line down, Cypress discards all frames arriving on those lines except Cypress control messages. Discarding incoming frames insures that the line will not be used at all if the link fails in one direction only. Our experience shows that when a phone line fails, it often fails in only one direction, continuing to carry opposite direction traffic correctly. Unidirectional links cause considerable confusion to higher level routing protocols such as routed [9], which accept datagrams without verifying that the source of the datagram can be reached. Requiring a line to be operational in both directions has helped reduce the time needed to detect and correct operational problems.

4.2 Remote Network Monitoring

To monitor packet switching activity on remote implets, Cypress uses a server based monitoring system. A client program running on a local machine (not necessarily an implet) opens a TCP connection to a server running on the implet being monitored. The server sends the client statistics collected by the Cypress-level packet switching

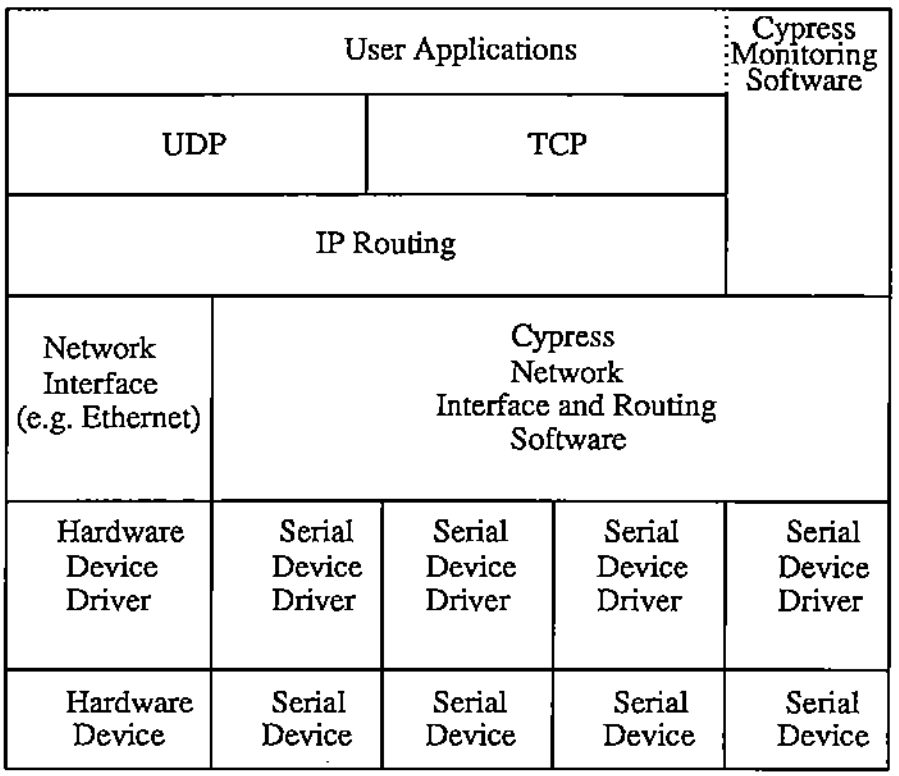

Figure 7: The placement of Cypress packet switching code relative to IP and hardware devices.

functions, which client software displays in a graphics-oriented manner. To keep the display up to date, the server sends updates every five seconds. Statistics collected and displayed include per-link packet counts, characters transferred, cpu utilization, and queue lengths.

The server based monitoring system is extremely powerful and has worked extremely well. The choice of TCP as a standard transport protocol offers several benefits. First, when debugging network problems, the information being monitored must be transmitted correctly and reliably. Building a separate transport protocol for monitoring purposes would require a substantial amount of effort, and would not provide more functionality than that provided by TCP. Second, an implet can be monitored from any host in the Internet. The monitoring host is not required to connect to the Cypress network itself. This is important for Cypress because hosts at subscriber sites do not directly connect to Cypress, only the implet does. Finally, using a standard protocol allowed us to concentrate on the information being collected, rather than on the details of how to get information from the implet to the monitor. In fact, the ease with which the monitor can be modified encouraged us to change the server several times to collect better statistics.

4.3 Kernel Functions

The Cypress packet switching code resides in the Unix kernel between the IP layer and hardware devices as shown in Figure 7. The method of passing datagrams between IP and Cypress is identical to that used between IP and other network interfaces. IP hands datagrams to Cypress via an input procedure made visible to IP, and Cypress places datagrams destined for IP in a shared queue. No changes to the IP code are required to use Cypress.

Cypress uses standard serial line devices for its lines. In Berkeley Unix, device drivers for terminal devices support multiple *line disciplines* [19], an abstraction that allows a single device driver to perform specialized operations on a per-line basis. Most serial line devices support multiple lines per device and the line discipline mechanism allows each line to be configured independent of the others. Cypress uses the line discipline mechanism to bind specific serial lines to Cypress. A Cypress utility program opens the terminal device, and changes the line discipline to that used by Cypress. The act of changing the line discipline invokes a Cypress initialization routine that binds the physical hardware device to a logical Cypress line, and Cypress proceeds to use it.

Once bound to Cypress, the line discipline mechanism handles the transmission and reception of Cypress frames from the hardware device. Complete Cypress frames ready for transmission are placed in a contiguous buffer that Cypress then hands to the output device driver. Devices that support direct memory access (DMA) transfer the entire buffer as a single operation, interrupting the CPU only after the complete frame has been transmitted. In contrast, input processing is complicated by the interrupt-per-character nature of serial devices. To reduce the interrupt load of input processing, device drivers use clock interrupts to poll the hardware device periodically. When the communications line is running at maximum speed, each poll retrieves several characters, reducing the total number of interrupts. When a complete frame has been received, the driver places the frame in a queue shared with the Cypress layer. Communicating with Cypress through a common queue allows the hardware device drivers to run at high priority, while the Cypress routines run only when the devices do not require CPU resources. Hardware devices run at the highest priority; when using low speed lines, we cannot afford to let lines remain idle when frames are ready to be transmitted.

Finally, hardware device drivers use upcalls [2] to notify Cypress of changes in device status, such as the loss of carrier. The upcall is accomplished through the line discipline mechanism. Although carrier loss itself does not prompt Cypress to stop using the line, logging status changes to permanent storage is important for trouble shooting and monitoring phone line quality.

User level programs access Cypress in two ways. Special system calls allow programs to install new routing tables, extract logged error messages, and determine the current configuration of lines. In addition, extensions to the Unix *raw* socket mechanism [20] provides user programs direct access to the Cypress packet switching layer. User programs pass complete Cypress frames to the Cypress layer, which route and queue them on the appropriate output line. Received Cypress control messages are passed to user

processes for processing.

5 Coaxial Packet Switch

In addition to performing routing and packet switching, an implet handles data transfer to and from the network interface. For the serial devices that were readily available when the Cypress project began, the transfer of Cypress frames to and from the serial line involved servicing an interrupt per character, limiting the number of lines an implet could operate simultaneously without performance degradation. For example, Cypress was first run over serial lines connected to a Digital Equipment Corporation DMF-32 terminal device. Like other devices available at the time, it performs direct memory access (DMA) on output, but was designed to accommodate slow, mostly idle terminals, processing input characters one at a time. Because the device contains only a small buffer for storing arriving characters, the CPU must service an interrupt per character under high loads, severely limiting the number of lines that can be attached to a single implet. To service more than a handful of lines, a hub site would need to add additional implets, at a relatively high cost. We developed a data transfer device called an *X-box* as a way to provide a high degree of fan-out at low cost.

A Cypress X-box provides a method of incremental hub site expansion. An X-box is a small, inexpensive machine that connects a Cypress line to the backbone LAN at a hub site, as shown in Figure 3. The key idea is to separate the function of routing from the function of data transfer to and from the network interface. Conceptually, an X-box handles the data transfer functions of the packet switch while a central router handles the IP-level routing functions. Each X-box accepts frames from the serial line and forwards them to the router for routing.

To make our novel architecture efficient, X-boxes cache routing information so that, in the steady state, each X-box routes IP datagrams (received over the serial line) using the Backbone Ether to their proper destination without using the central router. An X-box acquires the IP routes through the ICMP redirects it receives from the central router. Thus, because the central router is only needed to reestablish caches after a power failure, many X-boxes can be connected to a given hub. This efficiency enhancement cannot work if a standard IP gateway is used at a hub site along with the X-boxes (see Figure 3) because IP gateways send ICMP redirects only to originating hosts and not to routers (an X-box in this case). Thus, we must use a modified Cypress implet that recognizes the presence of X-boxes at the hub site and provides appropriate ICMP redirects.

Conceptually, an X-box is an implet that has all but the minimal functionality stripped from it. Thus, although the X-box processes and forwards IP datagrams, it is not a true Internet gateway, as it does not run any IP-level routing protocols. When it receives datagrams from the LAN, however, it encapsulates them within Cypress frames for transmission over the serial line.

An X-box is also useful for small sites that can only afford smaller, inexpensive

personal computers and cannot afford to devote an implet (a minicomputer) to connect their LAN to the Cypress hub (see Figure 1). At such sites, an X-box (e.g., a PC) can replace the implet on the LAN and still provide the necessary packet switching function. Thus, a site can start small and add processing power later as its network usage grows.

The following summarizes the salient features of an X-box; the details of the X-box design and implementation are described in [6].

5.1 Dual Nature of X-box

One of the most important constraints in the design of an X-box was that it should be possible to use a "standard" IP routing gateway for the Internet Router, as Figure 3 shows. In particular, the router should not be required to understand Cypress protocols. At the same time, we wanted an X-box to fully implement the Cypress protocols, allowing it to be monitored and controlled from its Cypress lines. Specifically, an X-box must respond to Cypress echo requests that arrive on the serial line.

As a result, an X-box plays a dual role with respect to the sites in the Internet world and the Cypress world. On the Internet side, an X-box is viewed as an extension of a Cypress implet and an Internet router forwards the IP datagrams destined for subscriber hosts to the X-box. On the other hand, the Cypress hosts view the X-box as an extension of an Internet router and as a means of getting to other Internet hosts. In the Cypress world, an X-box is also treated as another Cypress node. The latter view is very useful for testing and maintenance purposes.

Each X-box has an Ethernet IP address for its Ethernet interface and its own Cypress implet identifier and Cypress IP address. An X-box uses its Ethernet IP address and the standard communication methods (e.g., ARP) when communicating with the IP router. It also handles ICMP requests addressed to its Ethernet address from either side and uses ICMP redirects received on its the Ethernet interface to builds a cache of IP routes. It does not act as a full-fledged IP gateway, however, omitting certain gateway functions such as fragmentation.

5.2 X-box Routing at the IP and Cypress Level

The X-box knows (or learns at boot time) its implet id (and hence, its Cypress IP address), its Ethernet IP address, and the IP address of a default router/gateway. All X-boxes on a physical net need not use the same router and an X-box need not know *a priori* what other X-boxes are on the same net at a hub site.

The X-box uses its route cache when routing IP datagrams from the serial line to the Backbone Ether, but ignores the routing cache when routing from the Backbone Ether to the serial line. It also uses the cache when routing ICMP echo replies back to the Backbone Ether. Routes in the cache are flushed periodically; if no clock is available, they are flushed after every k uses or by a global sweep that is activated by traffic.

Apart from IP, an X-box recognizes and fully implements Cypress neighbor handling

and Cypress link-level protocols. The X-box sends an Ethernet broadcast of Cypress *RPF* and *Flood* frames using a frame type field that indicates "Cypress" in the type field of the Ethernet frame. It forwards the *direct* frame arriving from the Ethernet across the serial line if they are not broadcast (using the Ethernet broadcast facility), or if they are not destined for itself. The X-box design also correctly handles Cypress packets that are RPF, flood, or directly routed IP requests.

5.3 X-box Implementation

An X-box consists of a CPU, memory, boot program (typically in ROM), Ethernet interface, and a serial line. The first production version used a Digital Equipment Corp. LSI-11/23 processor with 64Kb memory, and it has since been ported to an IBM PC. The entire X-box software is written as a single, interrupt-driven program that fits in a 32 kilobytes of static RAM. An X-box was used to connect a LAN at the Geosciences Department at Purdue to the hub in Computer Science Department. Atmospheric scientists in the Geosciences Department used an X-box for over two years for remote login and to transfer files of over 200,000 bytes from the National Center for Atmospheric Research (NCAR) in Boulder Colorado, going through at least 4 Internet gateways and Cypress.

6 Prototype Network

To test our ideas, we built and operated a prototype network centered at the Computer Science Department of Purdue University. The prototype used mainly 9600 bps leased lines with a 56 kbps line added later. Traffic between the prototype and the rest of the Internet flowed to the ARPANET through an Internet core gateway also situated at Purdue. The prototype, operational since November 1985, had lines run east to Massachusetts, west to California, southwest to Arizona, and north to Chicago, connecting the sites listed in Table 2.

Several of these sites relied on Cypress as their only connection to the Internet. For example, the Computer Science Department at the University of Arizona relied exclusively on Cypress for over two years. They routinely sent mail and logged on remote machines using Cypress.

6.1 Performance

Performance of the Cypress prototype has been extremely good. Because the protocols are optimized for transport of IP datagrams, the link level introduces little overhead. During file transfers, for example, user data accounts for approximately 85% of the raw hardware line speed. Typical throughput for a file transfer across a set of 9.6 kbps Cypress lines averages 800 Bytes/second (user data). FTP, TCP, and IP overhead accounts for most of the rest, with less than 4% attributable to Cypress.

Table 2: Sites on the Cypress Network.

Site	Location of Connection
CSNET Coordination and Information Center	BB&N Inc, Cambridge, Massachusetts
Digital Equipment Corporation Western Research Laboratory	Palo Alto, California
Williams College	Williamstown, Massachusetts
Boston University	Boston, Massachusetts
University of Chicago	Chicago, Illinois
University of Arizona	Tucson, Arizona
Atmospheric Sciences Department at Purdue University	West Lafayette, Indiana
Silicon Graphics	Mountain View, California

One of the important considerations in the design of the X-box was not to degrade the performance of Cypress by introducing a bottleneck at the X-box. We succeeded in achieving that goal, and X-box performance has been satisfactory. In experiments involving data transfer (message sizes ranging from 64 bytes to 1000 bytes) between two Cypress hosts with an X-box acting as a packet switch, we observed an average data transfer rate of 640 bytes/sec across 9.6 kbps lines. More details on the experiments and measurements are provided in [6].

Most important, user reaction has been fairly positive. Atmospheric scientists in the Geosciences Department at Purdue used Cypress without an interest in, or understanding of the underlying technology. They routinely transferred files of over 200,000 bytes from the National Center for Atmospheric Research (NCAR) in Boulder Colorado. Other sites also used Cypress for file transfer, remote login, and mail delivery.

7 Related Work

Since the development of the Cypress internetworking technology, two other methods, namely, point-to-point protocol (PPP) [14] and SLIP (serial line IP) [10, 17] have been proposed as Internet standards for transmission of IP datagrams over serial links.

The key difference between Cypress and the other two methods lies in functionality. Unlike PPP or SLIP, Cypress provides an internetworking technology that allows one to create a single physical network spanning a set of sites.

Like the Cypress link-level protocol, PPP is designed to carry traffic belonging to multiple network layer protocols including IP. However, PPP does not configure a group

of sites into a single physical network. Instead, it provides a method of establishing, configuring, maintaining and terminating a point-to-point connection over a serial link. A host must configure and establish a link-level connection for the duration of IP transmission.

Although the Serial Line IP protocol operates over low-speed serial lines [10, 17], it is a framing protocol for a single link rather than a physical network. Each SLIP serial line is treated as a single physical network, and each line requires an IP network number. Cypress creates a single network from a set of lines.

The ARPANET link-level protocols [11] differ from Cypress in many ways. First, the ARPANET use acknowledgments and retransmissions to insure that packets are correctly delivered to their proper destinations. Second, although the ARPANET has the ability to flood packets for the purpose of distributing routing updates to all nodes, higher layer protocols (e.g., IP) are only permitted to send unicast traffic. In Cypress, unicast IP datagrams are carried using direct handling, while broadcast datagrams are transmitted using reverse path forwarding. The idea of using reverse path forwarding in point-to-point network topologies is described in [7].

The original NSFNET backbone [13] used 56 kbps lines for its communication links, and its link-level protocols included checksums, acknowledgments, and retransmissions. The network provides no capability for sending IP broadcast, and each individual link is visible to the IP layer.

8 Conclusions

The Cypress project developed an Internetworking technology that allows sites to provide the connectivity at a fraction of the cost of other technologies available at that time. For the cost of a leased line and low-cost hardware (e.g., a workstation and modems), sites are able to join the Internet as full-fledged members, send and receive mail and files, and log into remote sites over Cypress and the Internet.

Our decision to use best-effort delivery at the link level has worked well. We were able to get the initial prototype network switching packets in a matter of months, and we have not observed any problems caused by the lack of link-level error detection and recovery.

Placing all serial lines under the control of Cypress, and permitting IP-level broadcasting across Cypress simplifies the distribution of IP routing information. All sites receive routing updates at (approximately) the same time, regardless of how many serial lines separated them. If each line is visible at the IP level, the speed at which routing updates reach remote nodes becomes proportional to the number of links separating nodes times the update interval, typically thirty seconds or more for the routing protocols in use at that time. [9].

9 Acknowledgments

Christopher Kent provided helpful comments and suggestions during the initial design and implementation. Mary Catherine Privette and Gregory Smith developed initial version of networking monitoring and control software. The participating sites provided funding, hardware, and leased lines rentals.

References

[1] R.T. Braden, D.A. Borman, and C. Partridge. Computing the Internet Checksum. DARPA Networking Information Center, September 1988. RFC 1071.

[2] David D. Clark. The Structuring of Systems Using Upcalls. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 171–180. ACM, December 1985. Available in ACM Operating Systems Review, Vol. 19, No. 5.

[3] Douglas Comer and Thomas Narten. Unix Systems as Cypress Implets. In *USENIX Conference Proceedings*, pages 55–62, February 1988.

[4] Douglas Comer, Thomas Narten, and Raj Yavatkar. The Cypress Link-Level Protocol. Technical Report CSD-TR-652, Purdue University, December 1986.

[5] Douglas E. Comer and John T. Korb. CSNET Protocol Software: The IP-to-X.25 Interface. In *SIGCOMM 83 Symposium*, pages 154–159. ACM, 1983. in Computer Communication Review, Vol 13., No. 2.

[6] Douglas E. Comer, Thomas Narten, and Rajendra Yavatkar. The Cypress Coaxial Backbone Packet Switch. *Computer Networks and ISDN Systems*, pages 383–388, March 1987.

[7] Yogen K Dalal and Robert M. Metcalfe. Reverse path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.

[8] Ira H. Fuchs. BITNET: Because It's Time. *Perspectives in Computers*, 3(1), March 1983.

[9] C. Hedrick. Routing Information Protocol. DARPA Networking Information Center, June 1988. RFC 1058.

[10] Leo Lanzillo and Craig Partridge. Implimentation of Dial-up IP for Unix Systems. In *USENIX Conference Proceedings*, pages 201–208, Winter 1989.

[11] John M. McQuillan and David C. Walden. The ARPA Network Design Decisions. *Computer Networks*, 1:243–289, 1977.

[12] David L. Mills. Exterior Gateway Protocol Formal Specification. DARPA Networking Information Center, April 1984. RFC 904.

[13] David L. Mills and Hans-Werner Braun. The NSFNET Backbone Network. In ACM, editor, *SIGCOMM '87 Workshop*, pages 191–196, August 1987. Printed in ACM CCR Vol. 17, No. 5.

[14] D. Perkins. The Point-to-Point Protocol: A Proposal for Multi-Protocol Transmission of Datagrams Over Point-to-Point Links. DARPA Networking Information Center, November 1989. RFC 1134.

[15] J. Postel. Internet Control Message Protocol. DARPA Networking Information Center, September 1981. RFC 792.

[16] John Quarterman and Josiah C. Hoskins. Notable Computer Networks. *Communications of the ACM*, 29(10):932–971, October 1986.

[17] J. Romkey. A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP. DARPA Networking Information Center, June 1988. RFC 1055.

[18] University of California at Berkeley. *Manual Page for slattach*, 4.3 bsd unix edition, June 1986.

[19] University of California at Berkeley. *Manual Page for tty*, 4.3 BSD Unix edition, 1986.

[20] University of California at Berkeley. *Networking Implementation Notes 4.3 BSD Edition*, 4.3 bsd unix edition, June 1986.