Order Number 1341484

Demonstration of a generic gateway for Ethernet connectivity: A Sytek to Ethernet IP router

Charette, Paul Gerard, M.S.

The University of Arizona, 1990

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DEMONSTRATION OF A GENERIC GATEWAY FOR ETHERNET CONNECTIVITY: A SYTEK TO ETHERNET IP ROUTER

by

Paul Gerard Charette

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

in Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 0

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Paul P Chretien

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Ralph Martinez — 17 July 1990

R. M. Martinez — Date

Professor of Electrical Engineering

ACKNOWLEDGEMENTS

I would like to thank Dr. Ralph Martinez for making this research possible, and also for providing guidance and assistance throughout. Thanks also goes to Drs. Max Liu and William Sanders for their technical review of this work. I would also like to express my gratitude to Jianyi Tao for his technical assistance and insight. Finally I would like to thank Yasser Al Safadi for his support and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

ABSTRACT

The University of Arizona Campus Internet consists of numerous Local Area Networks (LANs) attached to an Ethernet backbone. Any campus LAN which cannot interconnect with this backbone is effectively isolated. The generic gateway project was conceived to research and develop gateway systems which will help incorporate new LANs into the campus internet.

The work presented here is the design, development, implementation and testing of a gateway for a selected campus LAN. The candidate LAN which was selected was the University of Arizona Sytek broadband network, which includes the Sytek LocalNet 20 and System 2000 networks.

CHAPTER 1

INTRODUCTION

1.1 Background

Over the past few years, many University of Arizona departments and laboratories have implemented Local Area Networks (LANs) to interconnect their various computer systems. Each of these selected a LAN which best served its own needs, in terms of speed, throughput, flexibility and cost. The diversity of networking requirements among these University entities inevitably lead some to implement different LANs than others. This has resulted in a multiplicity of dissimilar LANs on the UA campus. This is unfortunate, as many of these entities now wish to exchange information.

In an attempt to alleviate the problem somewhat, an Ethernet backbone was installed at the UA campus. This helped campus connectivity significantly, since most of the campus LANs are Ethernets and could be directly connected to the backbone. The TCP/IP protocol suite was chosen as the protocol "spoken" on the Ethernet backbone. This allowed the UA network to become part of The Internet, giving the UA community access to information on a global

scale.

1.1.1 Statement of the Problem

As for the remainder of the campus LANs, the problem became how to internetwork with the Ethernet backbone, and thereby effectively become a part of the UA campus internet. Generally speaking, a network manager can facilitate communication between two different networks by using a gateway. A gateway is a computer system connected to two or more networks, appearing to each of these networks as a connected host. But instead of being the ultimate source or destination of data packets, it forwards them from one network to another [44]. Managers wishing to attach their LAN to campus-wide network must acquire a gateway which will connect their LAN to the Ethernet backbone.

1.1.2 Thesis Objective

In order to achieve campus-wide connectivity, each non-Ethernet LAN on campus requires a gateway. These gateways must convert between the non-Ethernet LAN's protocol and the TCP/IP-over-Ethernet protocol found on the UA backbone.

The focus of this research is to develop a "Generic Gateway" architecture which could be adapted to the design of other gateways which would incorporate currently isolated LANs into the Campus Internet. Towards this end, a particular non-Ethernet LAN was chosen for which a prototype generic gateway would be developed. After much consideration, it was decided that the LAN to be connected to the Ethernet backbone via the prototype gateway would be a Sytek Network (a Localnet 20 and System 2000 Network).

This thesis is concerned with the design, development, implementation and testing of this Sytek-to-Ethernet generic gateway prototype, shown in Figure 1.1.

1.1.3 University of Arizona Sytek Network

The University of Arizona Sytek Network is a connection-oriented broadband local area network consisting of approximately 500 nodes. The nodes are widely distributed in over fifty campus buildings through a coaxial cable plant. The Broadband cable plant contains four broadband networks and two video channels [43]. Each node on the Broadband cable plant represents a user device attached to the network via an intelligent interface device called a Packet Communication Unit (PCU). Each PCU has two serial

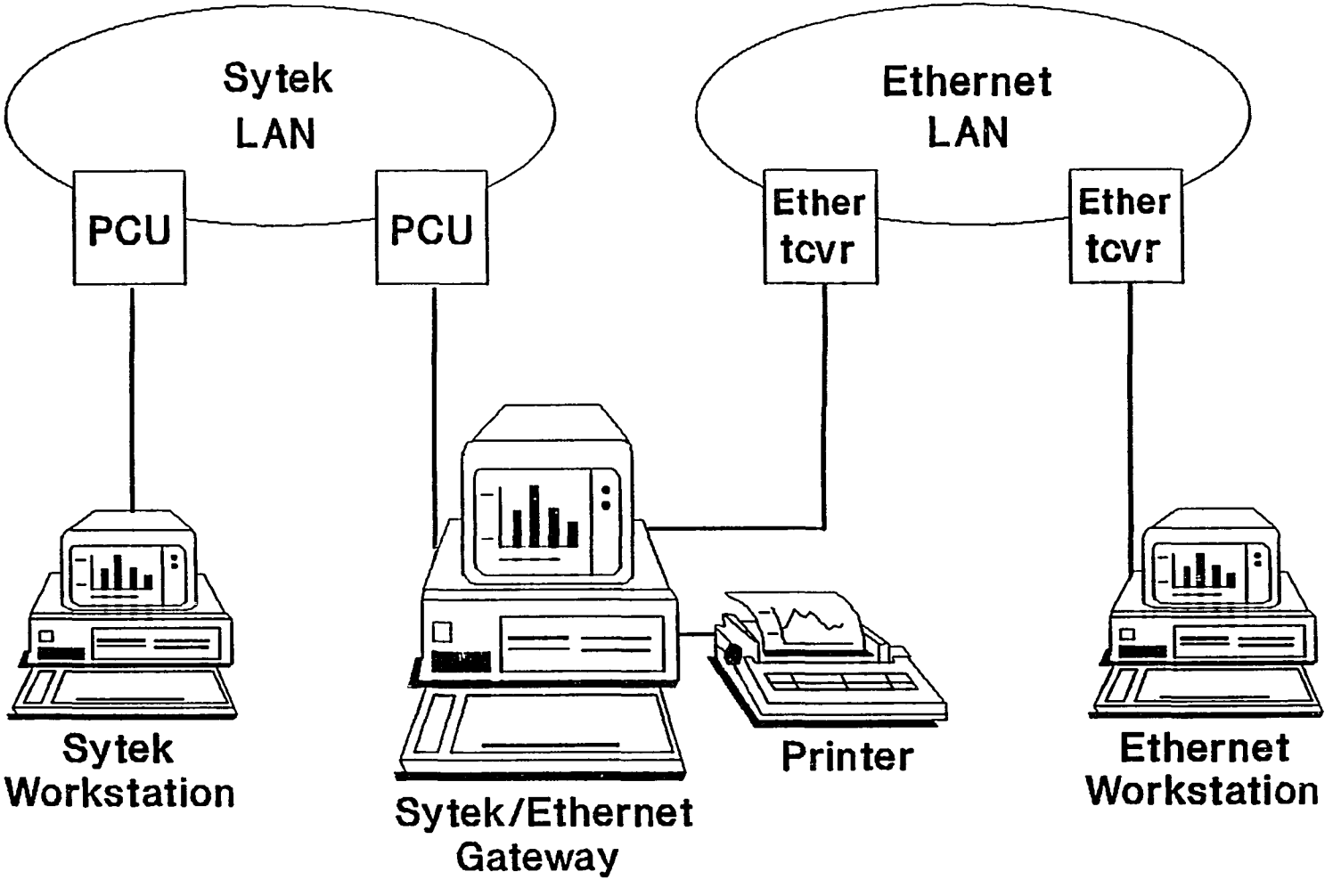


Figure 1.1 - Sytek to Ethernet Gateway

ports, allowing two user devices to be attached to the network by one PCU.

1.1.4 Sytek Network Services and Operation

The essential service provided to user devices by Sytek Network PCUs is the full-duplex transport of data between correspondent user devices over Sytek sessions. A PCU interfaces to user devices via ports that provide access points to a virtual terminal service provided by each PCU. Connection-oriented sessions provide the communication medium between ports.

The PCU provides the means for user devices to initiate sessions to other devices attached to the same or a different PCU. A source PCU, once a session has been initiated, takes data provided by a transmitting user device, formats that data into packets and uses the Sytek Network internal data communication protocols to route that packet to the destination PCU. The destination PCU removes the data from the received packets and transmits the data to the receiving user device. Data transmission is full duplex so that both correspondent user devices can be simultaneously transmitting and receiving. The manner in which data is presented to the PCU, and by the PCU to it's

attached devices, is under the control of the device.

The PCU is an intelligent network interface unit for a Sytek Network which has networking layers defined up to the session layer. The complete PCU protocol stack is given in Figure 1.2. The interface seen by the PCU user is the Session Management Protocol (SMP), which provides the network and session management functions to the PCU user. The next lower layer is the Sytek Network's transport layer, implementing the Reliable Stream Protocol (RSP). RSP provides reliable end-to-end data transport. The next lower layer is the Packet Transfer Protocol (PTP), Sytek's Network Layer protocol, which provides unacknowledged packet transfer service between PCUs on a Sytek Network. The next lower layer is Sytek Network's Link Access Protocol (LAP), which facilitates transfer of PTP packets using data link layer functions.

Each PCU provides a range of communication services to it's attached devices. These services include:

PCU Management - Overall management of the PCU, including specifying its identity, location, and protection attributes.

Session Management - Management of individual sessions, including session initialization, termination, and control.

Data Handling - Handling and processing of user data through sessions, including flow-and-error control.

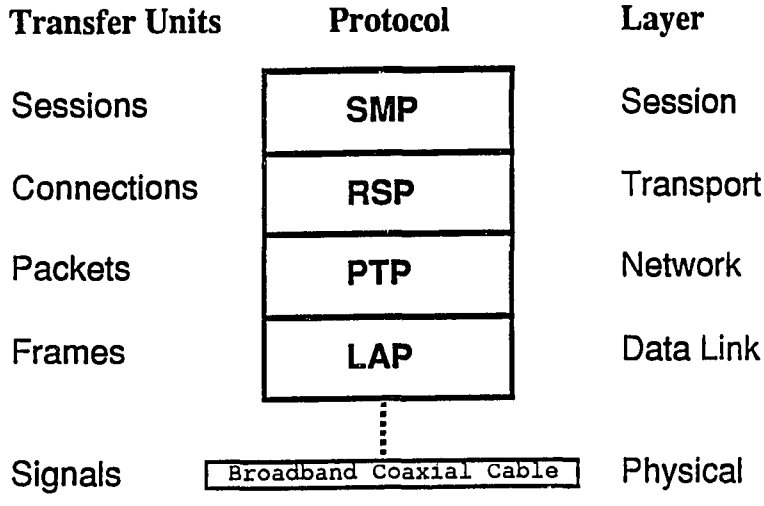


Figure 1.2 - Sytek Protocol Stack

Signaling - Out-of-band signaling across session boundaries. Allows the sending of an interrupt control signal without affecting buffered data packets.

Maintenance - Facilities provided for network diagnostics and maintenance, particularly local PCU diagnostics.

These PCU features are controlled by a set of attributes. These attributes are set via PCU commands from the user device. Attributes are divided into two categories: those maintained for the entire PCU, and those maintained for each PCU port. Sessions can individually control their per-port attributes.

Each PCU is identified on the network by a 16-bit address, called the PCU's unit ID. This allows over 65,000 PCUs to be configured onto a single Sytek Network. Individual ports on each PCU are identified by a port ID which can be either a 0 or a 1 for the standard 20/100 PCU. A complete Sytek Network node address then would specify both a unit and port ID, i.e. E905,0.

A PCU responds to both local and remote requests to initiate sessions. A local session initiation request takes the form of a call command from a user device attached to a local PCU. If a PCU has been enabled to receive an incoming session and a local session is available , then it will accept a remotely requested session.

The PCU supports termination of sessions through the use of the done command. Both remote and local users may terminate a session in this way.

For more complete information concerning the Services and Operation of PCU's, the reader is referred to the "LocalNet 20 Reference and Installation Guide" [38].

1.1.5 University of Arizona Ethernet LAN Connectivity

The University of Arizona has a large Ethernet Local Area Network, encompassing most of the campus. All of the hosts on campus with network connectivity implement the TCP/IP protocol suite over this Ethernet LAN. The University of Arizona Campus Internet is shown in Figure 1.3.

As shown in Figure 1.3, the UA Campus Internet has been assigned a Class B IP address (by the Network Information Center, NIC), specifically 128.196.xxx.xxx. The campus subnet mask has been defined as 255.255.255.0, so that any locally assigned subnetwork address will be Class C. Note that the same connectivity could be accomplished by having each subnet using a NIC-assigned Class C address. However, due to the size of the Internet, the Network Information Center (NIC) has adopted the philosophy of granting a large

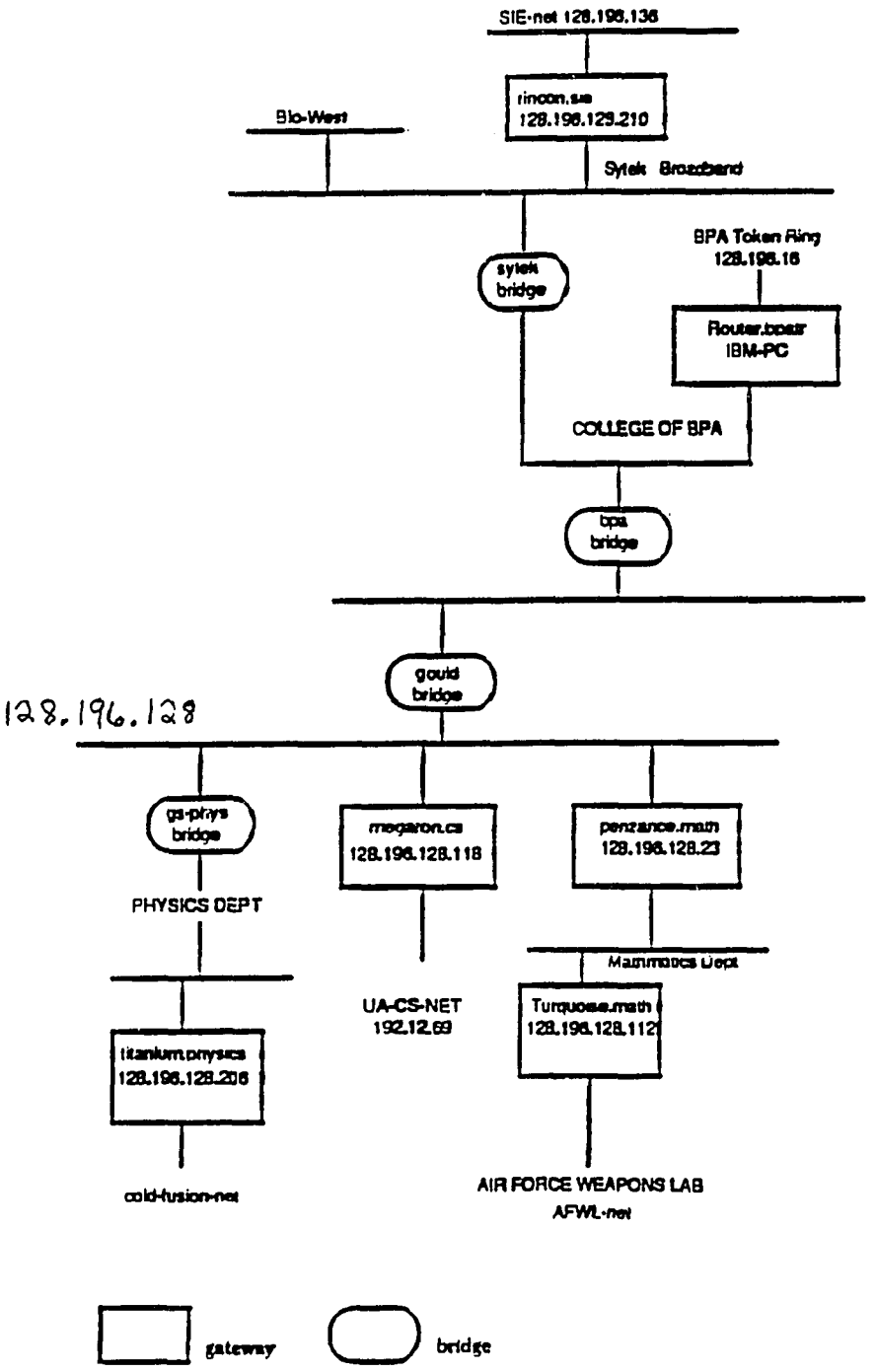


Figure 1.3a - The UA Campus Internet

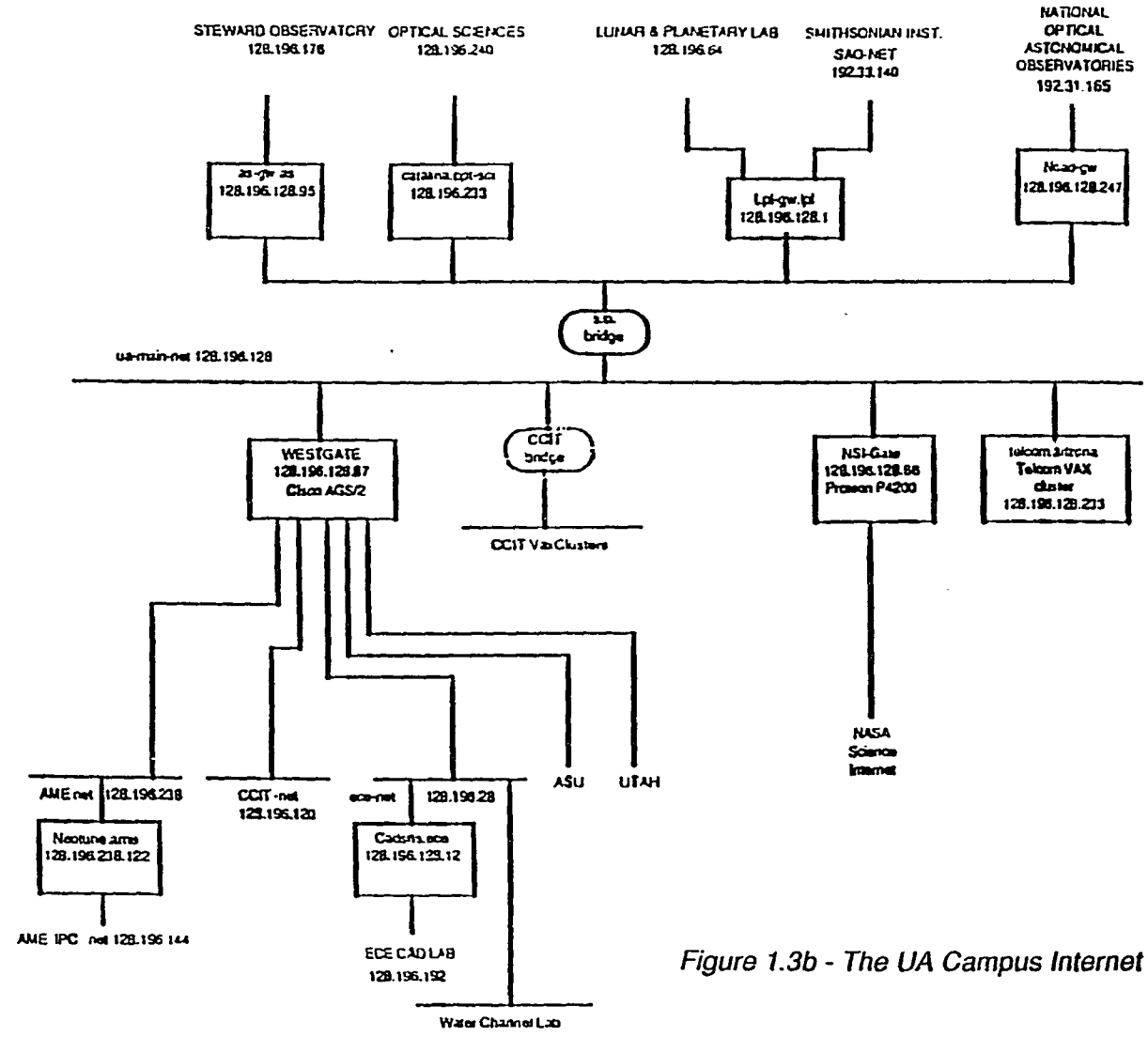


Figure 1.3b - The UA Campus Internet

address space to an institution, which is delegated the authority of defining subnetworks within this address space. This subnet strategy is discussed in RFC950, "Internet Standard Subnetting Procedure", [18].

UA is connected to the Internet primarily via a T1 link to University of Utah (shown in the figure as the link to UTAH from WESTGATE). This connects us to WestNet, and thereby the rest of the Internet. The connection from NSI-Gate to the NASA Science Internet is also used for Internet connectivity, but only as a backup link, when the WestNet link is down. This connectivity was subject to immediate change at the time of writing, so the interested reader should consult the UA Telecommunications Department for more information.

The astute reader may note the presence of a "Sytek Bridge" which connects the Sytek Broadband to the Campus Internet. This should not be misconstrued as an already existing Sytek-to-Ethernet gateway. What is being done here is that a channel of the Sytek Broadband frequency spectrum has been allocated as an Ethernet channel. Certain campus Ethernets use Chipcom Ethernet RF modems to run the Ethernet protocol over this RF channel.

1.1.6 TCP/IP Overview

This section is provided as a brief overview of the Transmission Control Protocol and Internet Protocol (TCP/IP) suite and it's function in an internetworking environment. Readers interested in a more detailed description are encouraged to first consult the eminently readable "Introduction to the Internet Protocols", [32], before attempting to read the specific Internet protocol specifications themselves.

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a packet switching network. It was developed by a community of researchers centered around the ARPAnet for the Department of Defense. ARPAnet was originally part of the Defense Data Network (DDN). Incidentally, the most accurate name for the set of protocols being described is the "Internet protocol suite". Because TCP and IP are the two best known protocols in this suite, it has become common to refer to the whole family as "TCP/IP" [32].

TCP/IP protocol software is organized into four conceptual layers that builds on a fifth layer of hardware. Figure 1.4 shows the conceptual layers as well as the form of data that passes between them.

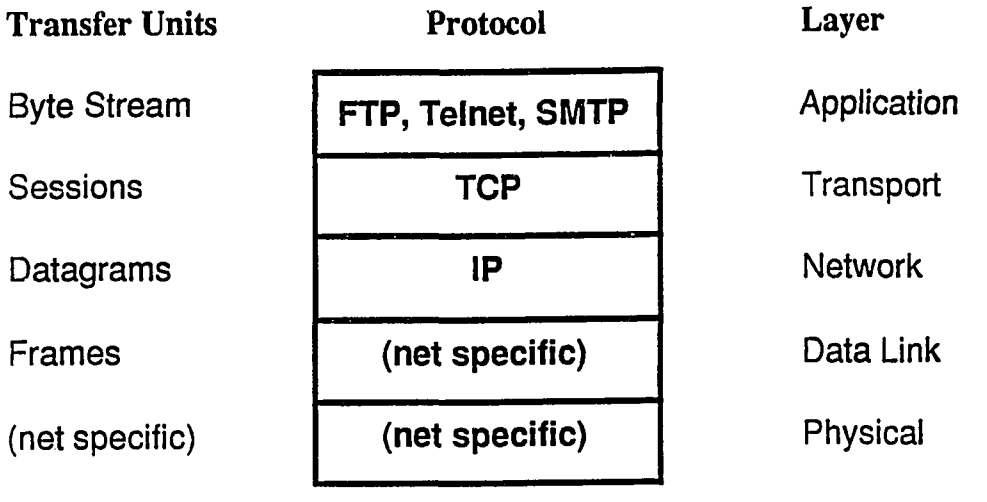


Figure 1.4 - Internet Protocol Stack

Application Layer - At the highest level, users invoke application programs that interact with the transport layer to transfer data over the network.

Transport Layer - The primary duty of the transport layer is to provide error-free end-to-end communication from one application program to another, even if they reside on hosts in different networks.

Internet Layer - The internet layer handles machine-to-machine communication. It accepts a request to send a packet from the transport layer along with an identification of the machine to which the packet should be sent. In an internetwork, it is responsible for routing a packet destined for another network to the appropriate gateway.

Network Interface Layer - This layer is responsible for accepting datagrams and transmitting them over a specific network to a certain host.

The more popular TCP/IP protocols and their relationship to this five layer conceptual model are given in Figure 1.5 (which was derived from the figure on page 9 of the "Transmission Control Protocol - DARPA Internet Program Protocol Specification," RFC 793). There are more protocols than these, but this is the "basic set" that are most commonly implemented on an internet host.

The applications shown are Telnet, FTP and SMTP. Telnet is a network virtual terminal facility allowing users bidirectional, byte-oriented communications with remote hosts. It is most often used to facilitate remote logins. FTP (File Transfer Protocol) provides its users with the

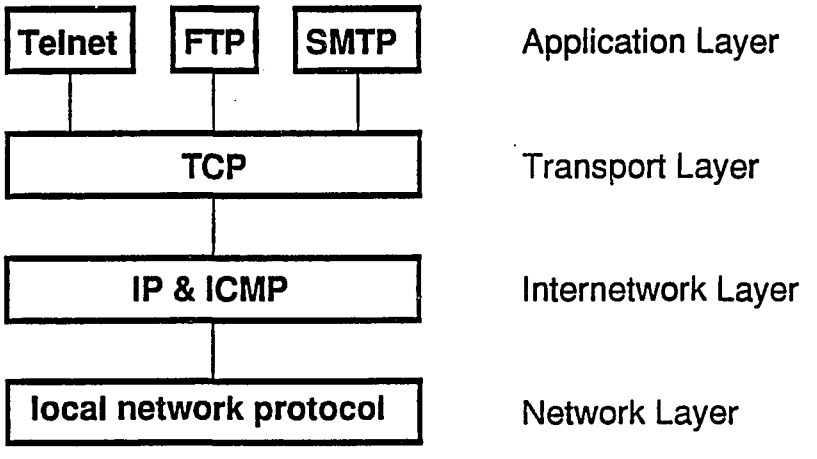


Figure 1.5 - Internet Protocol Architecture

capability to transfer complete files of data between distant hosts. SMTP (Simple Mail Transfer Protocol) is used to transfer electronic mail reliably and efficiently. These applications rest on top of TCP.

TCP (Transmission Control Protocol) is a connection-oriented, end-to-end reliable data transfer protocol. It is designed to provide communication between pairs of processes in host computers attached to distinct but interconnected networks. TCP assumes it can obtain a simple, potentially unreliable datagram service from the internet layer. Specifically, TCP expects an internet layer which may lose, duplicate, or damage datagrams which may be delivered out of order. To provide securable logical connection service between pairs of processes, TCP employs a complex scheme of windows, timers, sequence numbers, and acknowledgements which is beyond the scope of this thesis. The interested reader should consult the TCP Protocol Specification [27] and Chapter 12 of Internetworking with TCP/IP", [8].

IP (Internet Protocol) provides basic connectionless datagram service to a transport user (i.e., TCP). It encapsulates a segment of data in a datagram and attempts to deliver it. TCP only knows the segment's ultimate

destination; IP examines this and routes the datagram accordingly. If the destination is on a directly connected network, IP passes the datagram to the appropriate network interface for delivery. Otherwise, IP makes a routing decision as to what gateway this datagram should go to next and sends it on it's way. IP also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks. Within any single LAN, the IP is transparent and is dispensable, but it is critical for data transport among various multiple networks [42]. For a more exhaustive discussion of IP, consult the IP Protocol Specification [25] and Chapters 7-9 of Internetworking with TCP/IP", [8].

Occasionally a gateway or destination host will communicate with a source host to report an error in datagram processing. To accomplish this, ICMP (Internet Control Message Protocol) is used. ICMP is considered an integral part of IP, although it is architecturally layered upon IP. ICMP provides error reporting, flow control and first hop gateway redirection [29].

Below the internet layer is the network layer, which varies between implementations. It accepts IP datagrams from the internet layer and encapsulates them in network

specific frames for transfer to the appropriate host or gateway on their specific network. It also ensures that incoming frames destined for the local IP layer get there. A network interface may consist of a device driver, when the network is a local area network to which the machine attaches directly, or a complex subsystem that uses its own data link protocol, when the network consists of packet switches that communicate with hosts using HDLC [8].

1.2 Approach

The following section discusses the approach used in this research. It should be pointed out that at the time of writing, the area of gateway design has seen a great deal research. A reader interested in a general overview of recent developments in gateway design are directed to references [20], [35], and [37].

1.2.1 Gateway Design Issues

An interconnected set of networks is referred to as an internet, illustrated in Figure 1.6. Each constituent network supports communication among a number of attached devices. In addition, networks are connected by devices that are referred to generically as gateways. Gateways

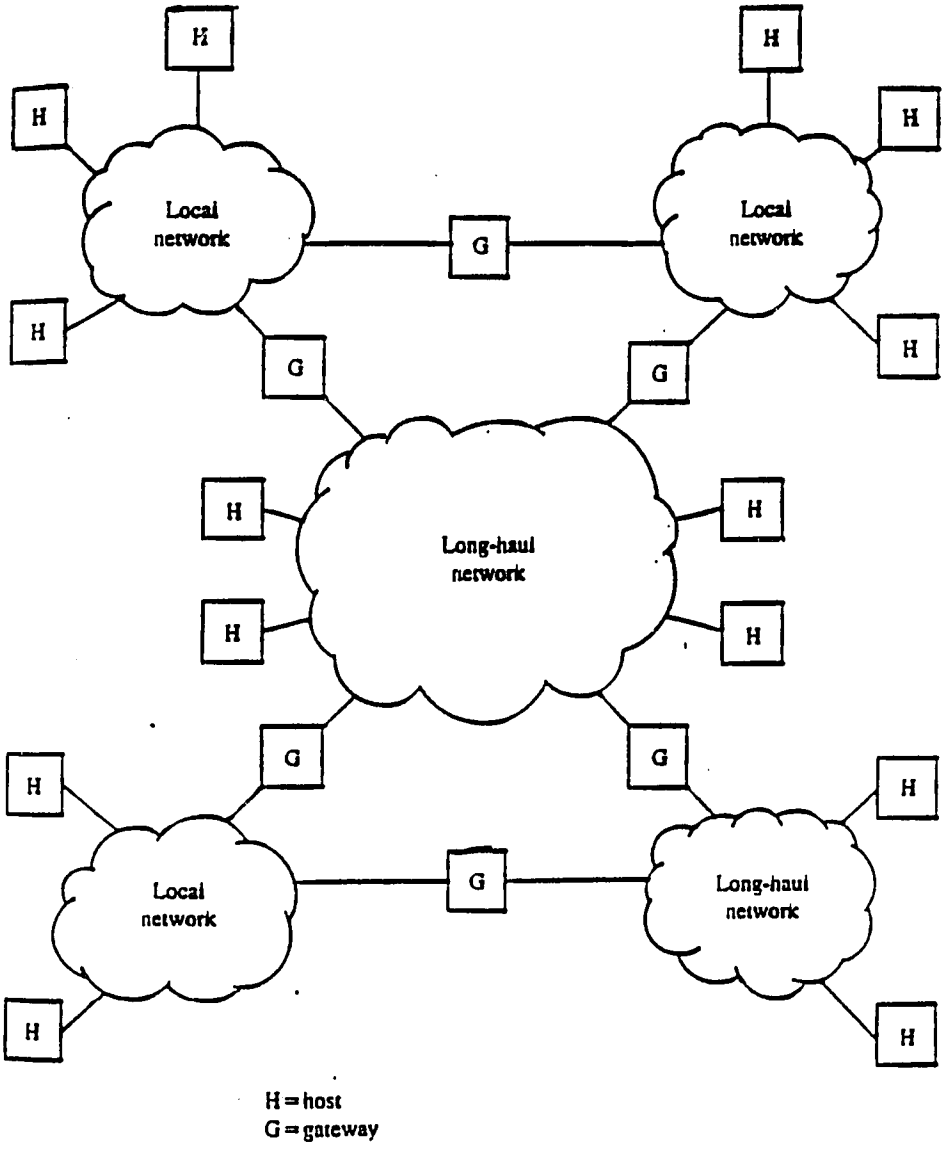


Figure 1.6 - A Typical Internet

provide a communication path so that data can be exchanged between networks [34].

Internet gateways, regardless of their implementation, must perform a variety of functions in order to make data communications between different networks compatible. These functions are (from [43]):

> Medium Transformation and Media Access - A gateway must translate messages between different transmission media, such as LAN RF baseband digital signals and a serial interface bit stream.
>
> Address Translation - Network addressing schemes are different on each network, so that address translation must be done by the gateway.
>
> Protocol Transformation - The network protocols of each network must be transformed through encapsulation and decapsulation of datagrams in network frames.
>
> Message Buffering and Flow Control - The gateway must be able to buffer messages from each network and flow control the network interfaces when the buffers are full.
>
> Error Connection Management - The gateway must provide an error-free link between two end users on the networks by adhering to the error control and retransmission mechanisms in the network protocols.
>
> Fault detection and Reporting - In establishing and maintaining a connection between two end-users, the gateway must be able to detect connection status and report to the users the condition of the link, if a problem should occur.

While providing this functionality, an internet gateway must meet the following requirements:

a. Provide advance routing and forwarding

algorithms.

b. Be implemented on a reliable system with a high degree of availability.

c. Provide advance network and gateway operations and management features.

d. Be implemented on a dedicated high-performance system.

To provide this functionality and meet these requirements for the generic gateway, it will be necessary to address the following design issues:

a. Addressing
b. Routing
c. Datagram Lifetime
d. Fragmentation and Reassembly
e. Error Control
f. Flow Control

Basically, there are two approaches to internet gateway design (Figure 1.7). First are the upper layer gateways, generally implemented as "application layer gateways". These have complete protocol stacks for each network interface, and transport data between them at high level of abstraction (i.e., an application-layer gateway facilitating a file transfer would transfer the entire file into local memory from the source user using his/her network's file transfer protocol, and then transfer it's copy of the file to the destination user using his/her network's protocol).

Application layer gateways are fairly simple to

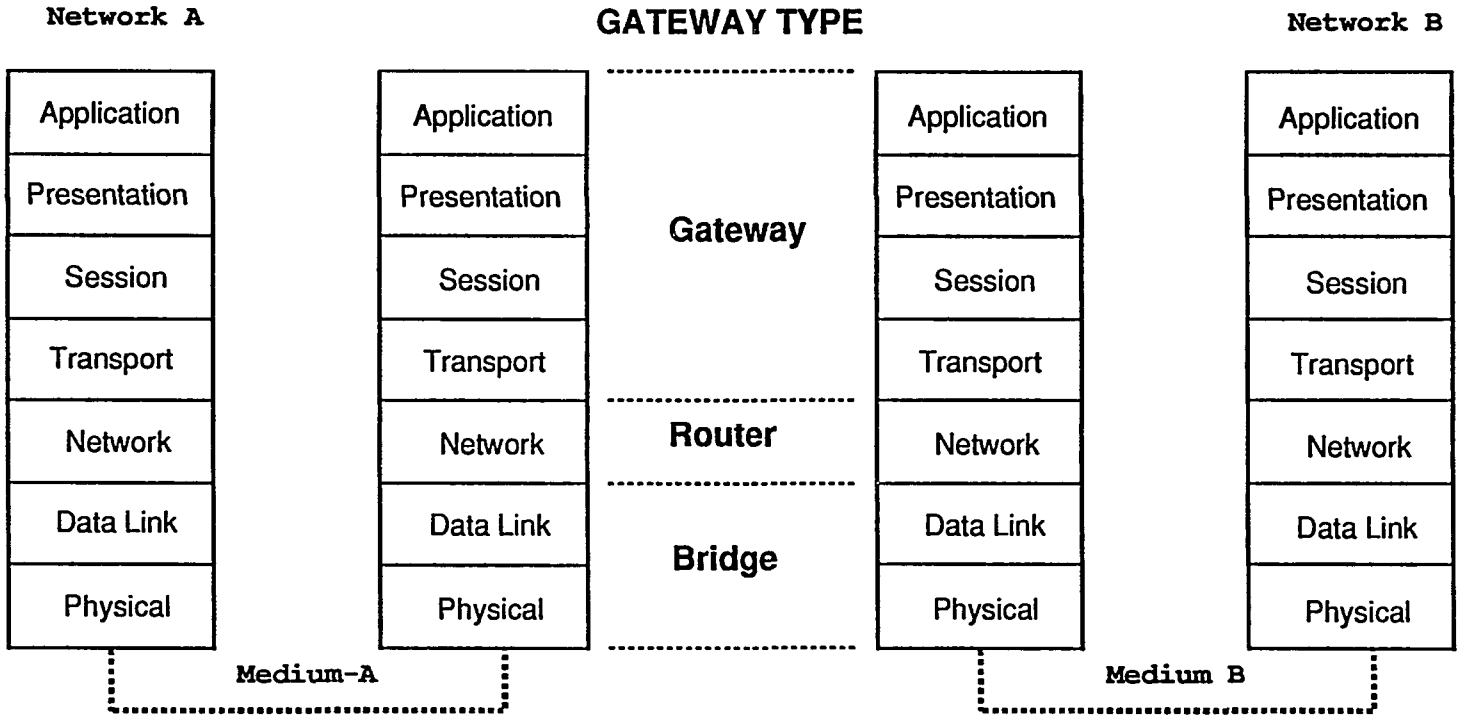


Figure 1.7 - Gateway Types

implement on multi-tasking computer systems by having a set of tasks operating as a user on each network with another task facilitating data transfer between them via shared memory buffers. However, application layer gateways have been shown to have very limited performance.

The second type of internet gateway are those implemented in the lower layers, generally at the network layer. These gateways are commonly referred to as routers, as they route independent packets of data without regard to connections. Performance of routers is better than that of application layer gateways because each unit of data requires less gateway processing. An internet connected by routers is also more robust, since a gateway crash only loses a few datagrams; with proper routing algorithms, a new path for datagrams associated with each connection will be found. However, routers are more complex to implement since you must have access to both network data link protocol and an internet protocol which will interface to each of them. Also, they depend upon similar upper layers in all communicating hosts.

1.2.2 IP-level Gateway

The primary objective in the design of the generic gateway prototype was to yield a gateway with a high degree of flexibility (e.g., "generic-ness"), so that the developed architecture could be used to implement gateways for other subnetworks on campus. It was decided to design the prototype gateway as a router, since this would yield the highest degree of flexibility. The fact that such a gateway would also yield higher performance also influenced this decision.

Since the gateway would be receiving Ethernet frames which encapsulate IP datagrams on one interface, it needed to implement an IP layer on top of an Ethernet network layer. By designing a network layer which would allow IP datagrams to be transmitted over Sytek Network, the prototype gateway could be implemented as an IP-router, which is the most common way to gateway between LANs in the Internet. Such a configuration is shown in Figure 1.8.

The "Sytek Network driver" would interface IP at one side, and the RS-232C hardware (to which a PCU is attached) on the other. The PCU provides the Sytek protocol suite to its user, defined up to the session layer (SMP).

At first glance, it may seem a bit strange to layer IP, a connectionless, unreliable datagram service, on top of SMP, a reliable, connection oriented byte-stream service.

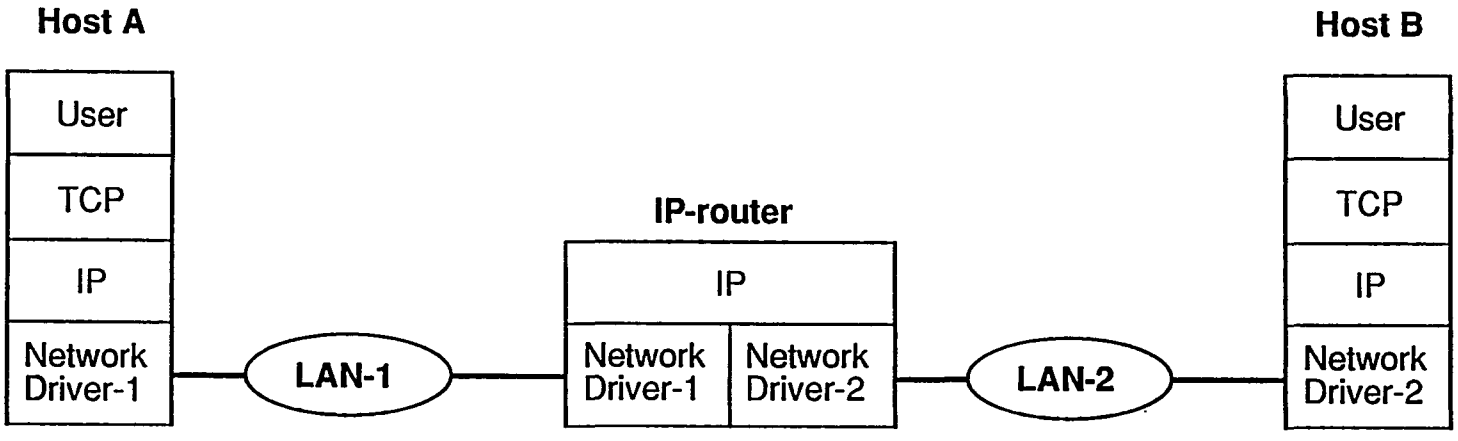


Figure 1.8 - Data Encapsulation With an IP Router

Indeed it is not the most suitable use of the Sytek Network's bandwidth. Ideally, we would like to be able to layer IP on top of the Packet Transfer Protocol (PTP), which would put our IP datagrams right out onto the coaxial cable for connectionless delivery. Unfortunately the protocol software of the PCU box is proprietary, and therefore unattainable. For this reason, the Sytek driver must layer IP on top of SMP.

The Sytek protocol suite is only defined up to the Session layer; There are no true Sytek application protocols to provide file transfer, electronic mail, and the like (Technically there is one application protocol - the "virtual terminal" protocol which provides transparent user access transparent access to SMP). Since the prototype gateway development will yield an IP driver for the Sytek Network, it makes sense to use TCP/IP and the standard TCP/IP applications in the Sytek hosts. This is will not only give Sytek Network users "Internet application access" (i.e., FTP, Telnet, and SMTP access to the Internet), but it will also allow them to use these applications between hosts on the Sytek Network.

Before we go on to discuss the design methodology used, let us summarize the steps taken by each IP module for a

standard internet data transfer including a gateway. This should clarify how an IP router fits into the standard internet architecture. To begin, the IP entity in the sending host receives a packet of data from it's user (quite likely this 'user' will be TCP) and performs the following steps:

1. Constructs an IP datagram for the packet base upon the service specified by the IP user.

2. Performs a checksum calculation and adds the result to the datagram header.

3. Makes a routing decision. Either the destination host is attached to the same network or a gateway must be selected for the first hop.

4. Passes the IP datagram down to the network access protocol for transmission over the network.

For each datagram that passes through a gateway, the gateway performs the following functions:

1. Performs a checksum calculation. If there's an error, datagram is discarded.

2. Decrements the time-to-live parameter. If this datagram's time has expired, discard.

3. Makes a routing decision.

4. Fragments the datagram, if necessary.

5. Rebuilds the IP header, including new time-to-live, fragmentation, and checksum fields.

6. Passes the IP datagram or datagrams down to the network access protocol for transmission over the network.

Finally, when a datagram is received by the IP entity in the destination host, the following functions are performed:

1. Performs a checksum calculation. If there's an error, datagram is discarded.

2. If this is a fragment, buffers until the complete datagram is reassembled.

3. Passes data and parameters from the header to the user.

1.2.3 Design Methodology

There are a variety of computer systems which could have been used as development platforms for the prototype gateway. Ideally, the development systems would have multi-tasking capability, could be dedicated solely to the project, and yet would be relatively inexpensive. Based on these criteria, 386-class IBM PC AT clones were selected. Normally these systems operate under MS-DOS, which is a single-task operating system. The hardware of these system does provide for multi-tasking, however and MS-DOS is not the only operating system which can be used on these systems.

There exist a number of public domain software packages which implement the TCP/IP protocol suite on 386-class IBM-PC clone systems. Rather than attempt to develop a TCP/IP

software package from scratch, it was decided that the best approach would be to acquire the public domain TCP/IP package best suited for development and integration of a new driver. This allowed concentration on the design, development, implementation and testing of the Sytek driver into this package.

By choosing to integrate the driver into a carefully selected, pre-existing package, a gateway can be implemented which already meets the requirements and performs the unctions mentioned in Section 1.2.1, provided the driver does not critically degrade the performance of the TCP/IP software.

It was determined that the candidate TCP/IP software package must be capable of multi-tasking (to exhibit the highly responsive nature needed for a LAN gateway), as well as be "developer-friendly" (easily extensible to include a new driver). It also must implement a "complete" TCP/IP protocol stack with all the "standard" applications and all the support protocols necessary for an internet gateway (e.g., ICMP, RIP, NDS).

In designing the actual Sytek driver, the following methodology was used: First the driver was designed as a protocol module specifying its handling of IP sends and

receives. It's functionality was partitioned into sub-protocol modules and these were defined in terms of services provided. The "user interfaces" (where the "user" may be another protocol or sub-protocol module) of each of these sub-protocol modules were then defined in terms of "C" functions and their passed parameters (i.e., from a "top-down" view). Then the actual code for these functions was designed (using flowcharts) and written. Each function was tested as it was developed (unit-level testing), and then they were synthesized into sub-protocol layers. Testing was performed to verify the functionality of these sub-protocol layers (sub-system level testing), and then they were synthesized into the Sytek driver, which was tested (system-level testing). Finally, complete gateway system level tests were performed.

It should be noted that this design-development-integration-testing cycle was performed in an iterative way (i.e., if a flaw was discovered in a function, it be corrected, and then that function would be tested, and then sub-protocol dependant on it would be re-tested).

1.2.4 Three Phase Development

The development of the prototype gateway occurred in three phases:

Phase I - TCP/IP Over Ethernet - During this phase, we wished to implement TCP/IP over Ethernet on a PC. The development system used is shown in Figure 1.9. Because of the popularity of Ethernet, finding a Public Domain TCP/IP with software support for the Etherlink card we are using was not difficult. The work during this phase involved deciding on which PD TCP/IP best suited our needs and then configuring it on our PCs such that it would run as expected. It was also confirmed that the package chosen could be compiled from it's sources.

Phase II - TCP/IP Over Sytek - During this phase, we wished to implement TCP/IP over Sytek on a PC. The development system used is shown in Figure 1.10. The protocol stack in each PC differs from Phase I only in the Network Interface Layer, "Sytek Driver". This layer facilitates IP communication over the Sytek Network. It's design and development was the main thrust of this research.

This development phase was broken down into five tasks:

1. Implement TCP/IP on two PCs communicating over a null-modem cable which connects to their serial ports. Such a configuration is logically similar to two PCs connected to PCUs who have a Sytek session established between them.

2. Connect the PCs to PCUs, and run TCP/IP over a Sytek session which has been established before runtime. The

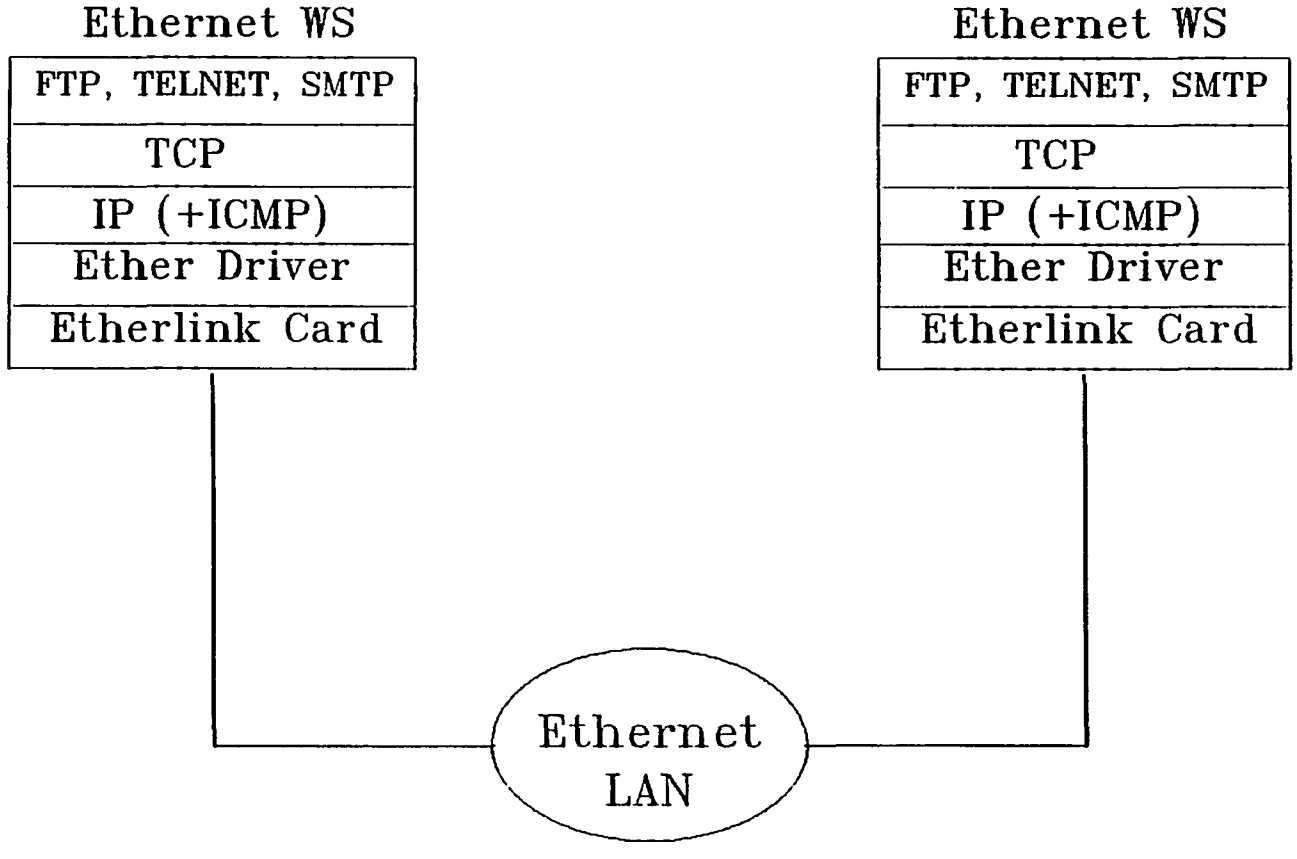


Figure 1.9 - Development System I

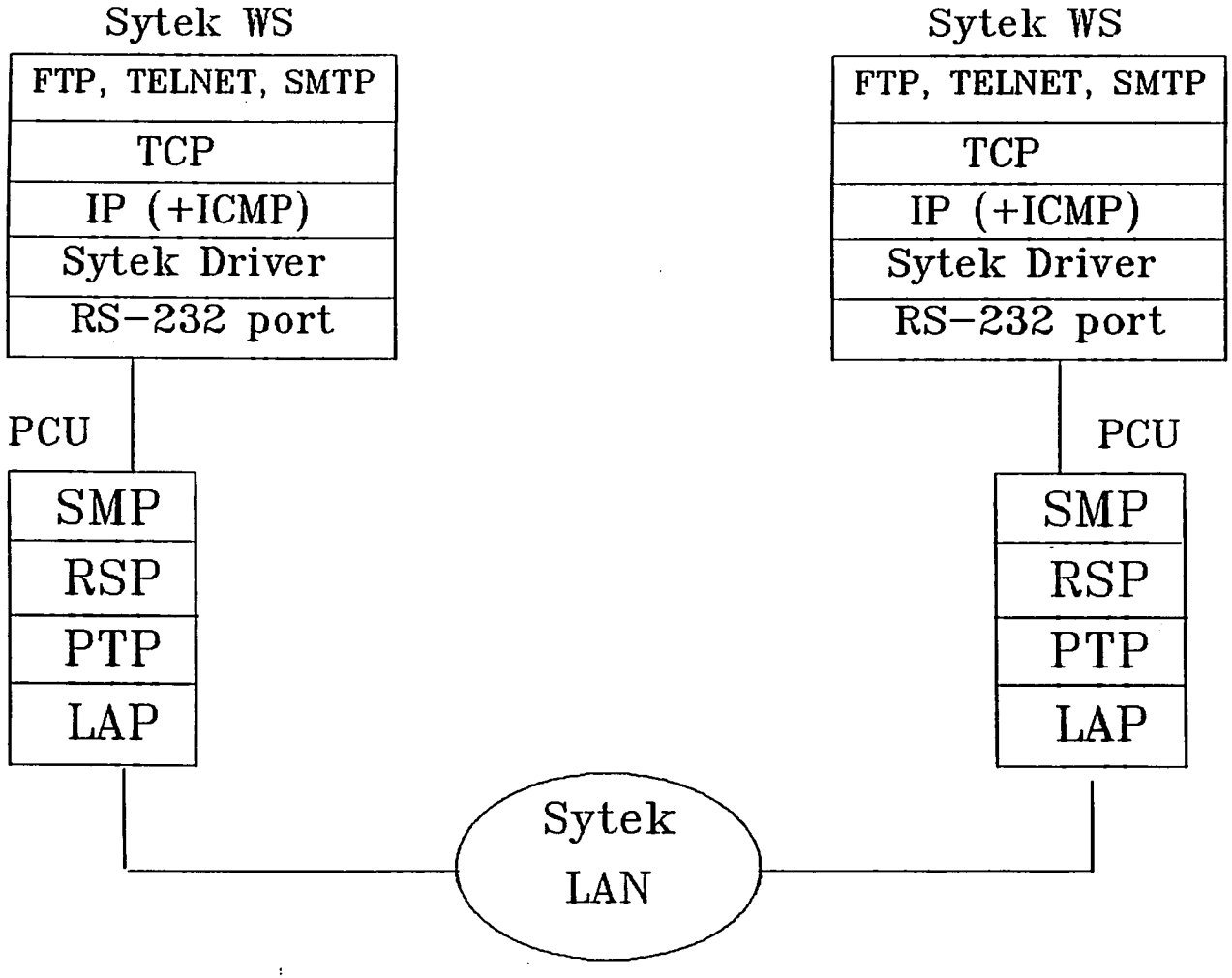


Figure 1.10 - Development System II

network driver must ensure that the PCU command mode escape sequence never appears in the data stream.

3. Extend drivers to ensure proper PCU parameter configuration before establishing a "hard-wired" session between the two participating PCUs, and run TCP/IP over it.

4. Modify the drivers to resolve IP addresses into Sytek hardware addresses via table lookup. Also, add session management; drivers must issue the appropriate calls and dones to establish and terminate Sytek sessions for IP datagram delivery. Such sessions should be terminated if left idle for too long.

5. Design and implement a Sytek Network address resolution protocol to allow resolution of IP addresses not found in our local tables.

Phase III - Sytek to Ethernet Gateway - During this phase, we wished connect a PC to both networks and configure it such that it would act as a gateway between them. The development system used is shown in Figure 1.11. The two workstations are identical to their counterparts in the first two phases. The gateway system implements the drivers developed in the first two phases on one PC under one IP layer. Above this is are the standard gateway management and control protocols, such as RIP and DNS.

1.2.5 Selection of KA9Q NOS

After careful examination of the available public domain TCP/IP software packages, the NOS version of the KA9Q Internet Software Package (hereafter referred to as NOS

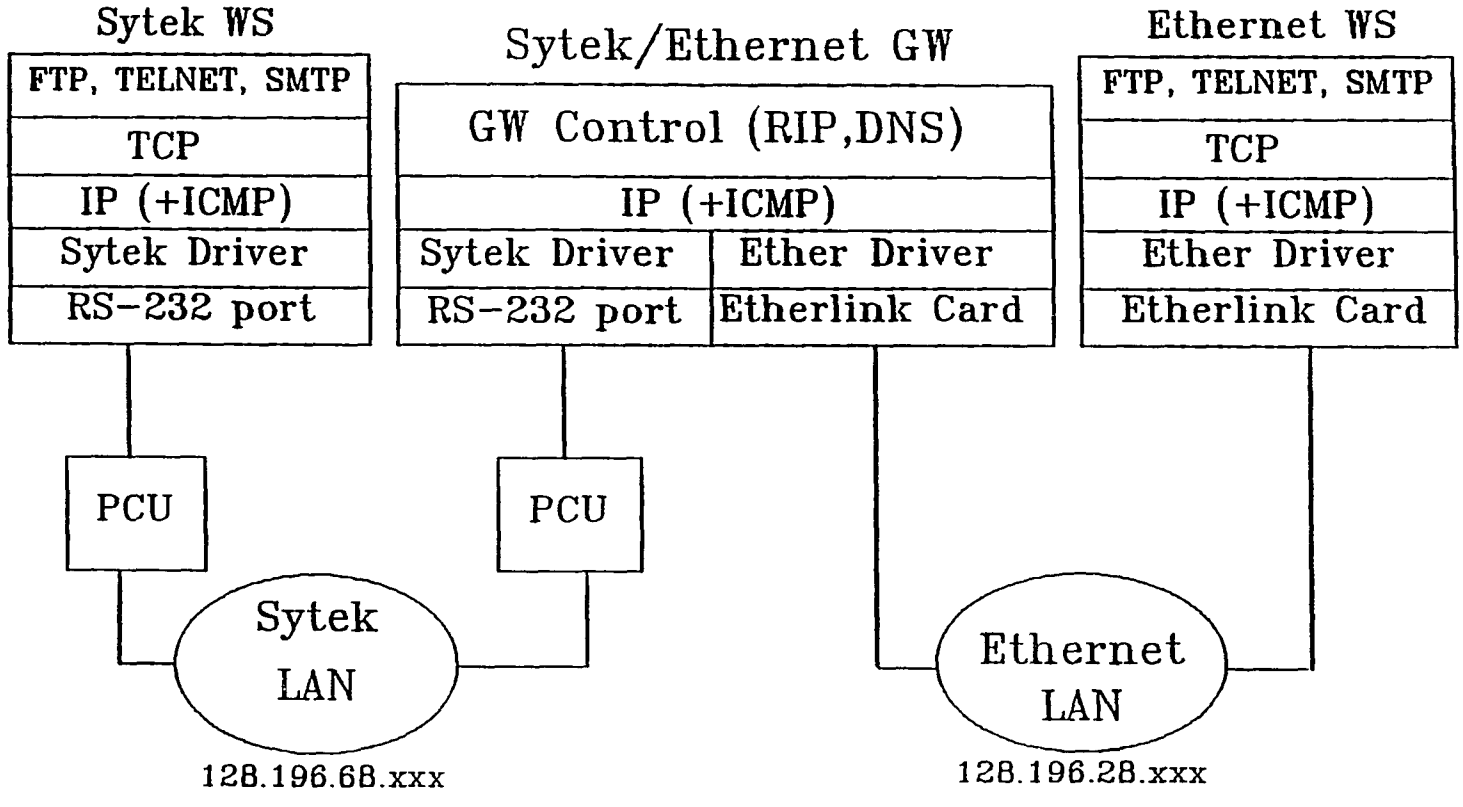


Figure 1.11 - Development System III

KA9Q) was chosen as the foundation for the prototype gateway. NOS KA9Q was originally developed for TCP/IP over packet radio, but it was designed to be flexible enough for further development. According to the KA9Q users' manual:

> "KA9Q was designed to be a robust platform on which to build real networks. To this end, the core protocols have been extensively tested and verified. In addition, great emphasis has been placed on increasing the portability of the software, supporting more and more hardware interfaces, and making it possible to use as many networking technologies as possible." [10]

It should be noted that there exist two distinct versions of the KA9Q package: the earlier, non-NOS version, and the newer, NOS version. The differences between these are primarily architectural, and should only concern developers (who may read Appendix B for further information). Unfortunately, the only available documentation is for the older version of KA9Q. The NOS version is undergoing constant revision, so it has yet to be documented. This made KA9Q seem, at first, to be less desirable, but in the end it was selected for many reasons: The code itself is very well written and internally documented, it is highly modular, and it has a wide base of users and developers in the Internet community.

NOS is an acronym for "Network Operating System". It is an OS kernel which KA9Q layers on top of MS-DOS during

run time, which gives programs access to the multi-tasking hardware in the 386-class PC. It is the vehicle whereby KA9Q exhibits multi-tasking while running on top of MS-DOS. Further details can be found in Appendix B.

KA9Q provides support for the IP, ICMP, TCP, UDP, FTP, SMTP, and Telnet protocols from the basic Arpanet set. In addition, the ARP protocol is available for address resolution on Ethernet interfaces. Support is also provided for SLIP (Serial Line Interface Protocol), which allows IP transmission over an asychronous serial line (which is similar to a Sytek session in data mode). KA9Q also implements RIP (Routing Information Protocol), the interior routing protocol made popular by Berkeley Unix, which is being used as the Interior Gateway Protocol (IGP) at UA. All gateways must participate in the IGP in order to maintain accurate routing tables [29]. In accordance with the latest Internet requirements, a Domain Name Service (DNS) client is available to consult nameservers for resolution of fully qualified hostnames (i.e., athens.ece.arizona.edu).

1.2.6 The UA Sytek Network's Relationship to The Internet

In preparation for the integration of the Sytek Network subnetwork into the Campus Internet it was necessary to obtain a subnetwork address space. Sytek hardware addresses are up to 20 bits long (i.e., the PCU unit ID is 16 bits and the port ID can be one bit (for 20/100s) or four bits (for 20/200s)). However, subnetworking on campus is done with Class C subnetwork addresses, which reserve only eight bits for the host-id. This means we cannot create a one-to-one mapping between all possible Sytek hardware addresses and a UA subnetwork address space. A very similar problem occurs with Ethernet subnetworks (Ethernet hardware addresses are 48 bits in length). Ethernet solves this problem via the Address Resolution Protocol (ARP) (c.f. RFC 826, "An Ethernet Address Resolution Protocol", [24]). As we shall see, the Sytek driver must implement a similar protocol to resolve IP addresses into Sytek hardware addresses. The class C subnetwork address space assigned to the UA Campus Sytek Network is 128.196.68.xxx.

The previous explains where the Sytek subnetwork fits in to the campus network architecture in terms of IP addressability. In terms of physical connectivity, the prototype gateway will attach to the ece-net (shown in Figure 1.3) - (domain name: ece.arizona.edu, subnet address

space: 128.196.28.xxx) with hostname "gengw.ece.arizona.edu" and IP address of 128.96.68.26. Of course, the gateway will be a multi-homed host, having an IP address on the Sytek subnetwork as well (128.196.68.2). The Sytek domain has been assigned the name "ece-sytek.arizona.edu", thus a host on the Sytek Network called "dino" would have the Internet hostname "dino.ece-sytek.arizona.edu".

The Internet is structured under the connectionless network paradigm. This means that each datagram traverses the internet independently, providing the more robust communication necessary for a large network. Networks participating in the Internet must conform to this paradigm. The Sytek Network (at least from our vantage point atop SMP) is connection-oriented. Thus we must "hide" the Sytek sessions used for IP datagram transmission. What is meant by this is that upon receiving an IP datagram, our Sytek driver must establish the proper session for it's delivery.

Datagrams received for IP by a network driver from the same user in a short period of time are often destined for the same host. Thus, it was decided that Sytek sessions should not be established on a per-datagram basis; The overhead from repeated establishment and termination of a session between the same two hosts should be avoided.

Instead, sessions established for an IP transmission are kept active for future datagrams. If a session is idle for too long, a timer will expire and cause that session to be terminated.

It bears repeating that establishing the Sytek Network as a subnetwork of the UA Campus Internet also makes it a real part of The Internet. Put simply, Sytek users will be able to FTP, or Telnet anywhere in The Internet (and vice versa); the prototype gateway gives Sytek users access to information on a global scale.

CHAPTER 2

GATEWAY SYSTEM DESIGN

This chapter discusses the prototype gateway's design. The overall architecture is discussed in Section 2.1 in terms of hardware used, protocols designed, and software designed and developed. The software design is discussed in greater detail in Section 2.2, where the functions used to implement the protocols are actually discussed.

2.1 Overall Architecture

2.1.1 Hardware Structure

The hardware for the prototype gateway is made up entirely of off-the-shelf components; no custom hardware design was necessary to implement the gateway system. Instead, the gateway's functional requirements were examined and appropriate existing hardware was selected and configured to meet these requirements.

As previously mentioned, the prototype gateway was developed for a 386-class PC AT clone. The gateway needs interfaces to both constituent LANs, so hardware to interface each is required. Specifically, to connect to the Sytek Network the gateway must have an available serial

port, a LocalNet 20/100 PCU and a serial cable to connect them. To interface the Ethernet LAN, the gateway has a 3Com 3C503 Etherlink Card, transceiver cabling and a TCL card (this card, made by TCL Inc., allows the concentration of many Ethernet transceiver cables into one shared Ethernet transceiver; for our purposes it is the same as if the gateway had it's own transceiver tap into the Ethernet). The gateway also has a printer card and an Epson FX-850 dot-matrix printer.

Another 386-class PC AT clone was purchased for this research, to act as the workstations shown in Figures 1.9 and 1.10 (c.f. Section 1.2.4). This system needed to act alternately as a Sytek and Ethernet workstation, so it required the same hardware used by the gateway to interface both LANs.

The NOS KA9Q package comes with support (via a "packet driver", c.f. Section 4.2.1.4) for the 3C503 Etherlink card. For a detailed description of this card, the reader should consult the "EtherLink II Adapter Guide", [1].

A more detailed description of the LocalNet 20/100 PCU is available in the "LocalNet 20 Reference and Installation Guide", [38]. As it is the hardware upon which our network driver rests, an overview of some of the PCU's important

features is given here.

There are four PCU commands of immediate interest to us: CALL, DONE, STATUS, and HELP. The CALL command initiates a session to the designated PCU port. Establishment of a session elicits the following response:

CALL COMPLETED TO <Sytek Network Address>

The format is "CALL <unitId>[,portId]" where "unitId,portId" is a fully qualified address and "unitId" is a rotary address (c.f. Section 4.2.2 and the "LocalNet 20 Reference and Installation Guide", [38]).

The DONE command causes the PCU to terminate the specified session. When closure is successful, the PCU generates the message:

SESSION <X> CLOSED TO <Sytek Network Address>

The format is "DONE [sessionNo]", where "sessionNo" specifies the session to close, or "DONE<CR><CR>" which closes the current session.

The STATUS command displays the current values of the port attributes and the status of any established sessions. Figure 2.1 represents the output from a status display. The first four digits of the unit number give the unitId of the PCU. The next number gives the portId of the PCU port whose status is being reported.

```
          LOCALNET 20/100      0D00 0000 16  V2.2.5+

UNIT         E905,0   BAUD          19200   IDLE           5
GROUP        A        PARITY        NONE    EOM COUNT      0
CHANSP       300      STOPS         1       EOM CHARACTER  NONE
LOCATION     12,238   AUTOBAUD      OFF     NEWLINE        0D
COMMAND      1B,1B    DCD CONTROL   OFF     EXPAND         NONE
LISTEN       ON       DSR CONTROL   OFF     XON            11
PRIVILEGE    ON       DTR CONTROL   OFF     XOFF           13
MAXSESSION   1        ECHO          OFF     FLOW           NONE
PCALL        OFF      QUIET         OFF     TIMEOUT        0
PUNIT        0000,0

SESSIONS -   NONE
```

Figure 2.1 Sample PCU Status Output

If a PCU user wishes to change a parameter setting, he/she simply types the parameter name followed by the desired setting. For example, if we wished to change the baudrate of a PCU to 1200, we would simply type "BAUD 1200<CR>" at the PCU prompt; Any parameter may be set by using the command which shares it's name.

The HELP command displays a list of enabled commands. The list does not include currently disabled commands. There is no indication of the parameters which the displayed commands may require.

Commands which are not required may be selectively disabled at a PCU, preventing their use by the PCU's user. Privileged PCUs (those with PRIVILEGE set to "ON") may override this disablement [38]. Commands are enabled by using the ENABLE command (i.e., to enable the CALL command, a user would type "ENABLE CALL" at the PCU prompt).

There are a few PCU parameters which need to be set to appropriate values for the Sytek driver. If they are not, the Sytek driver will properly initialize the PCU (and restore it's original state when finished). The parameters of interest are given here with their desired settings: "COMMAND 1B,1B", "LISTEN ON", "MAXSESSION 1", "PRIVILEGE OFF", "ECHO OFF", "QUIET OFF", "FLOW NONE".

2.1.2 An Overview of SLIP (Serial Line Interface Protocol)

Sytek sessions act in a manner which is very similar to point-to-point serial links. Generally, TCP/IP implementations transfer IP datagrams via a serial connection using SLIP, the Serial Line Interface Protocol. Our Sytek driver is architecturally founded on SLIP, so a brief overview is provided here. For a slightly less brief overview, the reader is referred to RFC 1055, "A Non-Standard for Transmission of IP Datagrams Over Serial Lines: SLIP", [31].

Quite simply, SLIP is a packet framing protocol: SLIP defines a sequence of characters that frame IP packets on a serial line, and nothing more. The SLIP protocol defines four special characters: FR_END (Frame End), FR_ESC (Frame Escape), T_FR_END (Transposed Frame End), and T_FR_ESC (Transposed Frame Escape). They are defined as hexadecimal C0, C3, C4 and C5 respectively. Notice that these have their high bit set; they are not ASCII characters.

To send a packet, a SLIP host sends an FR_END, and then starts sending the data in the packet. If a data byte is the same code as the FR_END character, a two byte sequence of FR_ESC and T_FR_END is sent instead. If it the same as

an FR_ESC character, an two byte sequence of FR_ESC and T_FR_ESC is sent instead. When the last byte in the packet has been sent, an FR_END character is then transmitted.

The first FR_END is sent to flush any erroneous bytes which have been caused by line noise. In the normal case, the receiver will simply see two back-to-back FR_END characters, which will generate a bad IP packet. If the SLIP implementation does not throw away the zero-length IP packet, the IP implementation certainly will. If there was line noise, the data received due to it will be discarded without affecting the following packet.

A flowchart of a SLIP packet send is given in Figure 2.2. A SLIP packet receive is given in Figure 2.3. Note that they depend on two functions, send_char() and recv_char(), which send and receive a single character over the serial line.

2.1.3 Protocol Structure

During the second development phase (c.f. 1.2.4), the Sytek driver's (c.f. Figure 1.10) sub-protocol structure was developed. Specifically, the functions needed in the driver were analyzed and a rough sub-protocol scheme was designed. Throughout this phase, the sub-protocol structure was

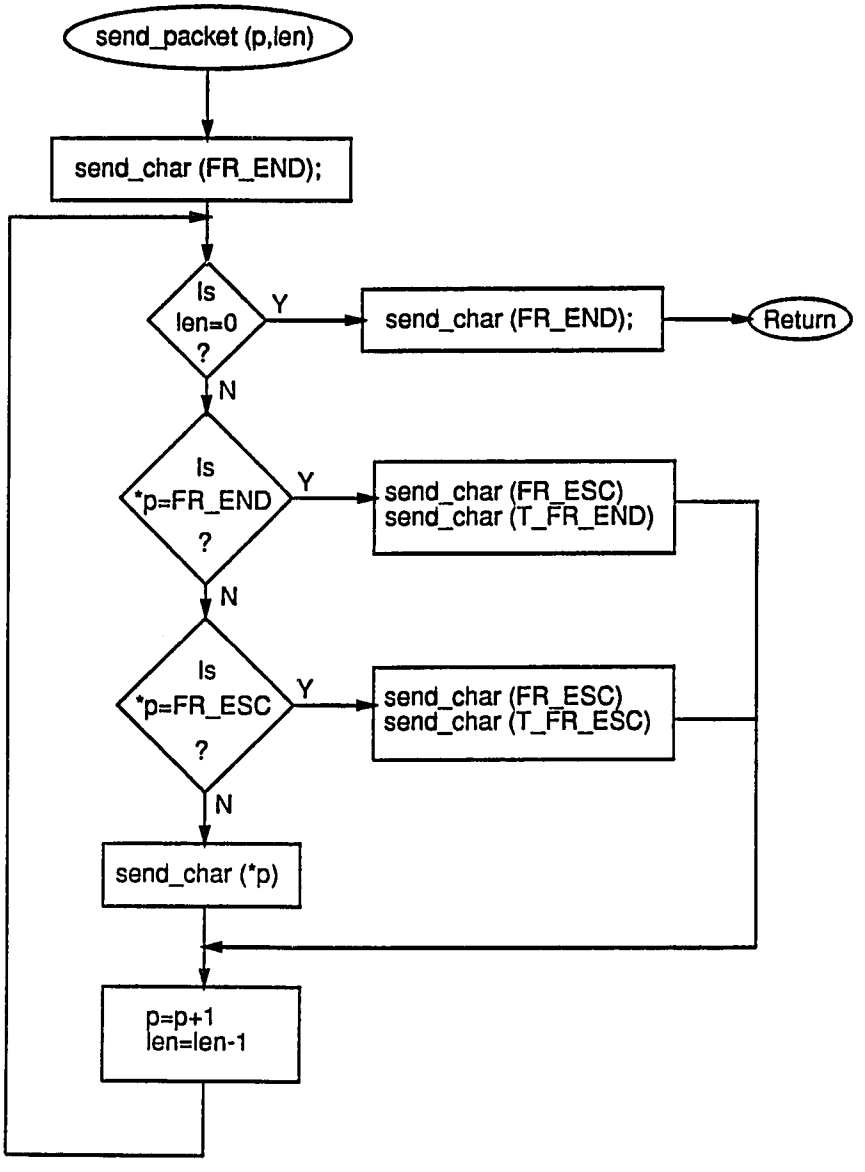


Figure 2.2 - SLIP send_packet() Function

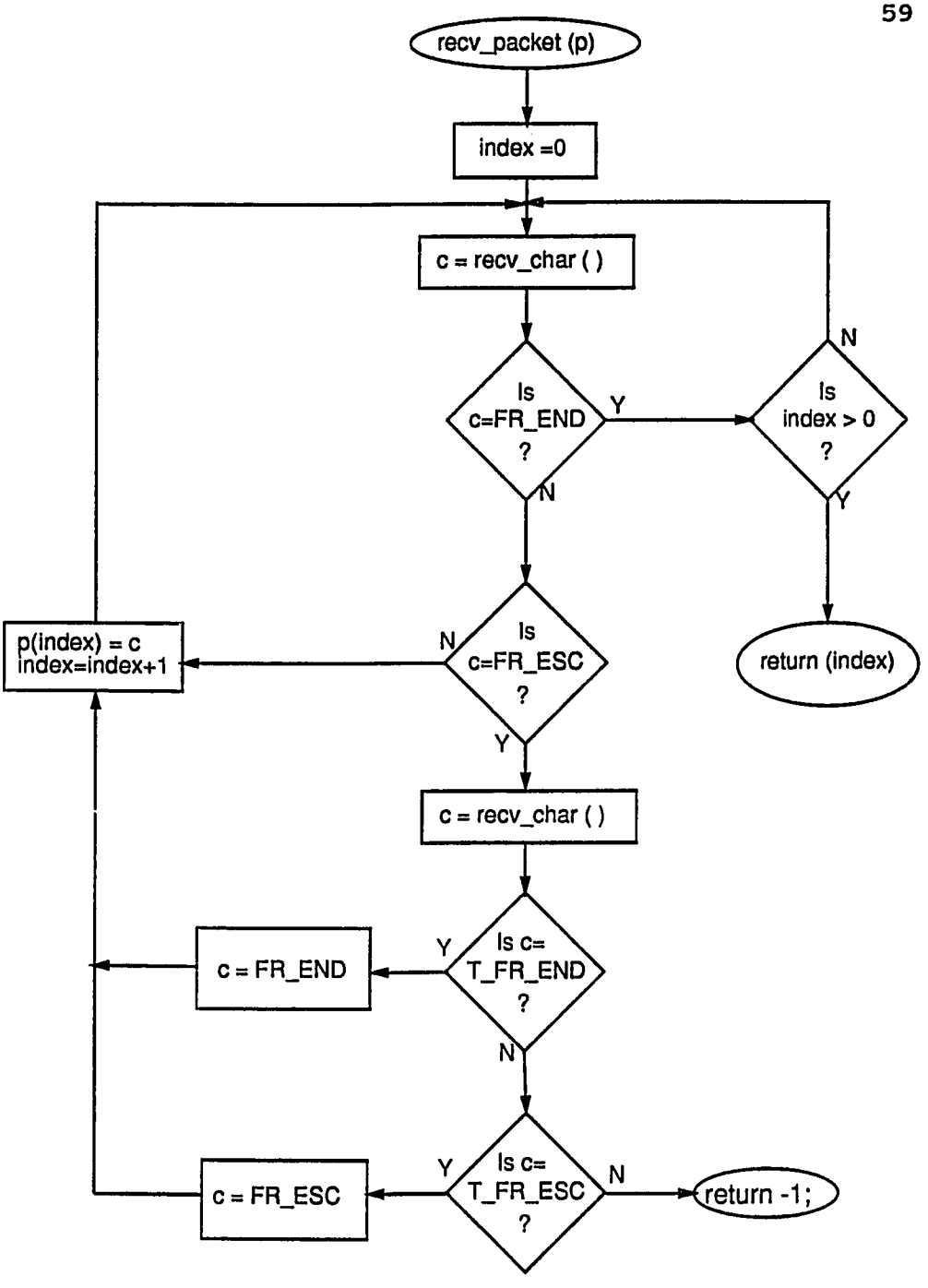


Figure 2.3 - SLIP recv_packet() Function

revised and refined until it reached it's final form, shown in Figure 2.4.

2.1.3.1 An Overview of the Sytek Driver

The heart of the Sytek driver is the module labelled "Sytek_Main". This module provides an interface to IP, facilitating the sending and receiving of datagrams. To achieve the first task of the second development phase (c.f. 1.2.4), Sytek_Main provided a transparent interface to SLIP. This configuration is sufficient for the transfer of datagrams over a serial connection.

The second task requirements were met by simply insuring that the PCU command character (ASCII Escape) never appeared in the data stream. This was done by using a modified SLIP module (hereafter referred to as SLIP') that would perform additional character stuffing for the PCU command character for Sytek interfaces only; datagrams for genuine SLIP interfaces will use standard SLIP encapsulation.

To achieve the third task, routines to initialize and restore the PCU state were written. These routines are outside of the concept of protocol layering, to which this section is devoted. They are discussed in more detail in Section 2.2.2.5.

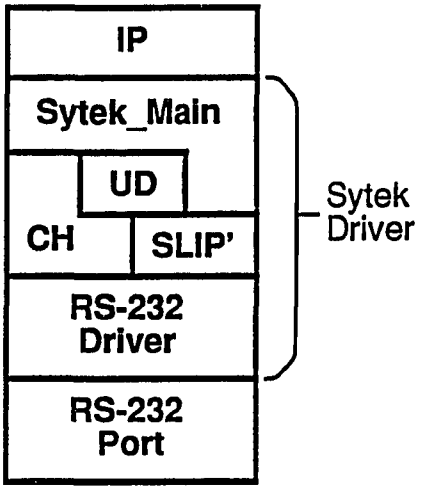


Figure 2.4 - Sytek Protocol Stack

The fourth task required a simple module to perform IP address resolution via table lookup. This is the skeleton of the "Unit Discovery" module ("UD" in Figure 2.4), which was completed in the fifth task. The principal work for this task, however, involved adding the ability in the driver to manage Sytek sessions for the transport of IP datagrams. This functionality was provided by the "Channel Handler" module ("CH" in Figure 2.4).

Finally, during the fifth task a more sophisticated Sytek address resolution protocol was designed and developed. This protocol is shown in Figure 2.4 as "UD", the Unit Discovery module. The UD module can be queried by Sytek_Main to resolve an IP address into a Sytek hardware address.

The module labelled "RS-232 Driver" in Figure 2.4 simply provides single character I/O for SLIP' (i.e. send_char() and recv_char() in Figures 2.2 & 2.3) and the Connection Handler (i.e., commands and their responses).

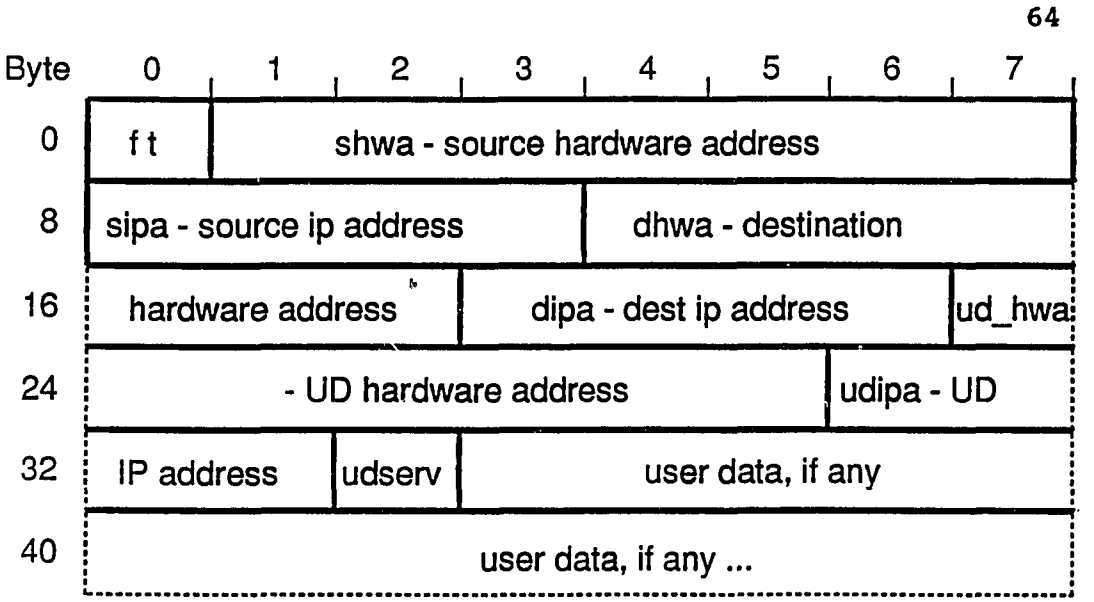
2.1.3.2 Sytek Frame Format

Unlike SLIP interfaces, which only transfer data frames, Sytek interfaces must send and receive many different kinds of frames (e.g., data frames, UD request and response

frames). As part of the Sytek Protocol specification, it was necessary to define a "Sytek Header" which would allow the demultiplexion of different data streams in the Sytek driver. Please note that this header and any frame data will be SLIP' encapsulated before it appears "on the wire", but we define the frame format here in terms of its pre-encapsulation appearance; After SLIP' encapsulation, the frame format is contents-specific. The Sytek Frame Format is shown in Figure 2.5.

The first 34 bytes comprise the Sytek frame header. It was decided that for simplicity, there would be only one type of frame header, with all the fields necessary to be used for any specific frame type. For instance, data frames carrying IP datagrams would not have a significant value in the udhwa field. This method is intended as an intermediate approach only; later development should include multiple Sytek frame formats to decrease wasted bandwidth.

After the header (from byte 35 on) there may be data (assumedly an IP datagram). There is no set limit on how large this data field can be; SLIP' encapsulation delimits the beginning and ending of the Sytek frame for us, so the frame recipient can easily determine the data field's length.



field	bytes	field name	value	meaning
1	0	frame type	0	SYTEK_DATA_FRAME
			1	SYTEK_UD_REQUEST
			2	SYTEK_UD_RESPONSE
			3	SYTEK_CONNECT
			4	SYTEK_DISCONNECT
2	1 - 7	source hardware addr	-	
3	8 - 11	source IP addr	-	
4	12 - 18	dest hardware addr	-	
5	19 - 22	dest IP addr	-	
6	23 - 29	UD hardware addr	-	
7	30 - 33	UD target IP addr	-	
8	34	UD server indicator flag	0	UD_SERVER
			1	UD_NONSERVER
			2	UD_SERVER_MAYBE
9	35 -	User Data (optional field)	-	

Figure 2.5 - Sytek Frame Format

The second, fourth and sixth fields hold Sytek hardware addresses. Each reserves seven bytes, with each digit of the Sytek address getting a byte. This was done for simplicity and debugging purposes. Future research directed towards a production quality driver should consider reducing these. Fields 1 through 5 are self-explanatory. The last three fields in the header are for use by the Unit Discovery module to facilitate address resolution. A quick outline of the five frame types follows:

1. SYTEK_DATA_FRAME - These carry IP datagrams in their data field (byte 35 on). They do not have significant values in any header fields, except for "ft".

2. SYTEK_UD_REQUEST - These are frames sent by a general Sytek host to a "UD" server. It contains the IP address (in the "udipa" field) of a "target system" for which the requestor needs a Sytek hardware address.

3. SYTEK_UD_RESPONSE - These are frames sent by a "UD" server to a Sytek host. It contains the Sytek address (in "udipa") which corresponds to the "target system" (specified by the IP address in "udhwa"). The server also fills in the "ud_serv" field which identifies the target system as a UD server or non-server (or "UD_SERVER_MAYBE" if it doesn't know).

4. SYTEK_CONNECT - These are frames sent by a system which is establishing a connection with the recipient. They may contain IP datagrams in their data field.

5. SYTEK_DISCONNECT - These are frames sent by a system to a currently connected neighbor announcing it's intention to close the current Sytek session between them.

2.1.3.3 Sytek_Main Module

This section provides the functional description of the user interface to the Sytek_Main module. The Sytek_Main module implements the following three functions:

sytek_send (buf, iface, gateway, prec, del, tput, rel)

buf = buffer pointer
iface = driver interface
gateway = IP address of destination system
prec = precedence
del = delay
tput = throughput
rel = reliability

This function sends an IP datagram over a Sytek interface to a remote Sytek system. The data to be sent is indicated by "buf", the interface to be used by "iface", and the IP address of the next-hop destination by "gateway". The last four parameters (prec, del, tput and rel) indicate the transport characteristics desired by the user. Currently they are ignored, but they have been included here for future expansion (and compatibility with KA9Q).

In order to send the datagram, sytek_send resolves the IP address (indicated by "gateway") into a Sytek hardware address by a call to the UD module. If UD indicates no immediate resolution is possible, the datagram is dropped (it is assumed that if UD can resolve the address via remote consultation, future re-transmissions of this datagram will succeed). If UD was able to immediately resolve the

address, a Sytek frame header is constructed for this datagram. The Sytek data frame is then delivered by a call to sytek_raw.

sytek_raw (sy_hdr, datagram, iface)

sy_hdr = Sytek frame header
datagram = datagram to be framed and transmitted
iface = driver interface

This function prepends the Sytek frame header to the datagram, yielding a Sytek frame. It then consults the CH module to establish the appropriate session (if it isn't already existent) to the destination system, using the given interface. The Sytek frame is then encapsulated SLIP'-wise, and delivered.

sytek_recv (iface, buf)

iface = driver interface
buf = buffer pointer

This function is used to process incoming frames (indicated by buf) from a Sytek interface. The Sytek header is stripped off and the frame type is examined. Based upon the frame's type, it is sent to the appropriate module; sytek_recv's main function is the demultiplexion of incoming Sytek frames.

Frames with type SYTEK_DATA_FRAME hold IP datagrams, which are sent to the IP module for routing. Unit Discovery frames (type SYTEK_UD_REQUEST or SYTEK_UD_RESPONSE) are sent

to the UD module for further processing. Frames with type SYTEK_CONNECT cause sytek_recv to notify the CH module of the new connection. Frames with type SYTEK_DISCONNECT cause sytek_recv to notify the CH module of the disconnection.

2.1.3.4 SLIP' Module

This section provides the functional description of the user interface to the SLIP' module. The functions described here are part of the KA9Q package. The have been slightly modified for use with the Sytek driver. The SLIP' module implements the following four functions:

slip_send (buf, iface, gateway, prec, del, tput, rel)

```
buf = buffer pointer
iface = driver interface
gateway = IP address of destination system
prec = precedence
del   = delay
tput = throughput
rel = reliability
```

This function sends an IP datagram over a SLIP interface to a remote system. The data to be sent is indicated by "buf", the interface to be used by "iface", and the IP address of the next-hop destination by "gateway". The last four parameters (prec, del, tput and rel) indicate the transport characteristics desired by the user. Currently they are ignored.

Due to the simplicity of it's "subnet" (i.e., a point-to-point link) SLIP has no link-level headers or any such protocols which must be layered on top of a "raw" SLIP transmission. For this reason, slip_send is essentially nothing more than a transparent call to slip_raw.

slip_raw (iface, buf)

iface = driver interface
buf = buffer pointer

This function encapsulates the frame specified by "buf" SLIP-wise (with a call to slip_encode). The resulting data is then transmitted over the SLIP interface specified by "iface".

slip_encode (buf, dev)

buf = buffer pointer
dev = asynchronous device ID

This function encapsulates the data specified by "buf" for point-to-point transmission. "dev" is used to determine if this data is destined for a SLIP or Sytek interface. SLIP data is encapsulated SLIP-wise, Sytek data is encapsulated SLIP'-wise.

"SLIP-wise encapsulation" means the data encapsulated in the standard way according to the SLIP protocol specification (c.f. Section 2.1.2). "SLIP'-wise encapsulation" is basically the same, except it stuffs two

more bytes: ASCII Escape (0x1B) and it's eight-bit "equivalent" (0x9B).

The reason for this is that the PCU command escape sequence is two ASCII Escapes. If this were to occur in the data stream by chance, the PCU would go into command mode in the middle of a frame transfer, corrupting the frame and causing the PCU to go into a non-deterministic state from which the interface software could not easily recover.

The reason we prohibit any ASCII Escapes in our frames (instead of just pairs of Escapes) is that under heavy load, if the PCU is asked to transfer an Escape followed by some other character, it often delivers these characters out of order. Finally, the reason we must stuff the eight-bit Escape equivalent is that ASCII characters are seven bits in length; the PCU hardware which searches for the command escape characters ignores each bytes' high bit. Thus, a pair of 0x9B's will also put the PCU in command mode.

slip_decode (buf, dev)

buf = buffer pointer
dev = asynchronous device ID

This function decapsulates the data specified by "buf". It is the analog to slip_encode. "dev" is used to determine if this data is from a SLIP or Sytek interface. SLIP data is decapsulated SLIP-wise, Sytek data is encapsulated SLIP'-

wise.

asy_rx (dev)

dev = asynchronous device ID

This function is the code run by the byte-oriented asynchronous receive process. Each character received is processed according to the current state of the PCU. If the PCU is passive (in command mode, but no local commands have been issued), data we receive is processed appropriately (i.e., we must listen for remotely initiated connections). If we're awaiting the response to a PCU command, data is put in a command response shared memory buffer. If the PCU is in data mode, then bytes are assumed to be part of a SLIP'-encapsulated frame. When a FR_END is received, the buffered data is sent to either sytek_recv or slip_recv for processing, depending on the associated interface type.

2.1.3.5 Channel Handler Module

This section provides the functional description of the user interface to the Channel Handler module. The Channel Handler module implements the following four functions:

sych_conn_request (src_hwa, dest_hwa, iface, buf)

src_hwa = source hardware address
dest_hwa = destination hardware address
iface = driver interface
buf = buffer pointer

A user calling this function wants to deliver a Sytek frame ("buf") to a Sytek destination ("dest_hwa") using a specified interface ("iface"). If the requested connection exists, "sych_conn_request" returns TRUE. Otherwise, it returns FALSE, indicating that the caller should abort it's delivery attempt. "sych_conn_request" attempts to establish the appropriate connection and deliver the frame when it is established.

sych_conn_report (src_hwa, dest_hwa, iface, state)

src_hwa = source hardware address
dest_hwa = destination hardware address
state = connection state

This function is used to "report" a connection to the CH module. Creates an entry in the Channel Handler database, and starts the timeout timer for this session. This function can be used to record both locally and remotely initiated connections.

sych_disc_request (dev, iface, entry)

dev = asynchronous device ID
iface = driver interface
entry = connection table entry

This function is called to request a connection termination. It sends a disconnect frame to the other side, issues the done command to the PCU and removes the connection from the CH database using sych_disc_report.

sych_disc_report (iface)

iface = driver interface

This function removes a connection from the CH database.

2.1.3.6 Unit Discovery Module

UD implements an address resolution protocol inspired by ARP, the address resolution protocol used by Ethernet. Because Sytek is connection-oriented, ARP (a broadcast-oriented protocol) could not be used. Instead, UD was designed to consult UD servers (systems which keep complete Sytek Domain tables) when an IP address needs resolving. This approach is certainly less robust than ARP, but Sytek's connection-oriented nature makes it unavoidable.

This section provides the functional description of the user interface to the Unit Discovery module. The Unit Discovery module implements the following three functions:

syud_resolve (ipaddr, iface)

ipaddr = IP address to be resolved
iface = driver interface

This function is called to resolve an IP address into a Sytek hardware address. If the IP address is in our local UD database, the associated hardware address is returned. If local lookup fails, a SYTEK_UD_REQUEST frame is assembled and sent to a known UD server. This a NULL is returned,

indicating to the caller that we are attempting to resolve the address (or we can't if we have no known servers). The caller should drop its frame.

syud_request_handler (sy_hdr, iface)

sy_hdr = Sytek frame header
iface = driver interface

This function is called by sytek_recv when a UD_REQUEST frame is received. The header is scanned for the IP address which we have been asked to resolve. If it is in our tables, we fill in the header's hardware address and "ud_serv" fields (which indicates if the resolved host is a UD server or not). If we can't resolve the IP address, the header's hardware address field is set to NULL. The header's frame type is reset to UD_RESPONSE, and the new header is prepended to a NULL length buffer and sent back out the specified interface, to the user requesting UD resolution.

syud_response_handler (sy_hdr, iface)

sy_hdr = Sytek frame header
iface = driver interface

This function is called by sytek_recv when a UD_RESPONSE frame is received. This frame is generally in response to a UD request we've issued, but that need not be the case (i.e., a Sytek host may "tell" another it's UD information

by sending an "unsolicited response"). Regardless, the UD information in the frame is used to update our UD database accordingly. If this is a failed response to a request we've issued, and there are more UD servers that we may consult, this routine attempts to resolve the address by sending out another UD request frame.

It should be noted that the UD functions have been designed such that multiple UD servers may exist (indeed, all hosts have the code necessary to be a UD server). This gives rise to the possibility of backup servers.

2.2 Software Design

This section discusses the Sytek driver software in detail. The actual functions designed to adhere to the protocol specification presented in Section 2.1.3 are given in flowchart form and any anonomolies are discussed. Data Structures necessary to these functions are also presented. This section does not discuss the KA9Q software in any detail. Interested readers should consult Appendix B.

2.2.1 Data Structures

As part of the Sytek driver design, a few essential data structures have been developed. This section identifies

these structures, what files they can be found in, and gives brief descriptions of each. The Sytek Driver was written in C; the data structures presented here are taken directly from the C source code of the driver.

The "sychud.h" include file contains definitions used by the CH and UD sub-protocol modules. In particular it defines the structure for internal representation of Sytek frame headers, as follows:

```
struct sy_header {
     char frame_type;

#define SYTEK_DATA_FRAME       0
#define SYTEK_UD_REQUEST       1
#define SYTEK_UD_RESPONSE      2
#define SYTEK_CONNECT          3
#define SYTEK_DISCONNECT       4

     char *src_hwa;
     int32 sipa;
     char *dest_hwa;
     int32 dipa;
     char *ud_hwa;
     int32 ud_ipa;
     char ud_server;

#define UD_SERVER              0
#define UD_NONSERVER           1
#define UD_SERVER_MAYBE        2

};
```

The fields in this structure correspond to the like-named fields shown in Figure 2.5 and discussed in Section 2.1.3.2. The fields "frame_type" and "ud_server" are

followed by "define" statements which enumerate their possible values. It should be noted that this is an internal representation only - Headers are converted between this format and the actual byte-oriented format prepended to datagrams by the "ntohsytek" and "htonsytek" functions (c.f. 2.2.2.5).

The "sychud.h" include file also defines the CH database as follows:

```
struct ch_entry {
     struct ch_entry *next;
     char *src_hwa;
     char *dest_hwa;
     struct iface *iface;
     int state;

#define CH_ATTEMPTING          0
#define CH_CONNECT_MADE        1
#define CH_CONNECT_HEARD       2
#define CH_SAY_GOODBYE         3
#define CH_DISCONNECTING       4

};

struct ch_entry *CH_List;
```

The "ch_entry" structure defines the format for an element in the CH linked list, which is defined as "CH_List". The "next" field points to the next element in the linked list. The session initiator's Sytek address is given in "src_hwa", and the session recipient's is given in "dest_hwa". "iface" is the network interface associated with this connection and "state" identifies the current

state of this "connection".

The reason for five states is that the CH database doesn't have to just record any connections we have (CH_CONNECT_MADE and CH_CONNECT_HEARD), but also connections we are attempting to establish (CH_ATTEMPTING), as well as those we are in the process of terminating (CH_SAY_GOODBYE and CH_DISCONNECTING).

One may ask how our connections need only five states, while TCP sessions can have eleven. Very simply, this is because TCP must render reliable service to it's user using IP, an unreliable datagram service. Thus TCP needs extra states to ensure that IP datagrams which bear TCP session establishment and termination information aren't "delayed duplicates" carrying outdated information. The SMP layer in the PCU provides accurate session management information, so CH needn't be as complex as TCP.

The "sychud.h" include file also defines the UD database as follows:

```
struct ud_entry {
     struct ud_entry *next;
     int32 ip_addr;
     char *sy_hwa;
     char ud_server;
     int state;
#define UD_PENDING  0
#define UD_VALID    1
};

struct ud_entry *UD_List;
```

The "ud_entry" structure defines the format for an element in the UD linked list, which is defined as "UD_List". The "next" field points to the next element in the linked list. The IP/Sytek address pair for each entry are held in "ip_addr" and "sy_hwa". The "ud_server" field identifies whether or not this element is a "UD server", and uses the same constants defined in sy_header. The "state" of each entry can be valid, or pending resolution.

The other major include file, "syuser.h", contains many definitions used to chronicle the condition of the network interface. In particular it defines the structure for internal representation of the PCU's current status, as follows:

```
struct PCU_STATUS {
     char model [8];
     char reset_no [4];
     char reset_type [4];
     char reset_addr [4];
     char buffers [4];
     char unit [4];
     char port [4];
     char group [6];
     char chansp [4];
     char channel [4];
     char lap [4];
     char command [6];
     char listen [4];
     char privilege [4];
     char maxsession [4];
     char pcall [4];
     char punit [12];
     char baud [6];
     char parity [4];
```

```
        char stops [4];
        char autobaud [4];
        char dcd [4];
        char dsr [4];
        char dtr [4];
        char echo [4];
        char quiet [4];
        char idle [4];
        char eom_count [4];
        char eom_character [8];
        char newline [4];
        char expand [4];
        char xon [4];
        char xoff [4];
        char flow [4];
        char timeout [4];
        char software [6];
        char helplist [340];
};
```

Each field of this structure corresponds to the PCU parameter of the same name. When the Sytek interface is first attached, the PCU's status is divined, and the results are stored in an occurrence of this structure.

The "syuser.h" include file also defines the following structure, PCU_STATE, which is basically the a Sytek interface control block:

```
struct PCU_STATE {
     char *command;
     int  command_length;
     char *response;
     int  expected_rl;
     int  actual_rl;
     int  mode;

#define SLIP_MODE              0
#define SYTEK_DATA_MODE        1
#define SYTEK_ACTIVE_1_MODE    2
#define SYTEK_ACTIVE_2_MODE    3
#define SYTEK_PASSIVE_MODE     4
```

```
#define PBSIZE 80

     struct proc *pass_daemon;
     char pass_buff[PBSIZE];
     int  pass_index;
     int  ready;

#define PCU_INITIALIZING      0
#define PCU_READY             1
#define PCU_RESTORING         2

     struct PCU_STATUS *status;
     struct stringll *restore_comms;
     struct proc *comm_daemon;
     struct timer conn_timeout;

#define CONNECTION_TIMEOUT 60

};
```

The first five fields declare the shared memories used to pass commands and their responses between a process wishing to issue a PCU command and the process designated to execute these commands (called the "PCU Command Daemon", c.f. Section 2.2.2.5 for more details).

The next field, "mode", identifies this interface's current transfer mode. SLIP interfaces will always be in "SLIP_MODE"; the remaining four states are for Sytek interfaces. When the PCU has no sessions (i.e., it's at the command prompt), the interface is in "SYTEK_PASSIVE_MODE". When there is an established session which is being used for data transfer, the interface is in "SYTEK_DATA_MODE". When a command has been issued and we

are awaiting a response, the interface is in "SYTEK_ACTIVE_1_MODE", unless that command was a done command, in which case the interface is in "SYTEK_ACTIVE_2_MODE". The first mode searches for a carriage return to signal the end of the command response. Done command responses have two carriage returns, so the second mode was necessary.

The next field, "pass_daemon" is a pointer to the "Passive Listener Daemon" process control block. This process runs in the background, listening for remotely initiated sessions when we are in "SYTEK_PASSIVE_MODE". The low-level I/O routines store passive data in the "pass_buff", using "pass_index" as an index. This is a shared memory buffer between the low-level I/O routines and the "Passive Listener Daemon".

The next field, "ready", identifies the PCU as either being ready for use, in the process of being initialized, or in the process of being restored. After this is the "status" field, which is a pointer to an occurrence of a "PCU_STATUS" structure, previously discussed. "restore_comms" is a linked list holding the commands necessary to return this PCU it's original state (i.e., before we intialized it for our purposes). "conn_daemon"

points to the process control block of the "PCU Command Daemon" process associated with this interface (c.f. Section 2.2.2.5 for more details). Finally, "conn_timeout" points to the timer structure associated with this interface's connection timeout timer.

The "syuser.h" include file defines an array of these "PCU_STATE" structures, one for each Sytek interface, as follows:

```
struct PCU_STATE pcu_state[ASY_MAX];
```

This array is dimensioned to the maximum number of possible asychronous network interfaces.

2.2.2 Sytek Driver Code

This section gives a detailed overview of the functions used to implement the Sytek driver. All major functions are described using flowcharts and any peculiar details are discussed. In particular, the functions specific to the four sub-protocol modules (Sytek_Main, CH, UD and SLIP') are presented in Sections 2.2.2.1 through 2.2.2.4. Section 2.2.2.5 describes the modules necessary to the Sytek driver's operation which are beyond the protocol specification.

2.2.2.1 Sytek_Main Module

The three main functions of the "Sytek_Main" module are sytek_send, sytek_raw, sytek_recv. The source code for these is in "sytek.c".

Figure 2.6 describes the sytek_send function. Note that if the gateway address is not locally resolvable, the datagram is discarded. It is assumed that syud_resolve is attempting to consult a remote server. Future transmissions (i.e., TCP retransmissions) should succeed if remote resolution succeeds.

Figure 2.7 describes the sytek_raw function. Note that if sych_conn_request indicates a failure, we return immediately. This happens if the requested connection isn't already established. "sych_conn_request" takes the frame to be delivered as a parameter, and will attempt to deliver it as soon as it can establish the proper connection. If it cannot, it will drop the frame itself. In any case, if it cannot immediately provide a connection to it's caller, it returns a "failure" status and forks a process to continue it's work, thus avoiding blocking it's caller.

Figure 2.8 describes the sytek_recv function. In the KA9Q architecture, it is "called" by being the source code

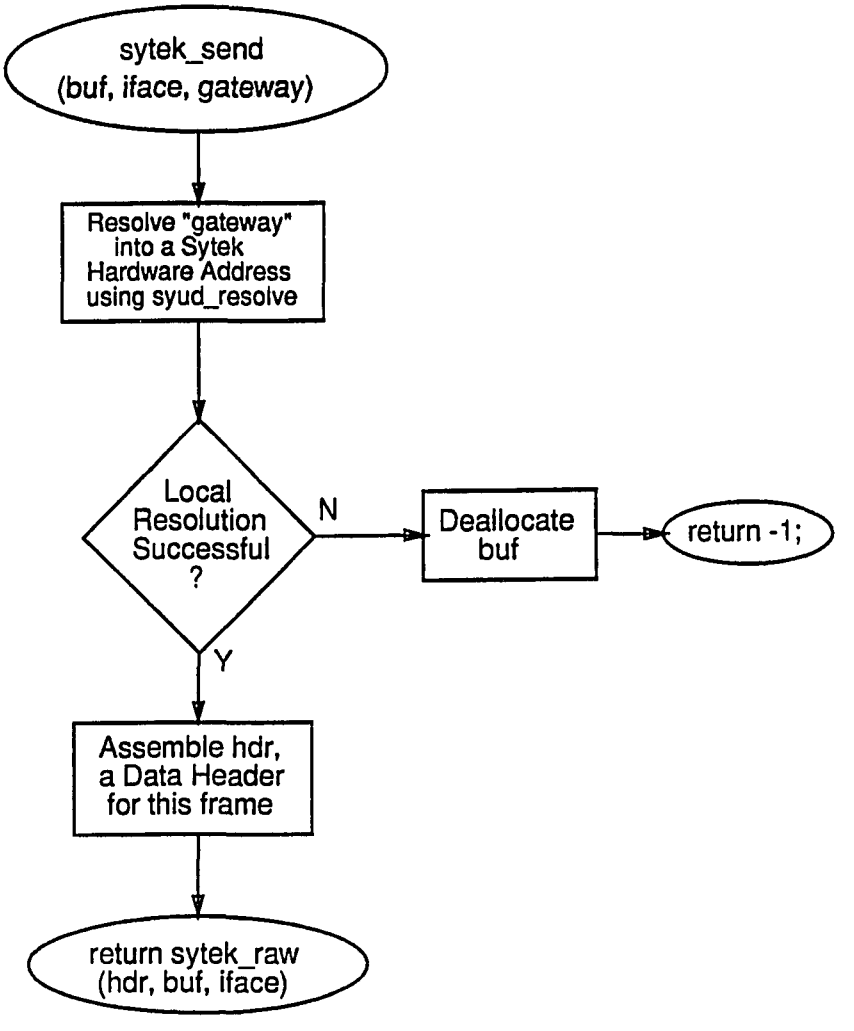


Figure 2.6 - sytek_send() Function

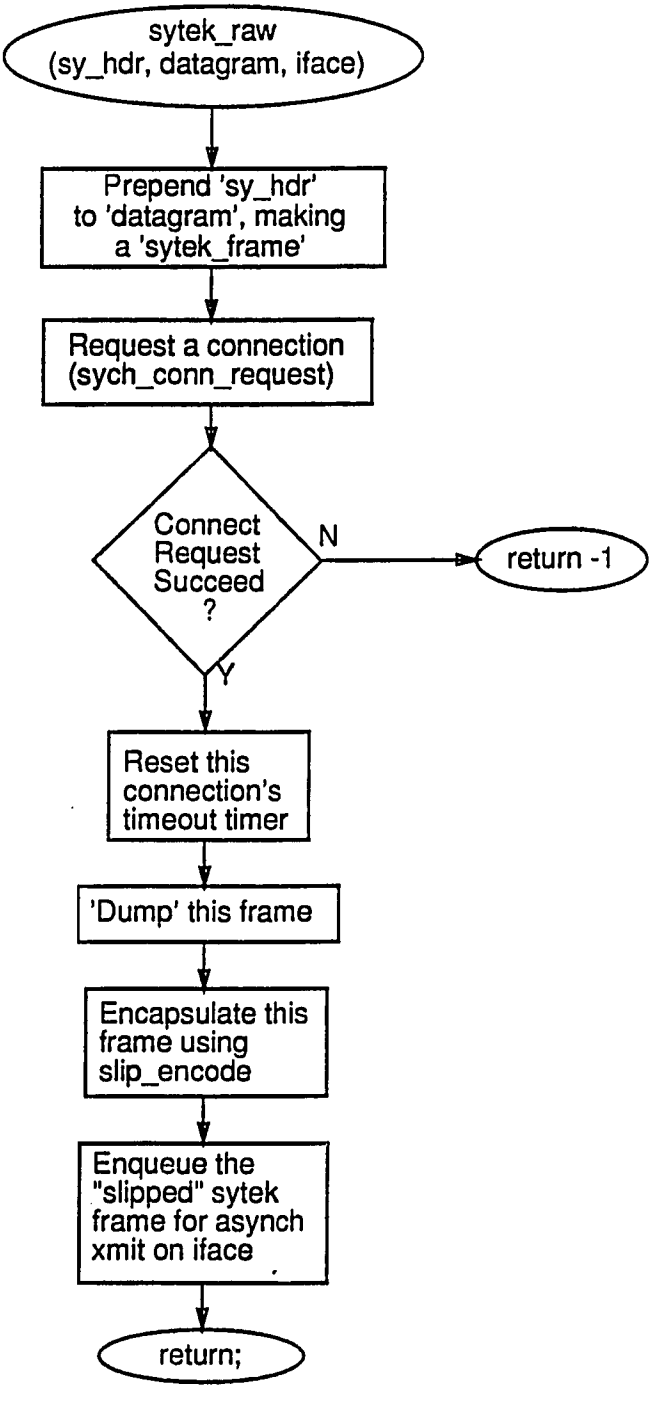


Figure 2.7 - sytek_raw() Function

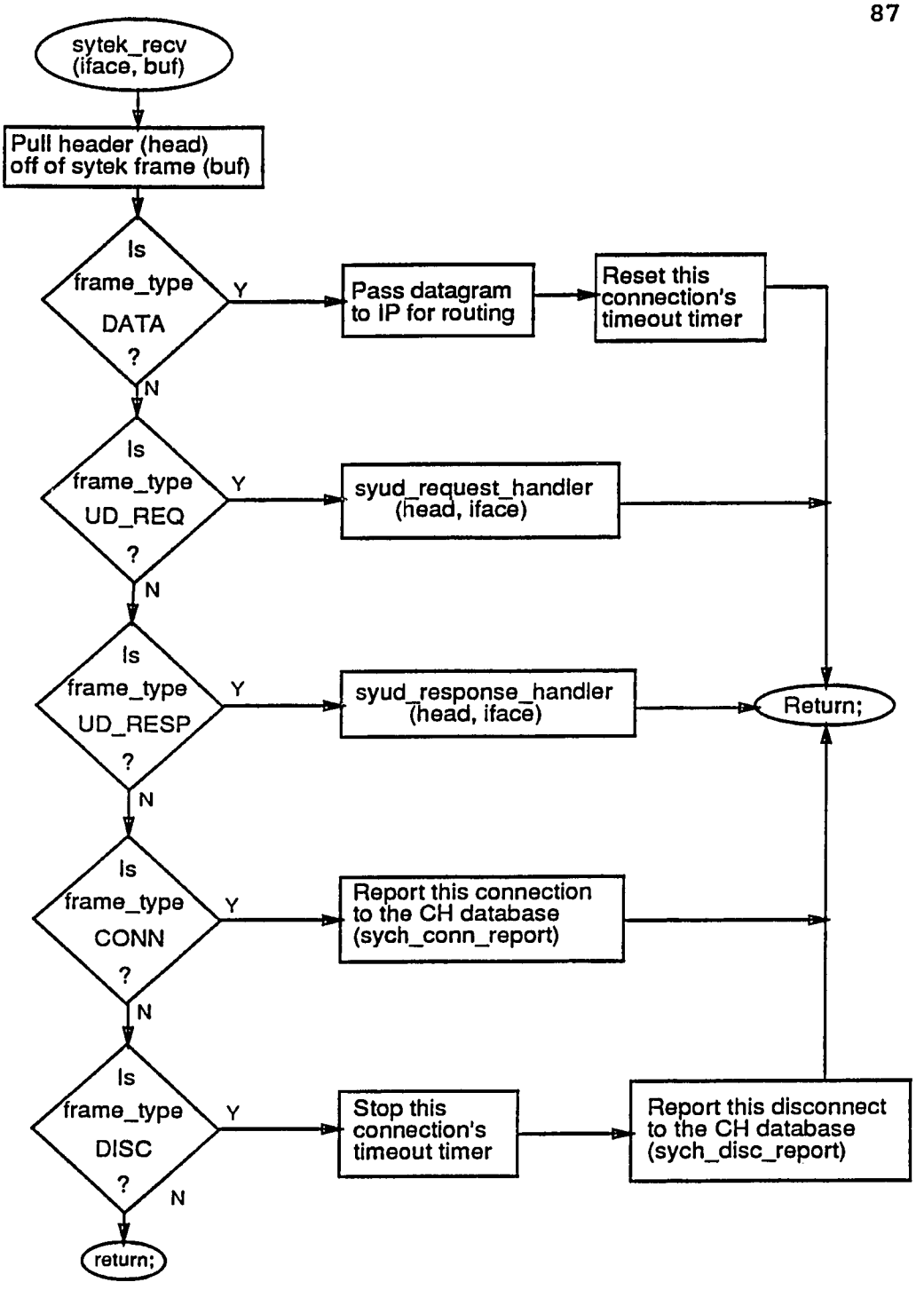


Figure 2.8 - sytek_recv() Function

for a new process which is created for each incoming frame on the associated Sytek interface. It is basically responsible for demultiplexing incoming frames to their appropriate modules according to each frame's type.

2.2.2.2 Connection Handler Module

The four main functions of the CH module are sych_conn_request, sych_disc_request, sych_disc_report, sych_conn_report. In addition to these, it was necessary to design some functions which these routines call, and one other, special function. This function, pcu_pass_daemon, is the source code for the "Passive Listener Daemon" process. The source code for all these is in "sytekch.c".

Figure 2.9 describes the sych_conn_request function. It is called by sytek_raw to determine if a useful session is available, and if not, to establish one. Note that if a connection is found in the CH database, but it isn't "valid", the frame is dropped, and a failure status is returned. This is to stop from informing sytek_raw we have a connection which may, in fact, be in the process of being attempted or terminated. If a the requested connection needs to be established, this routine makes a template entry in the CH database, and then forks a process to

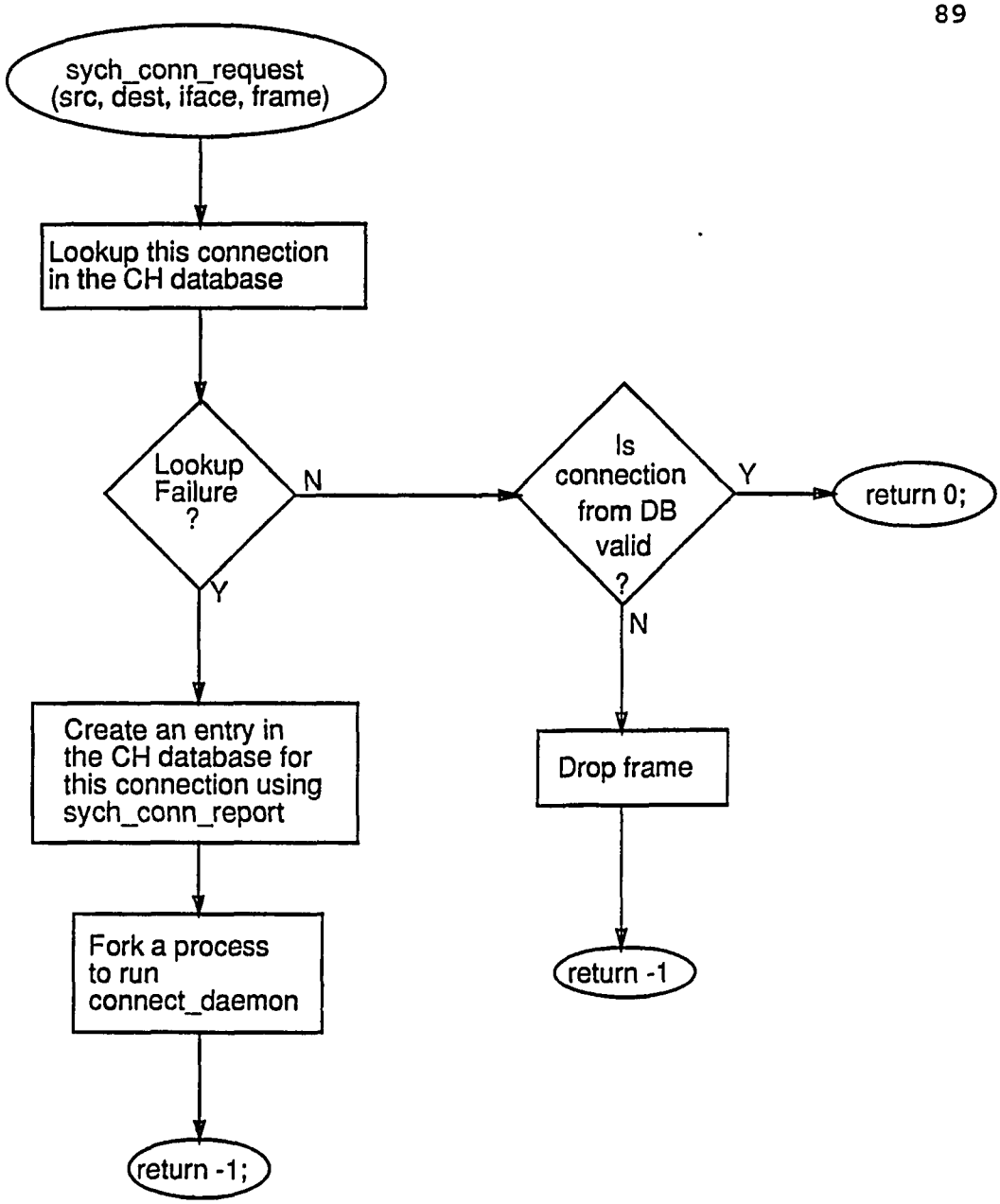


Figure 2.9 - sych_conn_request() Function

continue the session establishment, thus avoiding blocking sytek_raw.

Figure 2.10 describes the sych_conn_report function. This function is called by sych_conn_request (with state = CH_ATTEMPTING), and sytek_recv (with state = CH_CONNECT_HEARD). The "add_chlist_entry" routine mentioned in the last step is a simple function which adds an entry to the CH_List.

Figure 2.11 describes the connect_daemon function. This routine is the source code for the short-lived processes forked by sych_conn-request to create a new Sytek session. The "pcu_connect" routine mentioned here (and discussed below) issues the appropriate call command to attempt session establishment. "'Dump' this frame" means to dump the contents of this frame to the screen, if the user has enabled packet tracing on this interface.

Figure 2.12 describes the pcu_connect function. As previously mentioned, this routine interacts with the PCU (via the pcu_command_execute function, discussed in 2.2.2.5) to attempt a session establishment. The "pwait" function used here (and elsewhere) allows a process to wait at a given rendezvous point for another process, which will issue a "psignal", giving up the CPU while it waits. If no

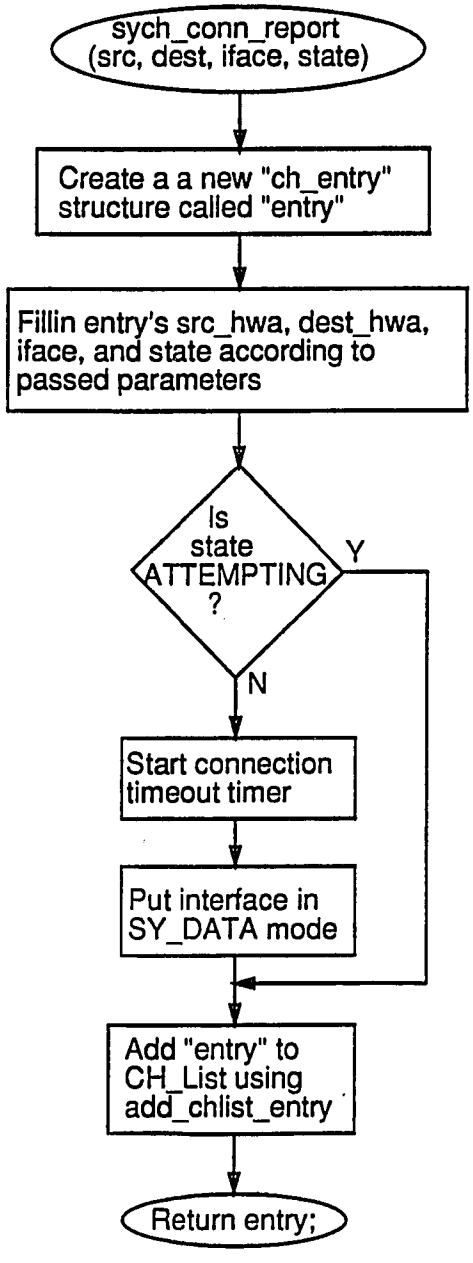


Figure 2.10 - sych_conn_report() Function

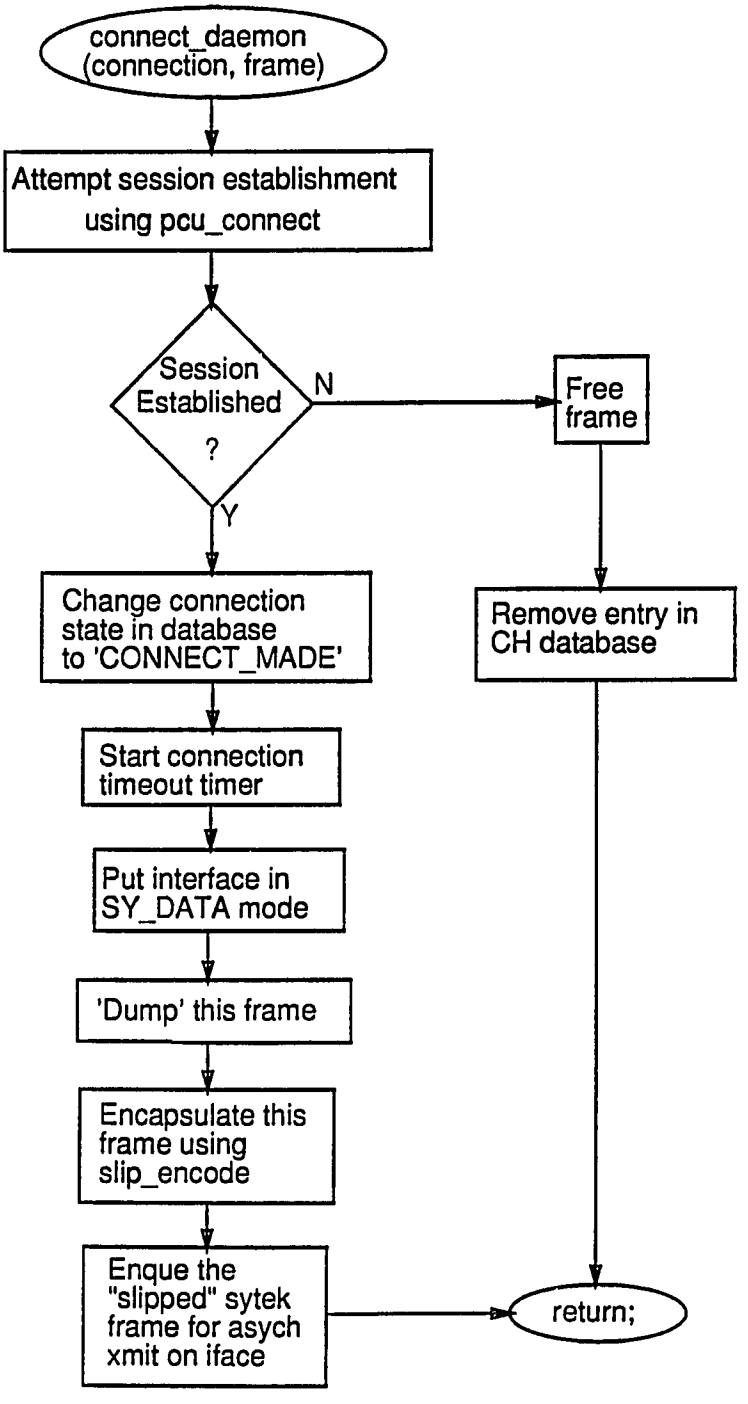


Figure 2.11 - connect_daemon() Function

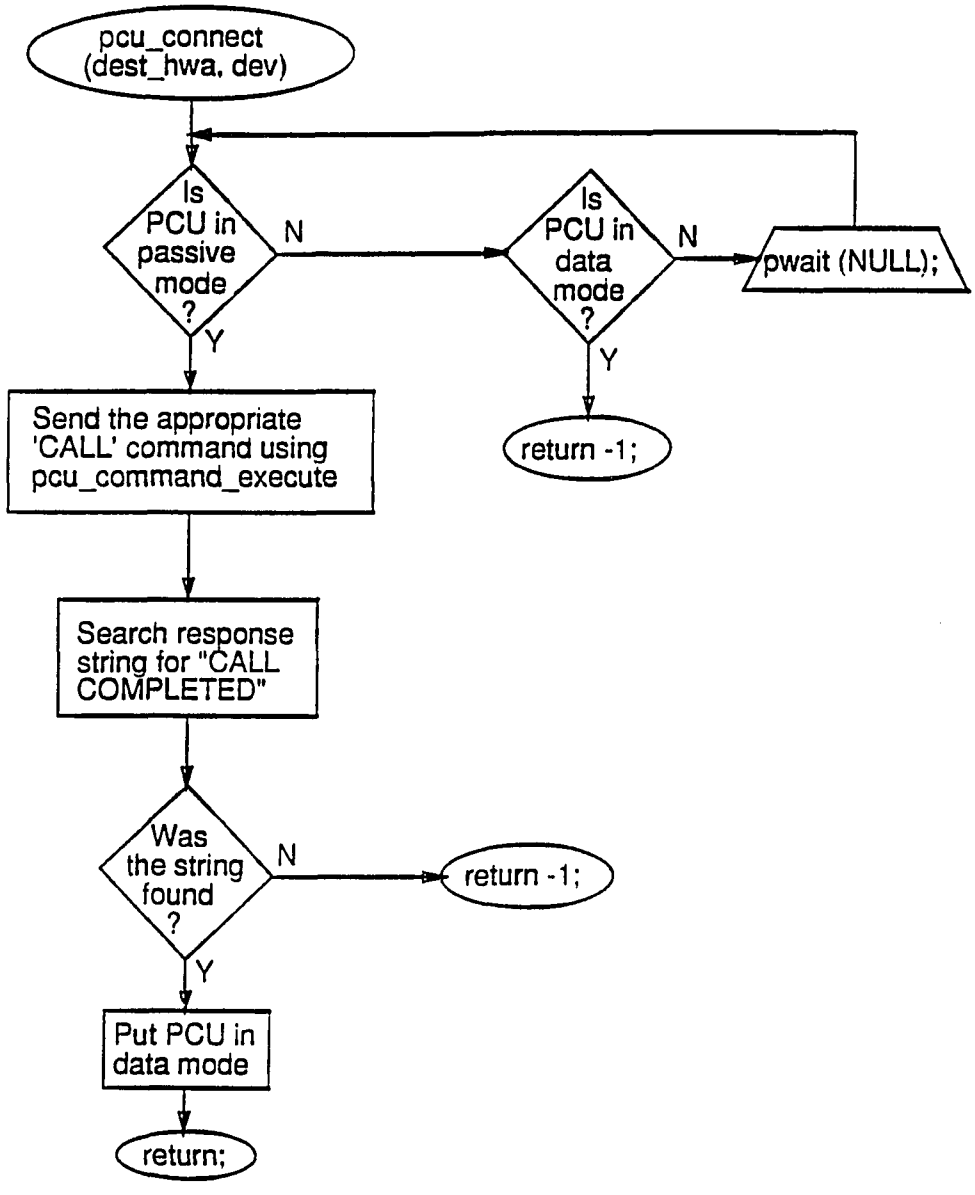


Figure 2.12 - pcu_connect() Function

rendezvous point is specified, as is the case here (i.e., "pwait (NULL);"), this process simply puts itself at the end of the list of active processes waiting to use the CPU.

Figure 2.13 describes the connection_timeout function. This function is called when a connection's idle timer expires. It checks to make sure that the specified connection is not already being disconnected because it is possible that after the timer has expired, but before this process gets a chance to run, the user may have manually initiated this connection's termination (e.g., exiting the program or detaching the interface associated with this connection). It sets the connection's state to DISCONNECTING, thus marking it as unusable for any processes that manage to sneak in before the forked sych_disc_request process can run. This separate process is forked out of necessity, since NOS does not allow a timer Interrupt Service Routine (ISR) to issue pwaits, which sych_disc_request does.

Figure 2.14 describes the sych_disc_request function. It can be run in it's own process, such as when scheduled by connection_timeout, or by direct call, such as when the interface associated with the connection is detached (possible during program exit - see "pcu_restore",

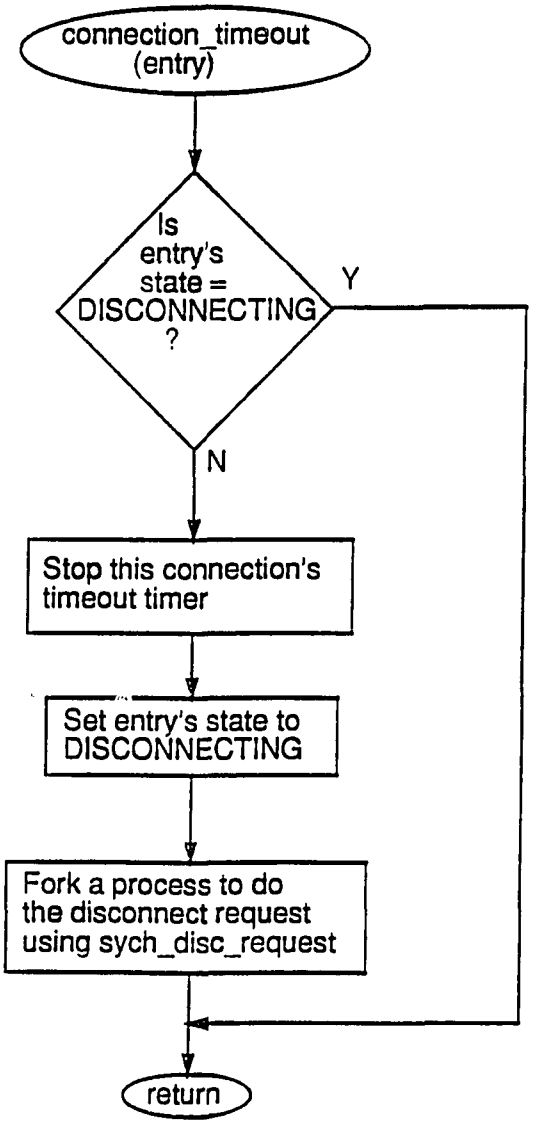


Figure 2.13 - connection_timeout() Function

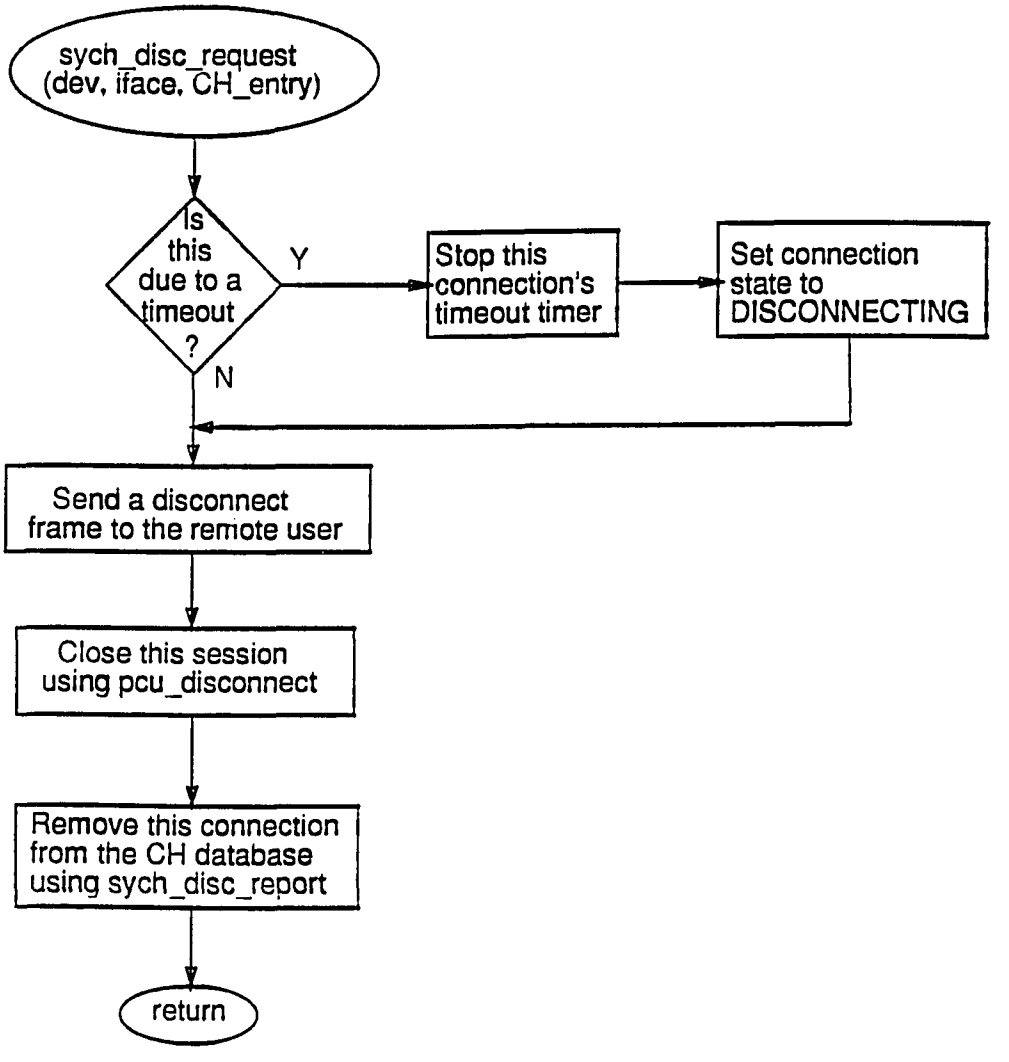


Figure 2.14 - sych_disc_request() Function

discussed in Section 2.2.2.5). It uses the "pcu_disconnect" and "sych_disc_report" functions, which are discussed below.

Figure 2.15 describes the pcu_disconnect function. It "escapes" the PCU out of data mode, into command mode. Then it issues the done command to terminate the session, using pcu_command_execute (c.f. 2.2.2.5).

Figure 2.16 describes the sych_disc_report function. It is called by sytek_recv when a disconnect frame is received, and also by sych_disc_request.

Figure 2.17 describes the pcu_pass_daemon function. This function is the source code for the "Passive Listener Daemon" process. It waits at a rendezvous point for a signal from the "asy_rx" process (c.f. 2.1.3.4) indicating that the PCU has sent an unrequested, carriage-return-terminated string from the PCU. Typically this will be notification of a remotely initiated session. This process updates the CH database accordingly, using sych_conn_report.

2.2.2.3 Unit Discovery Module

The three main functions of the UD module are syud_resolve, syud_request_handler, &

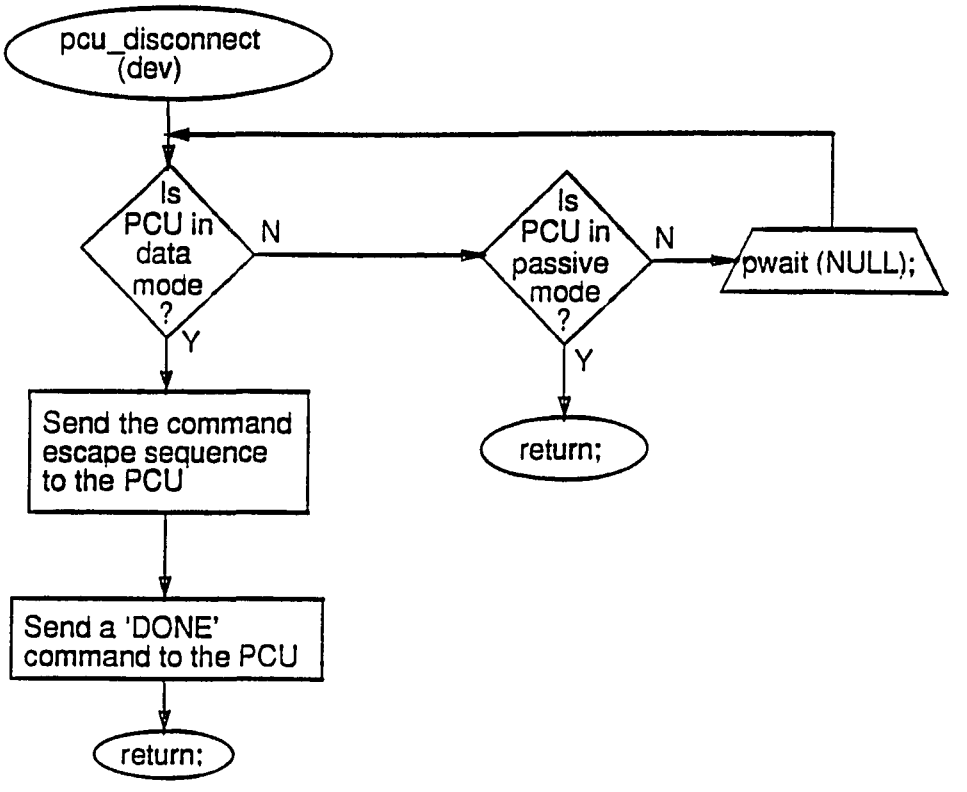


Figure 2.15 - pcu_disconnect() Function

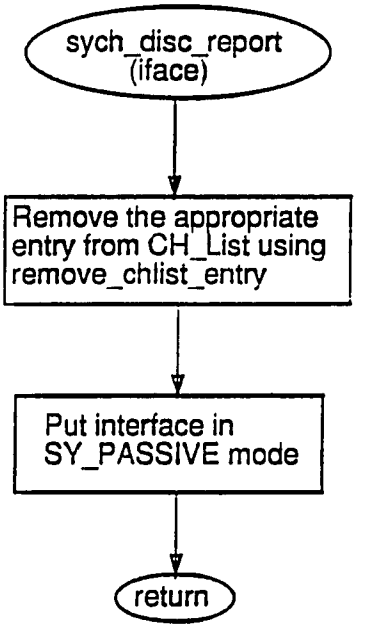


Figure 2.16 - sych_disc_report() Function

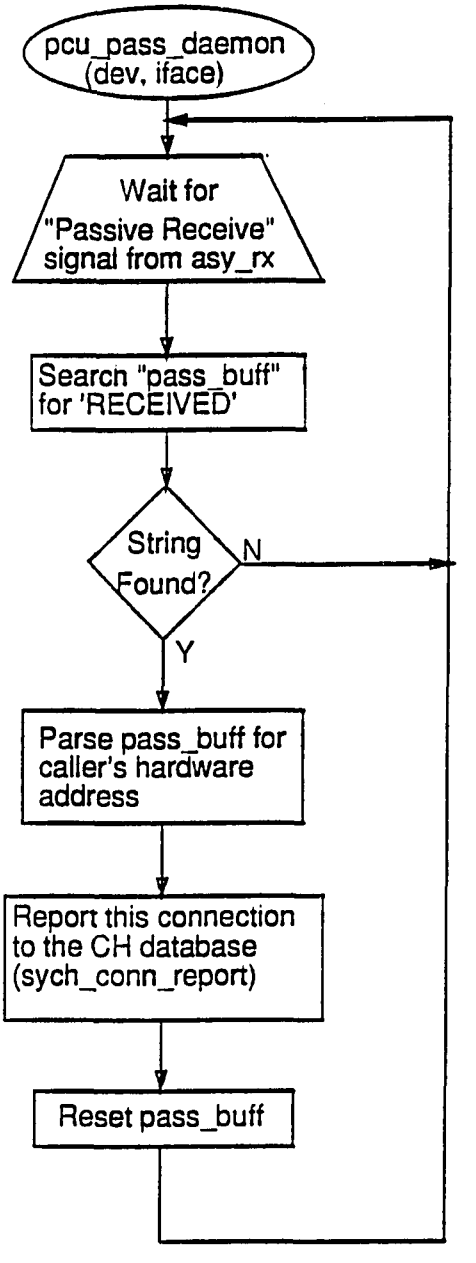


Figure 2.17 - pcu_pass_daemon() Function

syud_response_handler. The source code for these is in "sytekud.c".

Figure 2.18 describes the syud_resolve function. It is called by sytek_send to resolve an IP address into a Sytek hardware address. If the IP address isn't in our local database, a UD_REQUEST frame is sent to the first server in our UD_List.

Figure 2.19 describes the syud_request_handler function. It is called by sytek_recv to process a received UD_REQUEST frame. This routine mutates the frame header into a UD_RESPONSE. If the unresolved IP address is in the local UD database, it fills in the frame header accordingly. If it isn't, it fills in the frame header with NULL. This header is then bounced back to the requesting host over the connection it was received on. It is assumed that the remote system will terminate the connection, so we remove the CH entry with sych_disc_report, and stop this connection's idle timer.

Figure 2.20 describes the syud_response_handler function. It is called by sytek_recv to process a received UD_RESPONSE frame. If there is not template entry in the CH database, this is an "unsolicited" UD response, for which a new entry is simply added. If the frame carries a

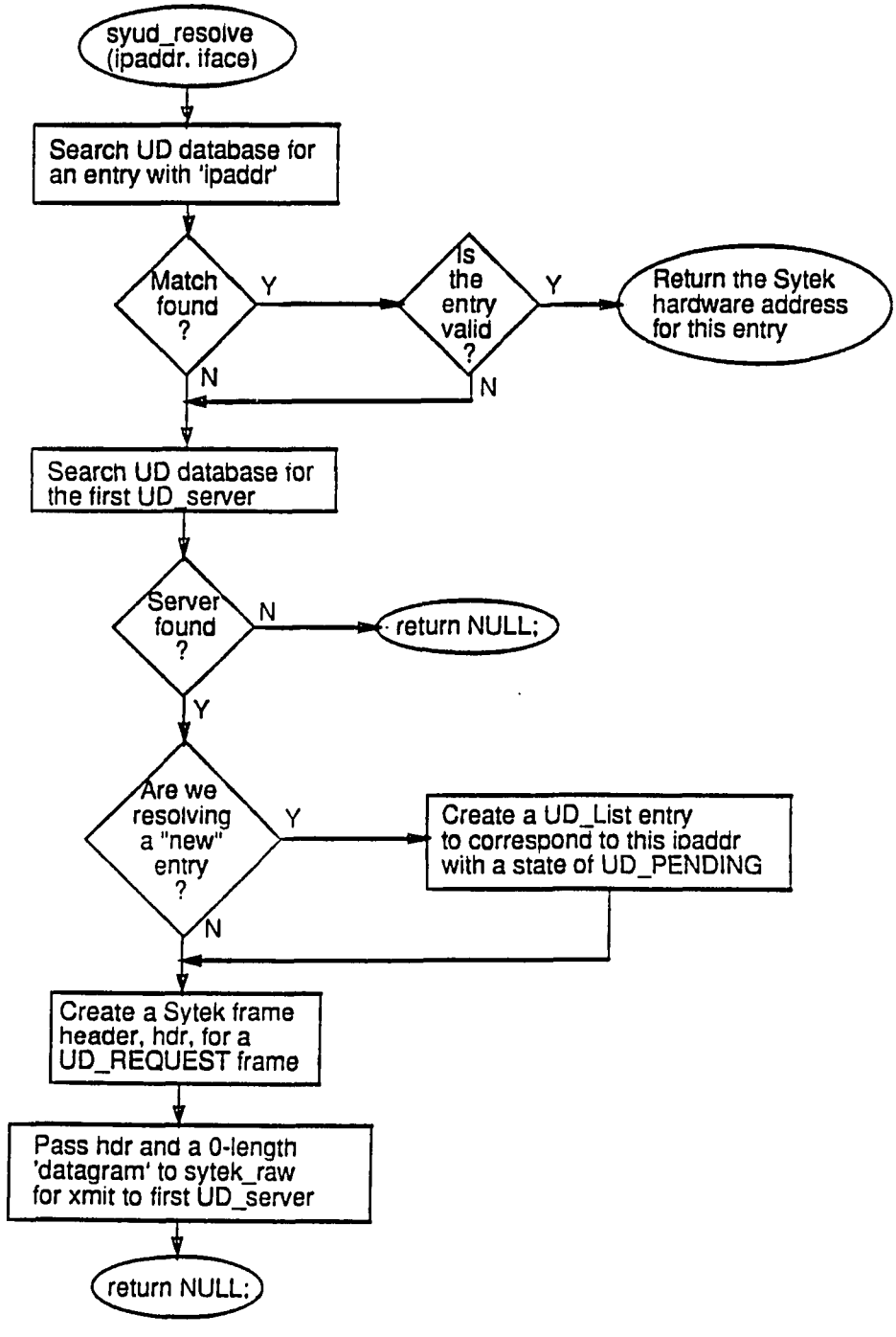


Figure 2.18 - syud_resolve() Function

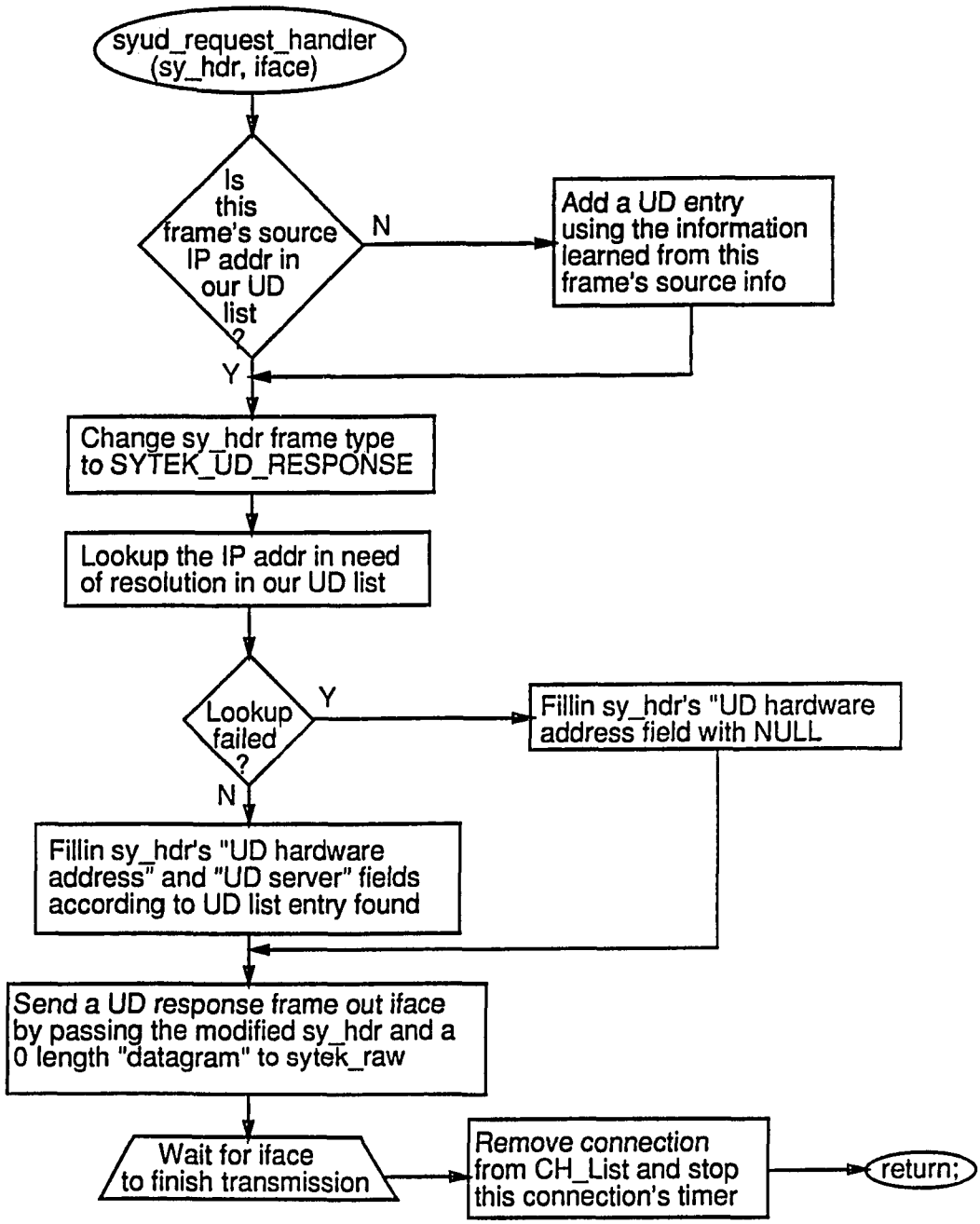


Figure 2.19 - syud_request_handler() Function

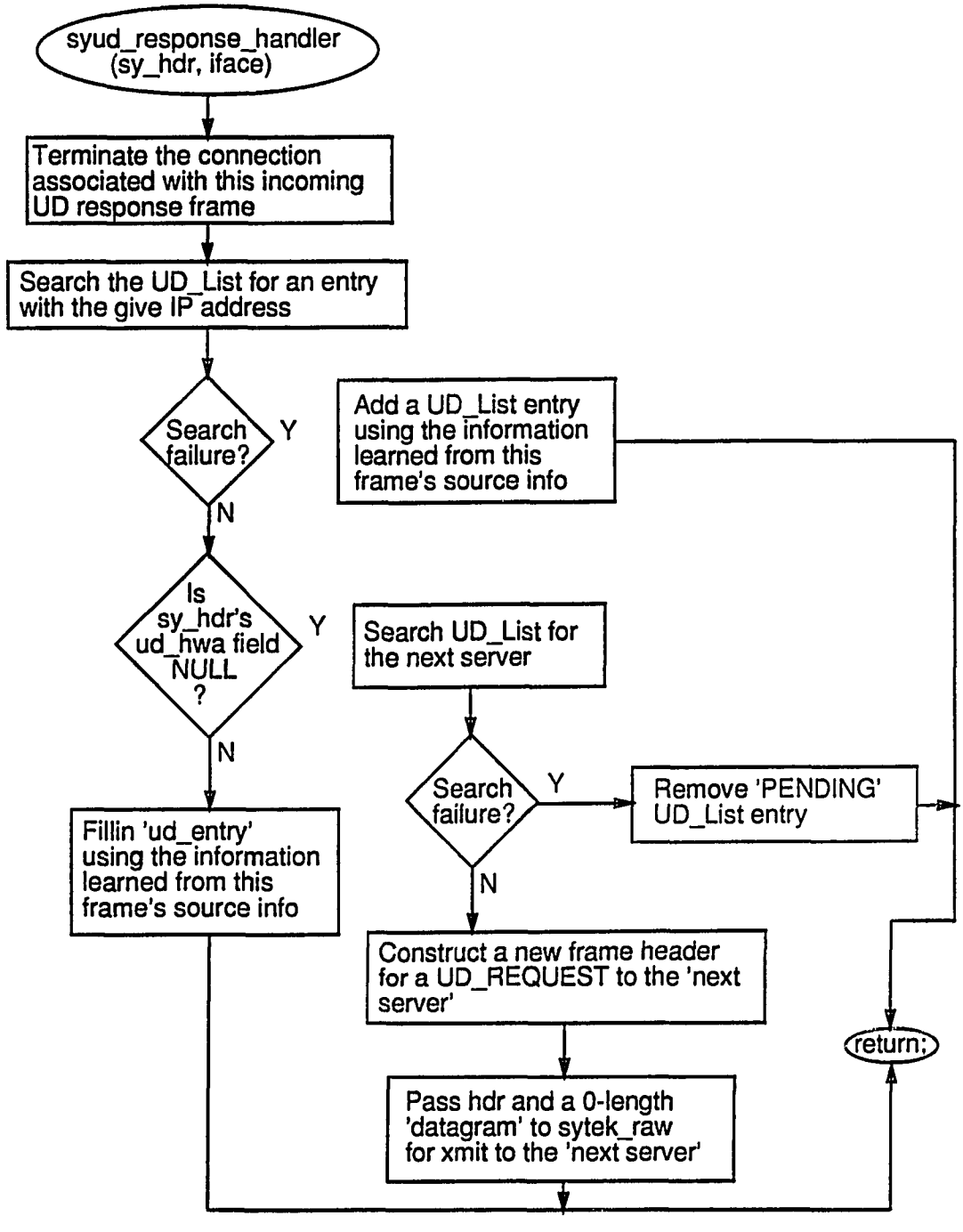


Figure 2.20 - syud_response_handler() Function

valid IP/Sytek address pair, the UD database is updated accordingly. If the header's "ud_hwa" field is NULL, the remote server couldn't resolve the IP address. In this case, syud_response_handler will and attempt to consult the next server listed in our UD_List, if one exists.

2.2.2.4 SLIP' Module

The five main functions of the SLIP' module are slip_send, slip_raw, slip_encode, slip_decode, and asy_rx. The source code for these is in "slip.c".

As mentioned previously, slip_send is practically just a transparent call to slip_raw. "slip_raw" does nothing more than encapsulate the frame SLIP'-wise using slip_encode and then enqueue the resulting data for asynchronous transmission. "slip_encode" and "slip_decode" simply perform the encapsulate discussed in Section 2.1.3.4. These routines are rather straight-forward, thus they were not flowcharted.

Figure 2.21 describes the asy_rx function. This routine is the source code for the "asyncronous receive" process. It's main function is to demultiplex incoming bytes according to the current state of the interface. The "Hopper queue" which complete frames are put into is the

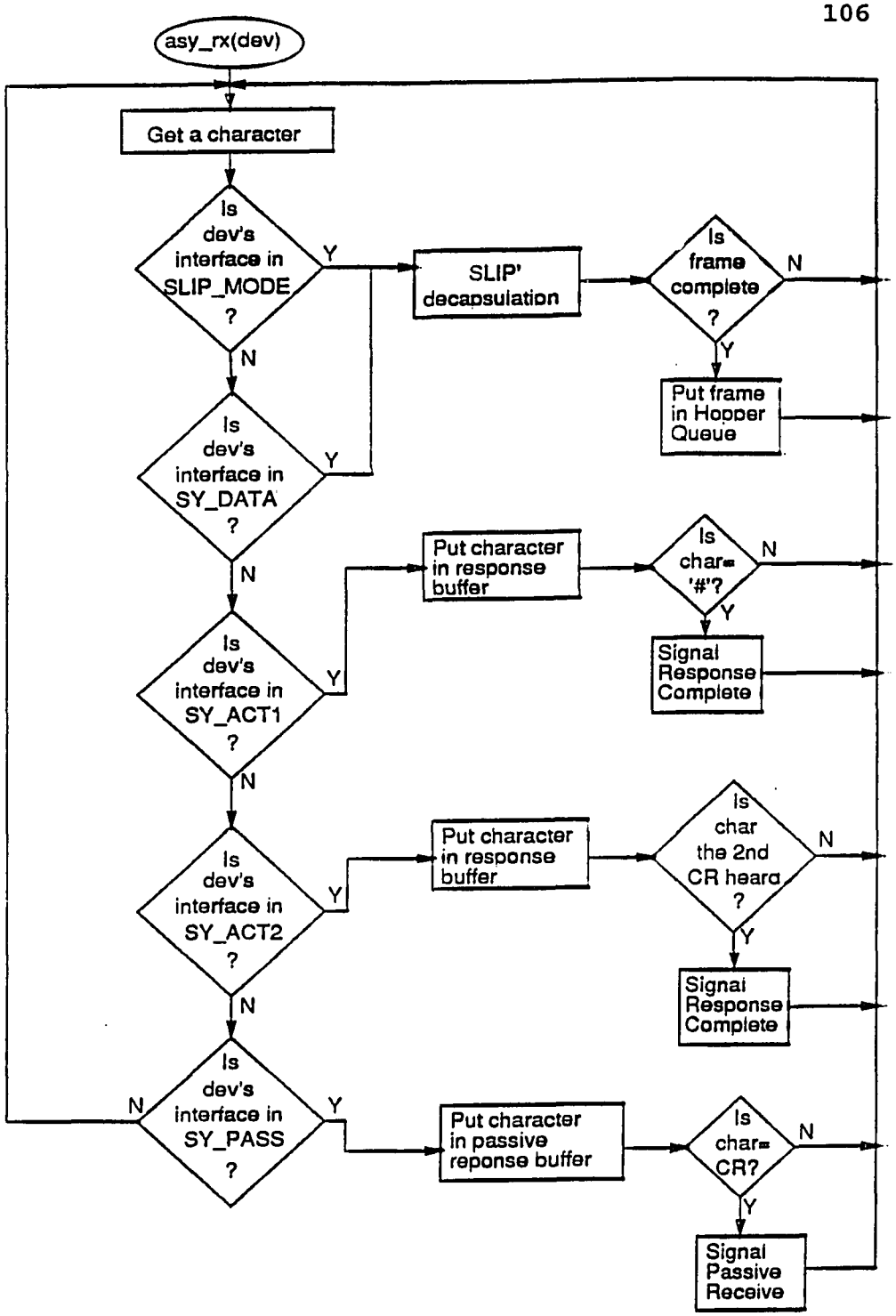


Figure 2.21 - asy_rx() Function

data structure KA9Q uses to pass all packets to their interface "receiving" routines. For instance, a frame which asy_rx enqueues in the Hopper that arrived on a Sytek interface will be dispatched to sytek_recv by the "network" process (whose main task is to distribute incoming frames). This architecture is part of the original KA9Q package and is discussed in more detail in Section B.2.

2.2.2.5 Non-Protocol Support Modules

This section is devoted to functions that were not discussed in Section 2.1.3, because they are not actually part of the Sytek driver protocol specification, but rather software-specific functions, although the line between these is very fuzzy.

First, we must mention that the UD module, in software, has two other important functions not yet discussed. They are used to load/save the UD database from/to a diskfile. When the first Sytek interface is attached, chud_init is called to initialize the UD database. This routine also initializes the CH_List pointer to NULL, thus the name chud_init. Then, every ten minutes (an arbitrary and easily modified value), syud_save is called to maintain the UD domain diskfile, \NET\SYDOMAIN.TXT. This periodic

saving is done to minimize loss of domain knowledge due to unexpected system crashes. When a Sytek interface is detached (usually during program exit), syud_save is again called to save the UD database. Both of these functions are in sytekud.c and are briefly described below:

chud_init ()

no parameters

This function initialized CH_List to point to NULL and also reads known Sytek Domain into UD_List from the file \NET\SYDOMAIN.TXT. It also starts the SDF ("Sytek Domain File") Periodic Update timer with an initial value of 600 seconds (ten minutes).

syud_save ()

no parameters

This function checks to see if the UD_List has been modified since it was last saved to disk. If it has, then the \NET\SYDOMAIN.TXT file is overwritten with a current version. Because this function uses DOS I/O calls to access the disk, it is horribly slow. To avoid blocking people, it gives up the CPU periodically, placing itself at the end of the active process list by using a "pwait(NULL)" NOS call (c.f. Appendix B.2).

The reason these functions were not discussed in Section 2.1.3 is that functions specific to software startup and shutdown are beyond the network layering paradigm. There are two other significant functions that fall into this category: pcu_init and pcu_restore.

Figures 2.22a and 2.22b describe the pcu_init function. It is called when a new Sytek interface is attached. It is responsible for initializing the PCU parameters to the expected state discussed in Section 2.1.1. Boxes in the flowchart which have only quoted strings inside indicate that the quoted string is sent to the PCU as a command. If this function changes any parameters, a command which would restore the parameter to it's original setting is added to the "restore_comms" linked list of the appropriate "pcu_state" structure (c.f. Section 2.2.1).

Figure 2.23 describes the pcu_restore function. It is called when a Sytek interface is detached, generally during program exit. It is responsible for restoring the PCU's original parameter settings. As the last user of this interfaces command and passive listener daemons, it also has the responsibility of killing there processes.

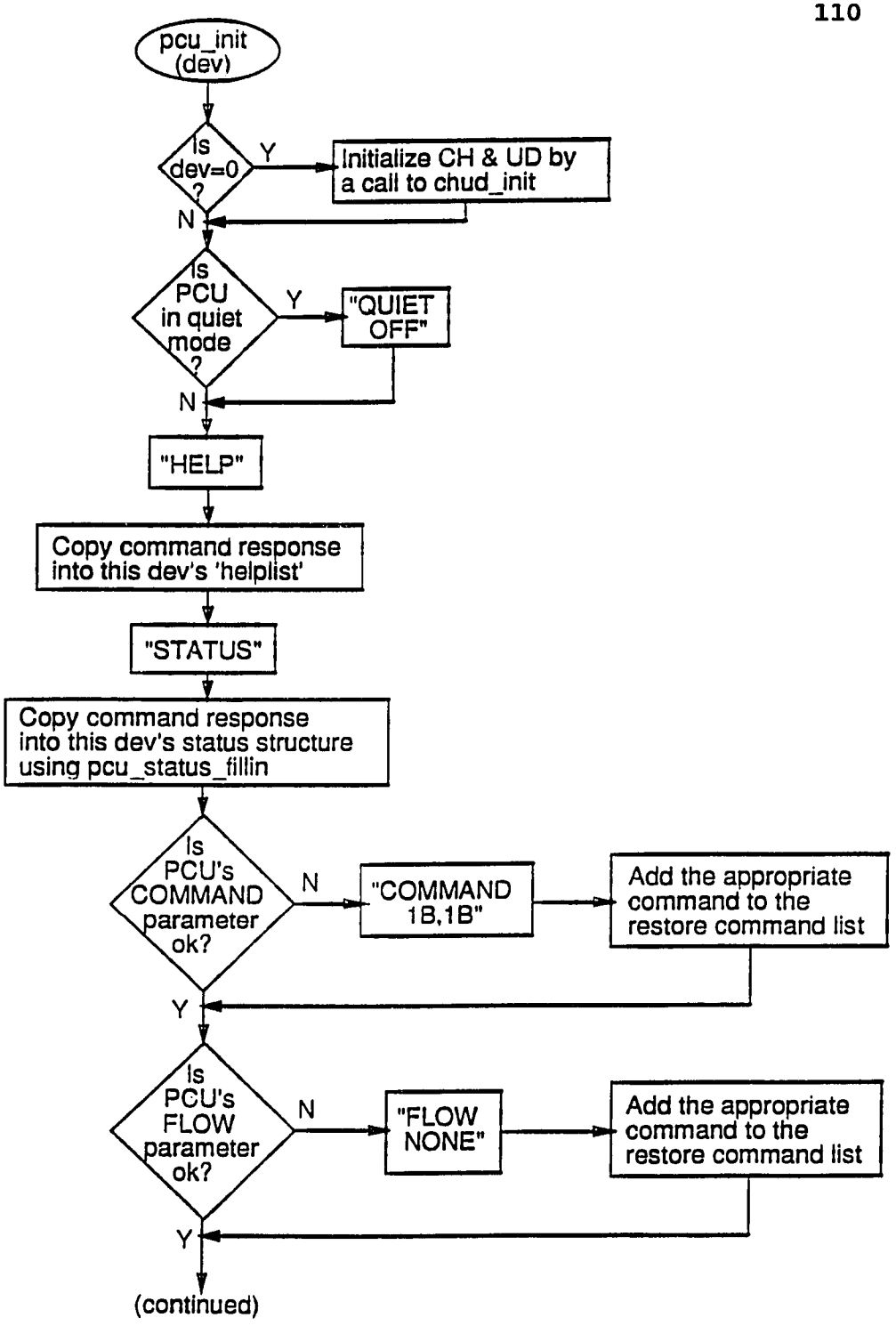


Figure 2.22a - pcu_init() Function

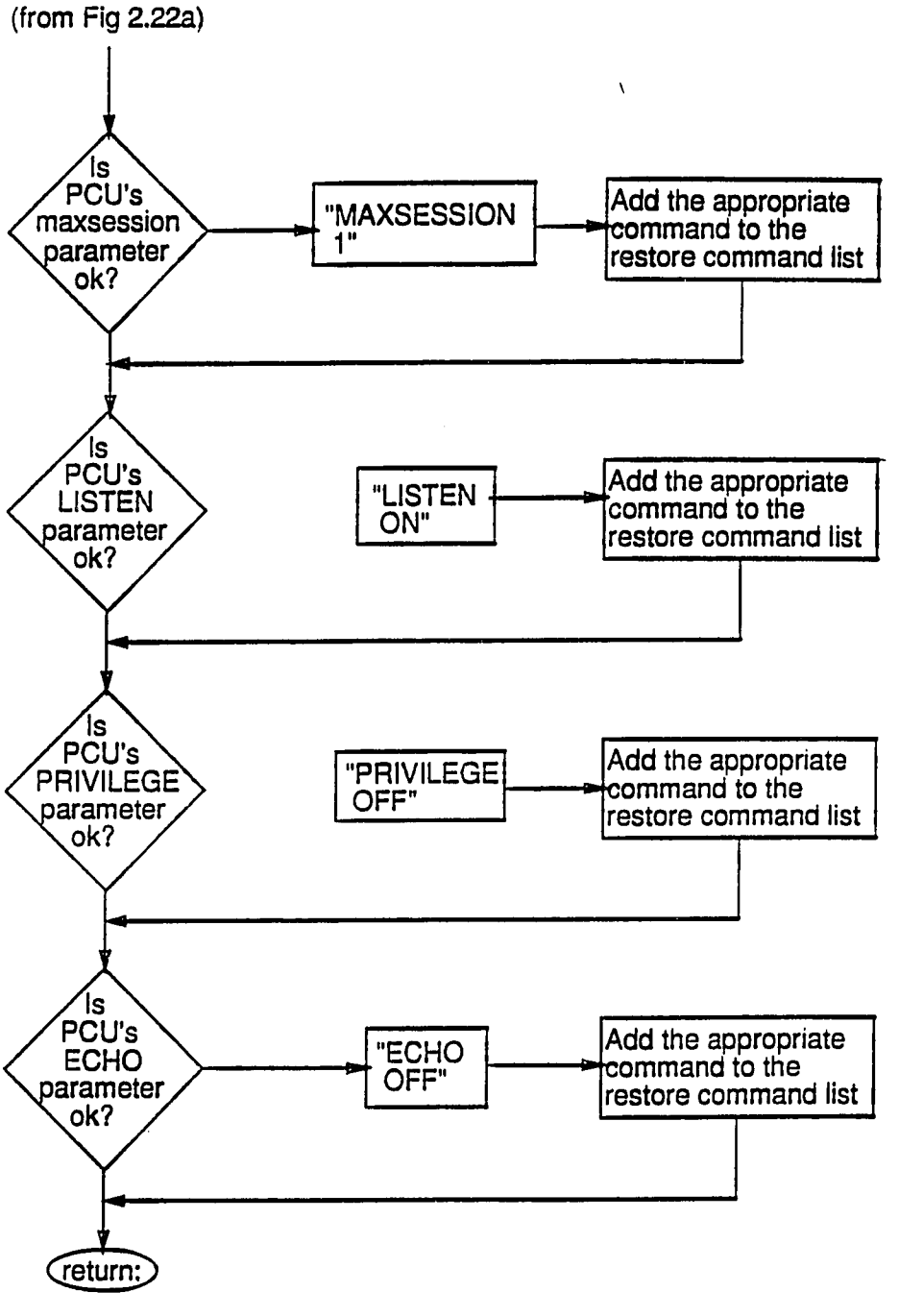


Figure 2.22b - pcu_init() Function (cont.)

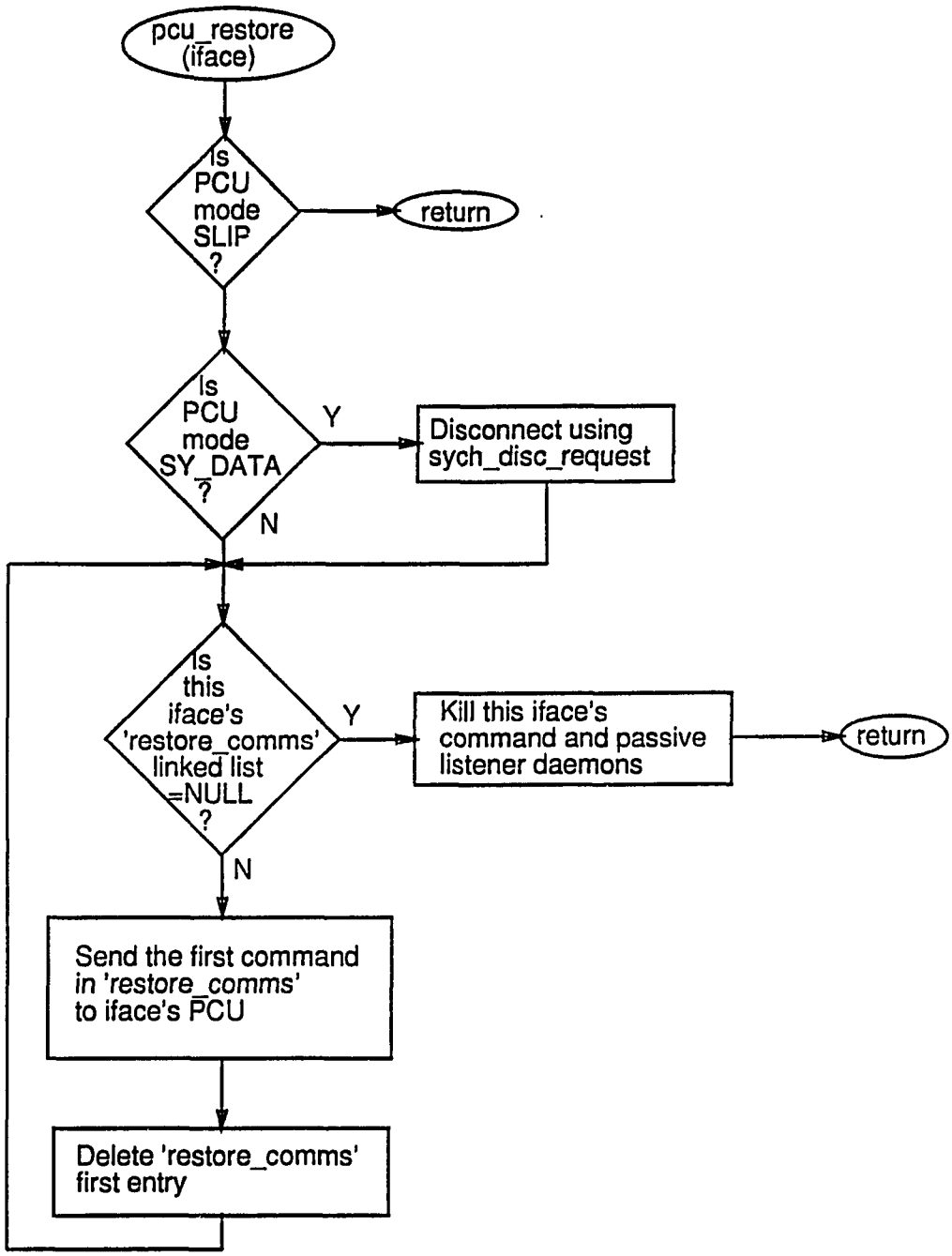


Figure 2.23 - pcu_restore() Function

As previously mentioned, some routines (c.f. pcu_init, pcu_restore, pcu_connect and pcu_disconnect) issue PCU commands and read the associated responses. It was decided that a separate process to interact with the PCU via the lower-level I/O routines would be appropriate, along with a general "command execute" function to provide a simple user interface to this process (and thereby, the PCU). The process was named the "PCU Command Daemon" and it's source code is the function pcu_comm_daemon. The user interface function is pcu_command_execute. These are in sycmd.c.

Figure 2.24 describes the pcu_comm_daemon function. It loops "forever" (until it's process is terminated) processing commands. It interacts with it's user (pcu_command_execute) via the interface's command and response shared memory buffers (c.f. 2.2.1) and three rendezvous points (PCU_COMM_EXECUTE, PCU_COMM_INDICATE, and PCU_COMM_DONE).

Figure 2.25 describes the pcu_command execute function. It allows it's caller to specify a command to be sent to the PCU, along with an "expected response length" and a time limit on how long a response should be waited for. The caller also passes a pointer to a buffer which the command response should be returned in. This function

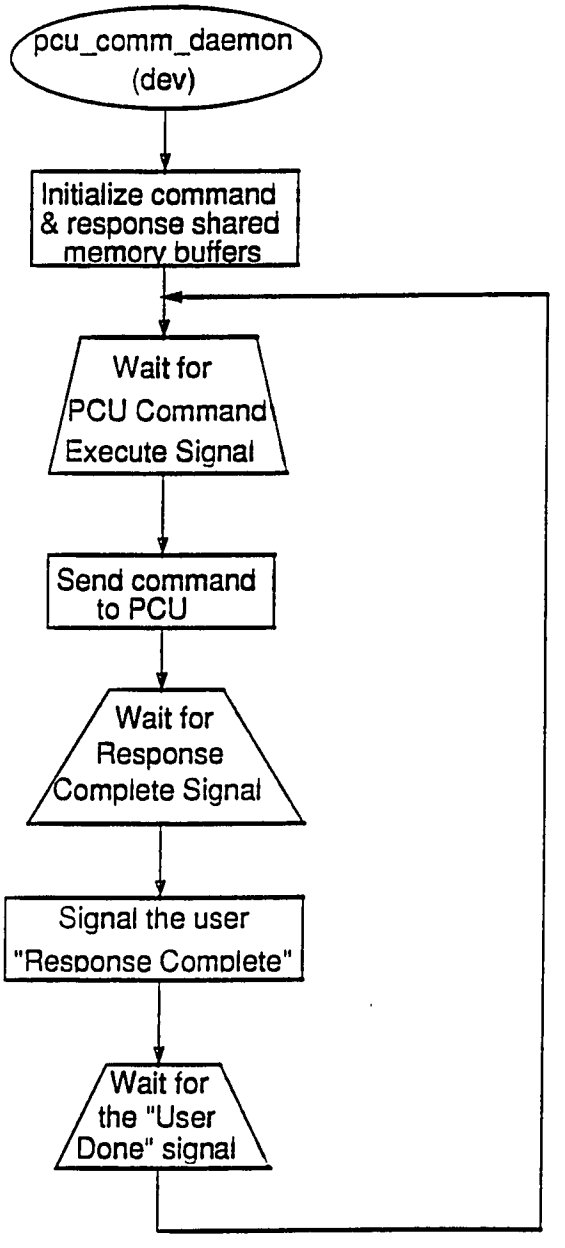


Figure 2.24 - pcu_comm_daemon() Function

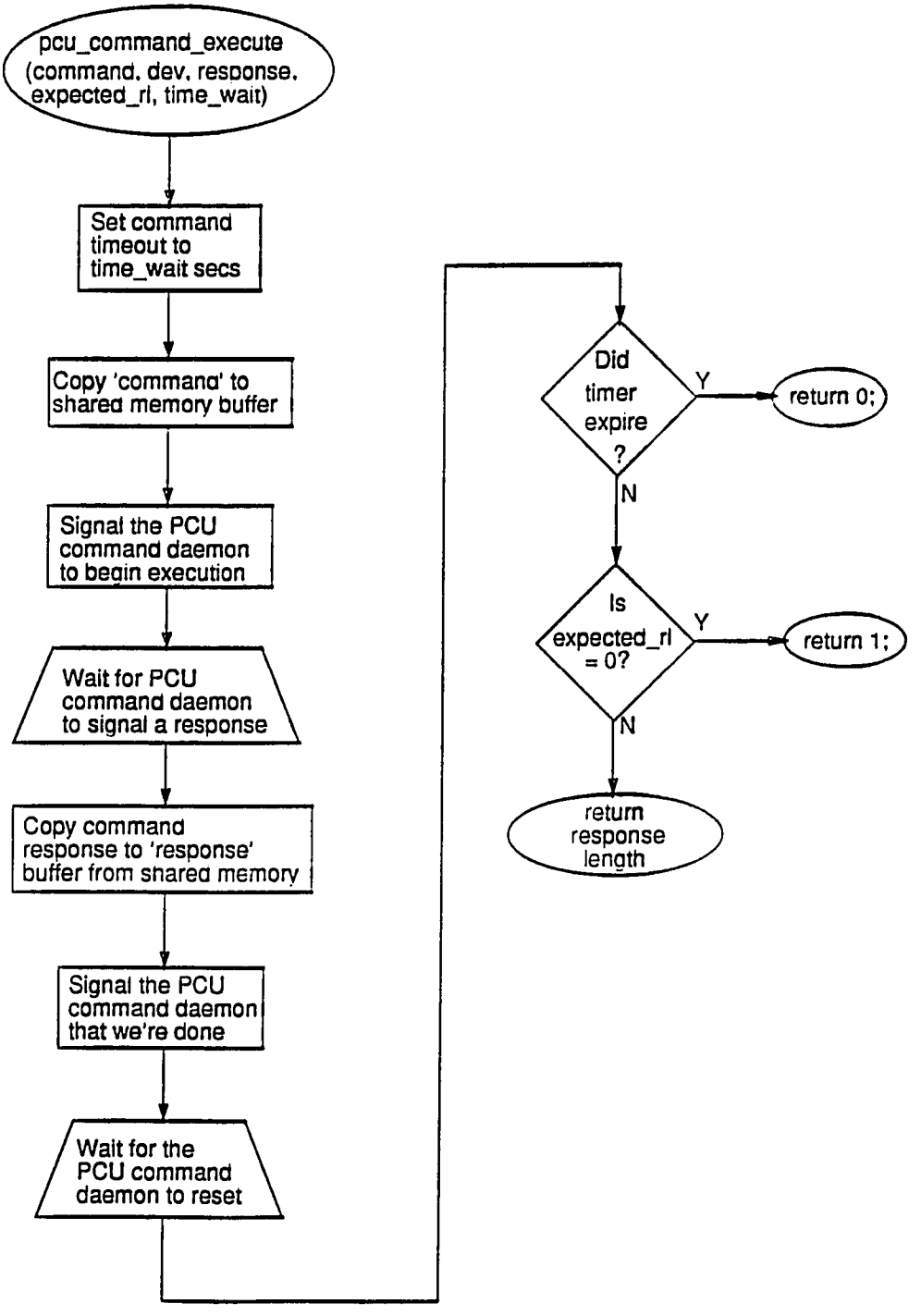


Figure 2.25 - pcu_command_execute() Function

returns the response string's length (or 0 if the "response wait" timer expires).

These two functions interact to provide a user with a generic command interface to the PCU. When pcu_command_execute is called, it copies the passed command into the proper shared memory buffer, and then signals the daemon on PCU_COMM_EXECUTE and waits for a rendezvous on PCU_COMM_INDICATE. The daemon issues the command to the PCU and waits for a "Response Complete" signal from the underlying network I/O routine, asy_rx. It the signals on PCU_COMM_INDICATE, waking the process which is running pcu_command_execute and waits for a rendezvous on PCU_COMM_DONE. The executing process wakes up, copies the response string into the caller-specified response buffer, and signals on PCU_COMM_DONE, which wakes the daemon. The executing process then puts itself at the end of the active process list, giving up the PCU to insure that the daemon gets a chance to reset before another command is issued. The three rendezvous points mentioned here are declared in sycmd.h.

In Section 2.2.1, a data structure for the internal representation of a Sytek header is given. It is a structure with fields of many different variable types.

For transfer over a Sytek session, this structure must be converted into a stream of bytes. This is done by the "htonsytek" routine. Of course, there is an analogous routine to convert from the network-specific format (a byte stream), back into an internal header representation; this routine is called "ntohsytek". These functions are fairly straightforward and are in "sytekhdr.c".

A KA9Q user has the option of tracing all the packets on an interface (c.f. Appendix A). This is a useful feature for debugging purposes. It required that a function be written and integrated into the KA9Q package which would 'dump' a Sytek header to the screen in a readable format. This is accomplished by the "sydump" function which is in sydump.c.

2.2.3 Documentation

Every module and function in my code has been carefully documented. This includes header comments explaining how each function is called, and parameters that must be passed to it, the value returned, and it's meaning. Comment lines are also included in-line in the program where necessary to clarify the actions taking place. The KA9Q source code is organized and commented very well.

Appendix A of this thesis provides a user's manual for the KA9Q package. It was intended to be useful to both beginning users and network experts. Appendix B provides some advice to future implementors and developers of KA9Q-based network software packages.

CHAPTER 3

GATEWAY TESTING AND USE SCENARIOS

This chapter discusses the testing performed to verify the functionality of the prototype gateway. In particular, testing was performed during each development phase at many levels of the design, from the unit code testing of each function, to the subsystem testing of each sub-protocol module, to the testing of the complete Sytek driver, and finally the testing of the complete gateway system. This is discussed in Section 3.1. This chapter also discusses the possible use scenarios for the prototype gateway in Section 3.2.

3.1 Gateway Tests

Recall that the gateway development was broken down into three phases in Section 1.2.4. These were "TCP/IP Over Ethernet", "TCP/IP Over Sytek", and "Sytek to Ethernet Gateway". The systems developed in each of these phases were extensively tested to verify their functionality. The tests used are discussed in this section.

3.1.1 Phase I (TCP/IP Over Ethernet) Tests

The first phase, "TCP/IP Over Ethernet", involved acquisition of an appropriate public domain software package. The package chosen, KA9Q, has been extensively tested by it's author and a large user community. For this reason no formal testing of the KA9Q software was performed. Verification of KA9Q's functionality was accomplished by running the software as both a server and a client for the three main applications (FTP, Telnet, and SMTP). The configuration in Figure 1.9 was also used to verify that KA9Q's Address Resolution Protocol (ARP) routines worked as expected over Ethernet, as well as it's Routing Information Protocol (RIP) routines.

3.1.2 Phase II (TCP/IP Over Sytek Network) Tests

The second development phase, "TCP/IP Over Sytek", was broken down into five main tasks (c.f. 1.2.4). Unit level testing was performed on each C function which was developed during these tasks. This was done using the Turbo Debugger to observe the function's execution and verify that function's operation according to the software specification in Chapter 2. Any pre-existing KA9Q routines which had to be modified for the integration of the Sytek

Network driver were also tested at the unit level.

The first task (TCP/IP over a null-modem cable) involved nothing more than implementing a Sytek driver which was a transparent interface to KA9Q's SLIP driver. The code development here involved integrating a new driver class into the KA9Q package. This fairly straightforward scenario was tested by simple "pings", FTP file transfers, and Telnet connections between the two PCs.

The second task (TCP/IP over a pre-established Sytek session) comprised extending the SLIP module to include character stuffing for the PCU command escape character (discussed in Section 2.1.3.4). Testing to verify this scenario was done by doing an FTP file transfer of a text file containing the following text string:

"<ESC><ESC>DONE<CR><CR>"

which would cause premature session termination if the stuffing failed.

The third task (adding PCU initialization / restoration) involved development of the PCU command daemon and it's user access function (pcu_command_execute) discussed in Section 2.2.2.5. Extensive testing was done to insure that this PCU command scheme worked properly. The routines were tested by dummy calls to

pcu_command_execute which attempted a variety of PCU commands and printed out the response strings for verification. Once this functionality was verified, the pcu_init and pcu_restore functions were developed and tested.

Testing pcu_init and restore involved using the Turbo Debugger to step through the routines and insure that the given commands had the anticipated effect. The program would be aborted after a PCU command was sent, to examine the PCU's status (using Kermit, a terminal emulation program) and verify that it was as expected. Code tracing was also done to insure that the proper restorative commands were saved by pcu_init and carried out by pcu_restore. A final test of these routines involved setting a PCU's parameters of interest to all be wrong, running the KA9Q package over this PCU, using this PCU to do the tests for the second task, and then exiting the program (which restores the PCU). The PCU was returned to it's original state after being used, as we would expect. We then ran KA9Q using this PCU again, aborting the program after initialization. The PCU's state was observed to be as we would expect after initialization, but before restoration.

The fourth task involved developing the software modules for the Connection Handler (described in Section 2.2.2.2) as well as adding a rudimentary Unit Discovery by table lookup module.

First the passive listener daemon code was developed and tested along with the connection timeout timer. This was done by having a terminal connected to one PCU (actually a PC running Kermit) and the test system running KA9Q connected to another PCU on the Sytek Network. Calls were issued from the terminal to the test system's PCU. The code printed out a report that a connection had been heard and entered into the CH database. After the connection timer expired, the test system would terminate the session, as observing the dummy terminal would show (it would receive a "SESSION CLOSED" message) and remove the CH database entry.

Then the remainder of the CH module was developed and tested. Explicit connection and disconnection requests were shown to work as expected, with one PC initiating and another "hearing" the connection. Then the UD by table lookup was added and it was shown that an IP datagram addressed to a remote Sytek Network host which was in our SYDOMAIN.TXT file would cause the initiation of the

appropriate Sytek session, while one that was not in our database would simply be dropped.

The fifth task involved developing and testing the more complicated Unit Discovery scheme discussed in Section 2.2.2.3. This UD was the most complex module of the Sytek driver to test, because it deals with a distributed system of hosts and UD servers. The participating machines performed "packet tracing", dumping the contents of packets traversing their network interfaces. This was used to verify that hosts were sending the expected network frames. The internal state of each system was verified having the software report significant events using printf statements.

The UD module test plan is shown in Figure 3.1. It supposes three systems, Fred, Barney and Wilma, which are hosts on the Sytek campus subnetwork (128.196.68.xxx). Each has it's own Sytek address ("E221,0", "E905,0" and "E432,1" respectively). Barney has been designated as this network's UD server. At the outset, the UD databases in Fred and Wilma reflect knowledge of Barney's role as UD server. Barney has no knowledge of either Fred or Wilma. The four successive tests shown were carried out and the activity shown was verified to have occurred.

Hostname **IP address** **Sytek address**	Fred 128.196.68.1 E221,0	Barney 128.196.68.2 E905,0	Wilma 128.196.68.3 E432,1
Initial Sytek UD database	*128.196.68.2 E905,0 S*	(empty)	*128.196.68.2 E905,0 S*
	Knows Barney	-----	Knows Barney
Test A	ping Barney sytek data frame → (ping fails)	Can't resolve Fred's IP address to respond	
Test B	ping Wilma ud_req(x.x.x.3) → (ping fails) ud_req failed ←	learns x.x.x.1's hwa *128.196.68.1 E221,0 S?* ud_resp(NULL) ←	
	Knows Barney ping Barney - ok	Knows Fred ping Fred - ok	Knows Barney
Test C		learns x.x.x.3's hwa *128.196.68.3 E432,1 S?* ud_resp(E221,0...) →	ping Fred ud_req(x.x.x.1) ← (ping fails) learns x.x.x.1's hwa *128.196.68.1 E221,0 S?*
	Knows Barney ping Barney - ok	Knows Fred, Wilma ping Fred, Wilma - ok	Knows Fred, Barney ping Barney - ok
Test D	ping Wilma ud_req(x.x.x.3) → (ping fails) learns x.x.x.3's hwa *128.196.68.3 E432,1 S?*	ud_resp(E432,1...) ←	
	Knows Barney, Wilma ping Barney,Wilma - ok	Knows Fred, Wilma ping Fred, Wilma - ok	Knows Fred, Barney ping Fred, Barney - ok

Figure 3.1 - UD Test Plan

Incidentally, "ping" is a simple program which sends a "ping request" in IP datagram to remote system. The remote system should immediately send a "ping response" datagram back. This is used to establish reachability in the Internet. We use it here to do the same.

In Test A, Fred attempts to ping Barney. Since Fred knows Barney (i.e., Barney is in Fred's UD database), he doesn't have to initiate any remote UD resolution in order to get a frame delivered to Barney. Barney receives the ping request and will attempt to send a data frame containing the ping response to Fred. Unfortunately, Barney's Sytek driver will fail to resolve Fred's IP address immediately (i.e., he doesn't know Fred). Since there are no UD servers in his database, Barney drops the response datagram and Fred's ping fails. In order for a ping to succeed, both participants must know each other.

In Test B, Fred attempts to ping Wilma. Wilma is not in Fred's UD database, so he must consult Barney. Fred drops the ping datagram and sends a UD request frame to Barney, asking for resolution of "128.196.68.3", Wilma's IP address. Barney receives the UD request and learns Fred's IP/Sytek address pair from it. Unfortunately, Barney cannot resolve Wilma's IP address, so he sends Fred a UD

response frame indicating failure. Fred has no other servers in his UD database, so he cannot resolve Wilma. But, Barney has learned about Fred. This means Fred and Barney could then ping each other.

In Test C, it is first verified that Fred and Barney can now ping each other. Then Wilma attempts to ping Fred. Fred is not in Wilma's UD database, so she must consult Barney. Wilma drops the ping datagram and sends a UD request frame to Barney, asking for resolution of "128.196.68.1", Fred's IP address. Barney receives the UD request and learns Wilma's IP/Sytek address pair from it. Since Barney knows Fred's address pair, he can send Wilma a UD response frame with this information. Thus Wilma learns Fred's address pair and Barney learns Wilma's address pair. Now Barney and Wilma should be able to ping each other. But Wilma and Fred still can't ping each other because Fred doesn't know Wilma yet.

In Test D, it is first verified that Barney and Wilma can now ping each other. Then Fred attempts to ping Wilma. Wilma is still not in Fred's UD database, so he must again consult Barney via UD request frame. Now, however, Barney knows Wilma's address pair, so the UD response frame he sends to Fred contains a useful information. Thus Fred

finally learns Wilma's address pair. Fred and Wilma should now be able to ping each other. In fact each system should be able to ping all the others (this was verified to be true).

Once the five tasks had been completed, and all the driver's subsystems (i.e., CH, UD) were tested, testing of the Sytek driver as a unit was performed. This merely involved configuring the second development system (Figure 1.10) and running KA9Q over the Sytek drivers as both a server and a client for the three main applications (FTP, Telnet, and SMTP). During these tests, the UD modules in the workstations were observed to do the expected remote consultations to resolve IP addresses. The Connection Handler was also implicitly tested, including observing dynamic establishment and termination of Sytek sessions to transfer IP datagrams.

3.1.3 Phase III (Sytek to Ethernet Gateway) Tests

The last phase, "Sytek to Ethernet Gateway", involved insuring that the Sytek driver would work as part of a KA9Q-based gateway. Tests were performed to verify this functionality, but a successful completion of the second phase was a clear indicator that these tests should

succeed.

Two main tests were done at the gateway system level. First, the gateway's functionality was verified by configuring the third development system (Figure 1.11) and establishing TCP sessions between the Sytek workstation and the Ethernet workstation, through the prototype gateway. This was done with both workstations as application program clients and servers.

Then the gateway was tested to insure that it could sustain continuous usage by establishing an FTP session through it and transferring many large files between the two workstations.

3.2 Gateway Usage

This section provides a brief synopsis of the usage scenarios resulting from incorporation of the UA Sytek Network into the Campus Internet. For details on how to use the KA9Q package on a Sytek workstation, consult the user manual in Appendix A.

Figure 3.2 shows how implementation of Generic Gateways could affect the UA Campus Internet's connectivity. This example shows the Sytek Network connected to the Ethernet backbone via the prototype gateway developed in this thesis

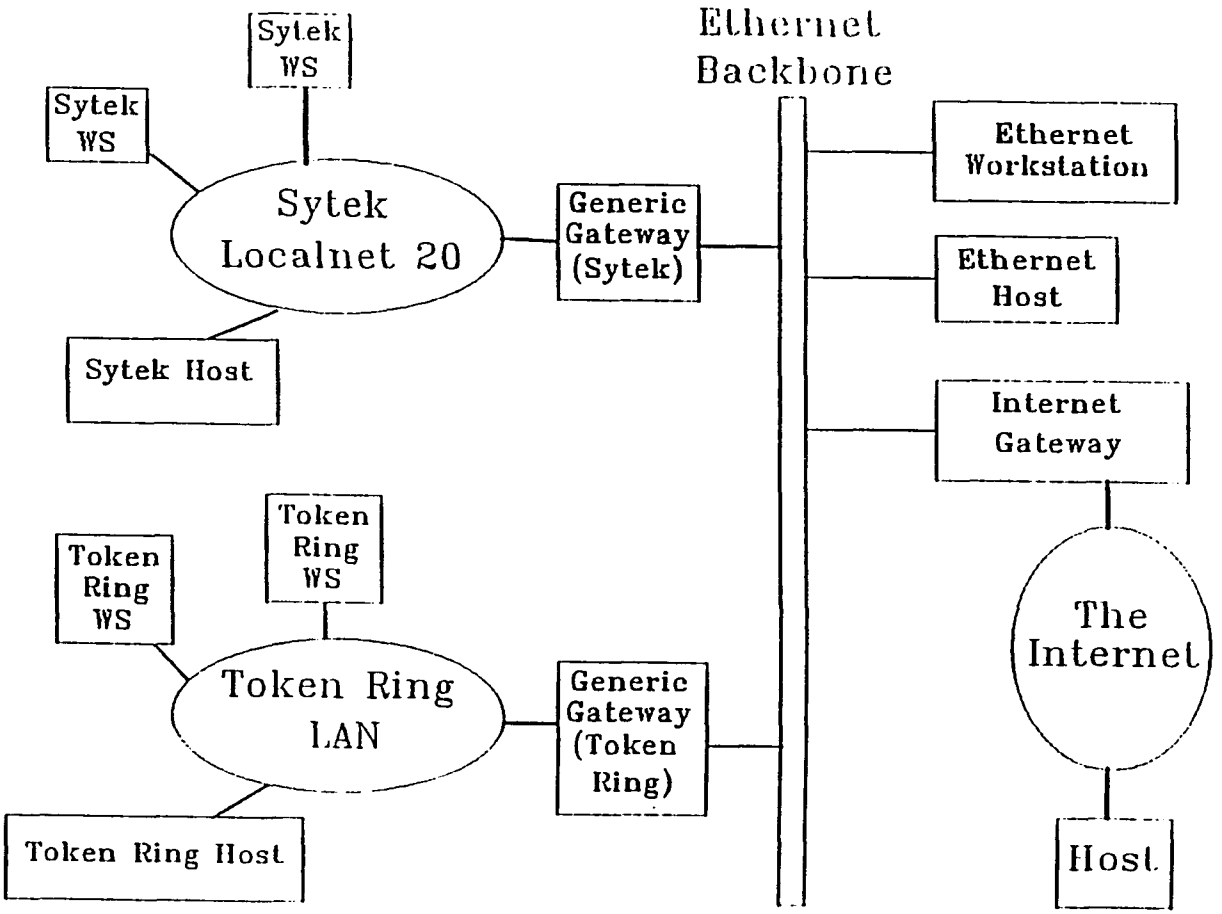


Figure 3.2 - Generic Gateway Usage

research. It also shows a generic gateway connecting a Token Ring LAN to the Ethernet backbone. Such an implementation does not yet exist, but may quite easily be accomplished using KA9Q, which already has support for Token Ring networks (c.f. 4.2.4).

Also connected to the backbone are an Ethernet workstation, an Ethernet host, and the Ethernet/Internet gateway. This picture does not fully reflect the campus backbone's physical connectivity (c.f., Figure 1.3), but provides an accurate model of it's logical structure.

From this diagram, we can see that the prototype gateway allows Sytek hosts access to all other hosts on the UA Campus Internet. This access is via the TCP/IP application layer protocols. That is, any participating Sytek user could act as client or server in FTP, Telnet or SMTP sessions with any other host on the UA Campus Internet. Since the UA Campus Internet is also part of The Internet, Sytek users have similar access to Internet hosts.

If another generic gateway were implemented, the Token Ring gateway shown here for example, it would afford the Token Ring users similar access to the UA Campus Sytek Network, the Campus Internet, and The Internet. Actually

the Sytek Network is part of the UA Campus Internet, which in turn is part of The Internet, so one could simply say the connected subnet users get complete Internet access.

It should also be noted that this research provided TCP/IP interconnectivity among the Sytek Network hosts themselves. That is, Sytek hosts can access each other using FTP, Telnet and SMTP. Previous to this research, Sytek users only had virtual serial connectivity to each other.

CHAPTER 4

SUMMARY & CONCLUSIONS

The focus of the "Generic Gateway" research project is to bring about greater inter-connectivity among the campus networks. Since UA already has a substantial TCP/IP internet with an Ethernet backbone, it makes sense to approach full campus interconnectivity by incorporating detached campus networks into this UA Campus Internet. It was determined that the best approach for assimilating detached networks is through the use of IP-level routers.

In this thesis, the UA Sytek Network was selected as a candidate network to be added to the UA Campus Internet. This research involved the design, development and testing of a gateway (IP-router) to interconnect Sytek and Ethernet. The current gateway architecture was intended as a prototype only; it demonstrates the viability of a TCP/IP Sytek Network which is a Class C subnetwork of the UA Campus Internet (and thereby part of The Internet).

Section 4.1 discusses some of the constraints of the prototype gateway, and Section 4.2 proposes some possible solutions to these constraints, as well as discussing directions for future research. In light of the

constraints discussed in Section 4.1, it is not recommended that the prototype gateway be used to support an "active" Sytek subnetwork with multiple gateway users.

4.1 Current Constraints

4.1.1 Problems with KA9Q

The KA9Q Internet Software Package has proven to be an invaluable resource for this research. Nevertheless, it does have it's limitations. KA9Q was designed to be a robust platform, with an emphasis on portability and the support of myriad network interfaces. This made it an ideal choice for the generic gateway research. The down side is that the user interface can at best be described as 'terse'.

This is not to say that the KA9Q applications are unusable. They are just not as user-friendly as some other public domain PC TCP/IP packages. For instance, the KA9Q's Telnet client provides virtual terminal service, as required by the Telnet protocol specification. This service, however, is that of a dummy terminal; NCSA Telnet (another popular public domain PC TCP/IP software package) has VT100 and TK4014 terminal emulation over it's Telnet.

KA9Q was selected for it's "developer-friendliness", unfortunately at the expense of end-user friendliness. This tradeoff was necessary, since many of the user-friendly packages were not suited for use as a gateway. KA9Q is quite usable, indeed it has hundreds (perhaps thousands) of users worldwide. It's user interface is just not entirely state-of-the-art. The good news is that KA9Q is undergoing constant development and the user interface is improving. Besides, Sytek users interested in Internet connectivity should be glad to have any connectivity, bells and whistles aside.

Another limitation of KA9Q that was realized in the course of this research is that it does not implement the FTP commands mput and mget. These are commonly employed by FTP users to transfer whole groups of files using only one command (i.e., mget *.* will get every file in the current remote directory). KA9Q users wishing to transfer a large groups of files must mget/mput each separately. From a user's perspective, particularly one who has used other FTPs, this is an unfortunate attribute of KA9Q. This limitation has been indicated to Phil Karn (KA9Q's author) and is included in his list of features of future KA9Q releases.

Another problem with KA9Q is it's conspicuous lack of good documentation. The user documentation is somewhat dated, but basically accurate; those who refer to "The KA9Q Internet Software Package", [10] and Appendix A of this thesis should have little trouble using KA9Q. The real shortcoming lies with the developer's documentation: there is none. All that can be said to mollify future developer's fears is that the code has proven to be self-documenting, to some degree.

Appendix B gives a partial overview of the KA9Q architecture, from a developer's perspective. It is not intended to be an exhaustive developer's manual (indeed, such a document would be quite an undertaking, particularly for a KA9Q neophyte), but it will give future developers a running start, at least until the long-awaited and often-promised "new" KA9Q documentation materializes.

4.1.2 Centralized Unit Discovery

The Unit Discovery (UD) protocol was designed to resolve IP addresses into their Sytek equivalents, much in the way the Address Resolution Protocol (ARP) is used on Ethernet LANs. One big difference between these two networks is that Ethernet is inherently connectionless,

while Sytek, from our network driver's perspective, is connection-oriented.

ARP was designed to take advantage of Ethernet's broadcast capability, implementing a distributed address resolution algorithm (c.f., RFC 826, "An Ethernet Address Resolution Protocol', [24]) which is highly robust. Due to Sytek's lack of any true broadcast capability, the UD design is considerably more centralized, and necessarily less robust.

More precisely, the UD server(s) are single points of failure. If all the UD servers on one Sytek Network are down, local hosts are limited to communications using previously known Sytek addresses. In fact, this limitation isn't imposed just when all the servers are actually down, but merely when they are all without available Sytek connections (i.e., when they're each "talking" to someone else).

For a scenario where there are very few Sytek Network TCP/IP users, these limitations are not critical. As number of these users increases, however, these restrictions become unacceptable. Recommended improvements to the Unit Discovery protocol are given in Section 4.2.1.3.

4.1.3 Single Session Limitation

Perhaps the most serious limitation which prevents the prototype gateway from being a final solution to the incorporation of the UA Sytek Network into the Campus Internet is it's inability to handle multiple simultaneous sessions. The prototype gateway could be used as a partial solution to this problem, until a more robust gatewaying scheme can be devised.

It should be noted that under the current architecture, there is no association between TCP virtual sessions and Sytek sessions. Sytek sessions are dynamically created to deliver IP datagrams, but if they remain idle for some predetermined period of time, they are automatically terminated.

At first, it was thought that careful tuning of this timeout parameter might yield a gateway with a crude multi-session capability. Assume a Sytek user establishes a TCP session through the gateway, and then remains idle long enough for his gateway connection to timeout. Now if another Sytek user wishes, he/she may establish a second TCP session through the gateway into The Internet. Now there are two TCP sessions virtually "going through" the

gateway. If these two users happen to interleaf their I/O (i.e., if each only sends IP datagrams when the other has been idle), each will see no degradation in the gateway's performance.

If, however, both users attempt to transfer IP packets concurrently, the first to establish a Sytek session with the gateway will succeed. The second user's Sytek driver will fail to establish a connection with the gateway and drop the IP datagram. Of course, some time later, the second user's TCP will timeout and instruct IP to attempt re-transmission. If the first user's session with the gateway has timed out, the second may now transfer his IP datagram.

This might seem at first to allow for multiple sessions at the cost of degraded performance, thanks to the vehicle of TCP retransmission. The flaw with this arrangement is best indicated by the following scenario: Assume user A has FTP'd to an Internet host using the gateway and user B has managed to establish a concurrent Telnet session to another Internet host. If both are idle, and then the FTP user begins a very large file transfer, the Telnet user will be effectively disconnected until the file transfer completes.

This scenario argues for setting the connection timeout ("CONNECTION_TIMEOUT" in syuser.h) to a large value. If this is done, however, a single user could monopolize the gateway with a fairly inactive session. Thus, it is proposed that a "production-quality" Sytek-to-Ethernet gateway must have multiple Sytek session capability. Section 4.2.2 analyzes the possibilities for expanding to multiple sessions.

4.2 Future Work

In the previous section, we examined some of the constraints on the current gateway architecture. This section attempts to address these limitations with the intention of proposing some possible solutions to the problems caused.

4.2.1 Improvements to Current Architecture

As previously indicated, the prototype gateway is not robust enough to provide production-quality support to even a small group of Sytek users. It does, however, provide some connectivity to the Sytek Network user community. It is assumed, therefore, that while strides are being made to develop a gateway capable of fully integrating the Sytek

Network into the UA Campus Internet, the prototype may be pressed into service. The purpose of this section is to discuss enhancements to the prototype gateway which will improve it's performance should it be used as a temporary solution.

4.2.1.1 Putting Gateway Into Production

The prototype gateway has been extensively tested in the Computer Engineering Research Laboratory (CERL), and it's functionality has been verified. Nevertheless, some work (mostly administrative) should be done if it is to be put into production (i.e., made available for general use).

One of the most popular traditional TCP/IP "applications" is electronic mail. Numerous Internet "gurus" maintain that electronic mail is primarily responsible for the explosive growth of the Internet. In any case, few can doubt the usefulness of a global mail system which delivers messages in a matter of minutes instead of days.

One of the by-products of the prototype gateway development has been a TCP/IP protocol suite for Sytek users. This includes the Simple Mail Transfer Protocol (SMTP). Thus, putting the prototype gateway into

production effectively creates the potential for a whole new community of email users.

Of course, SMTP is not a "mailer" per se. It doesn't help users send and receive mail; It is simply a mail delivery protocol to allows hosts to send and receive mail. There is a mailer, called "bm", designed for use with KA9Q. It is in the public domain, and is available via FTP access. See Appendix B for details.

One final note about Sytek mail: Most Sytek hosts will be most likely be PCs. SMTP was designed for use with "always connected hosts" (i.e., always connected to the network unless the system is down). PCs often are not running the networking software and thus are effectively disconnected; PCs are not "always connected hosts". The solution is to have "post office" systems which agree to receive mail for hosts which are not always connected, and which will forward the mail to that host upon request. "Post office" systems run some kind of "Post-Office Protocol" (POP). Interested readers may consult RFC1083, "Post Office Protocol", for more details.

The remainder of this section deals with administrative details necessary for a smooth integration of the Sytek Network into the UA Campus Internet.

First and foremost, Sytek Network users should be apprised of the limited and temporary nature of the prototype gateway.

As is generally required for any new subnetwork, there must be an administrator responsible for that network, a Sytek Network manager. This person's duties would include managing the Sytek Network's address space (128.96.68.xxx) and domain namespace (xxxxx.ece-sytek.arizona.edu); he/she must assign IP addresses and proper domain names to hosts which participate in the Sytek subnetwork.

There must also be a host acting as a nameserver for the "ece-sytek.arizona.edu" domain. It should be noted that this system need not be a part of the Sytek subnetwork; a host on another network can act as the proxy nameserver for the "ece-sytek.arizona.edu" domain. It should be noted that the latest KA9Q release only implements a DNS client, but there is a KA9Q DNS server, available from a third party, which is in the public domain. For more information, requests should be sent to the "pcip@udel.edu" internet newsgroup.

Finally, it is the recommendation of the Sytek driver's author that the Sytek Driver be put in the public domain at some time in the future. In the course of this research,

I have been an active member of the "pcip@udel.edu" internet newsgroup; at least two other members of that group have expressed interest in implementing TCP/IP over their own Sytek networks.

KA9Q is a public domain software package without which this research would have been much more difficult. It is through sharing of resources such as these that a greater global connectivity is achieved.

4.2.1.2 Improving Throughput

Real-time responsiveness is of first importance for a gateway running in a LAN environment, where the data traffic is highly bursty, but has a low average volume. It should be clear, however, that achieving acceptable real-time responsiveness does not increase gateway throughput; high throughput is achieved through eliminating system overhead. [44]

In this section we discuss some improvements to the Sytek driver which should increase the prototype gateway's throughput. It should be noted that the speed of the gateway's Ethernet interface is much greater than that of the Sytek interface, thus it can easily be assumed that improving the throughput of the Sytek Driver will

proportionately improve the throughput of the gateway.

Recall that each layer of the protocol stack prepends it's own header to the actual data to be transferred. It can be shown that decreasing the header-to-data ratio in each network frame will result in better utilization of the network bandwidth, and thereby, greater throughput. This can be accomplished in two ways: 1) by decreasing the actual size of a protocol layer's header, and 2) by header compression.

As for the first approach, we can only revise the way the Sytek driver frames IP datagrams. Obviously, we cannot redefine IP or any of the other standard TCP/IP protocols. Specifically, we can modify our Sytek Header design. Currently there is only one header format which is used for all flavors of Sytek frames (i.e., data frames, UD requests, UD responses, disconnect frames). This was the easiest scheme to implement and debug, but it impairs our throughput. By designing Sytek headers specific to each of the 5 types of frames, such that only necessary fields are included in each (i.e., data frames wouldn't have UD-specific fields), we will increase the throughput of our driver.

Even more wasted bandwidth can be squeezed out of the Sytek header format when one considers the current representation of Sytek addresses "on the wire". Recall that a Sytek address is a 16-bit unit ID and a 4-bit port ID. Currently, they appear in Sytek headers using seven bytes. By encoding the 20-bits of each Sytek address into 3 bytes, we will save four bytes of header per Sytek address in that frame. Doing this to the current, single-header format (which has three Sytek addresses) would decrease the Sytek header size by 12 bytes, which is 35%.

The second way to decrease the header-to-data ratio mentioned above, header compression, has recently been the subject of much research in the Internet community. For a detailed overview of this technique, the reader should consult RFC1144, "Compressing TCP/IP Headers for Low-Speed Serial Links".

Another major throughput bottlenecks in the Sytek driver design is due to the character stuffing added to SLIP. This was done to avoid having the PCU command escape character inadvertently appear in the data stream. It is necessary if we wish to operate the PCU in "data mode" for PCU connections. Fortunately, there is another mode for Sytek sessions, called "transparent mode", which passes all

data without looking for command escape characters. When the PCU user wishes to return to command mode, he/she sends a BREAK signal to the PCU.

To increase throughput, a developer could remove the extra character stuffing in SLIP, and use the PCU in transparent mode. Of course, the Connection Handler would have to be modified to send a BREAK signal instead of the command escape characters when terminating a session. Preliminary analysis has indicated that a throughput increase of approximately 10% could be realized from this modification. For more information on the PCU "transparent mode", see the "LocalNet 20 Reference and Installation Guide", [38].

Finally, under heavy workload, a Sytek session may drop an occasional character or two (i.e. - Sytek does not insure an error-free connection). Fortunately, this does not cause total failure, since IP will perform a checksum validation on the datagram, and if it fails the datagram is discarded. This fact does, however, influence our performance. For instance, on an error-free link, we could set the TCP Maximum Segment Size (TCP MSS) to a very large value, and thereby approach ideal throughput (since this decreases the header:data ratio). If TCP must occasionally

re-transmit a datagram, however, there is a point at which increasing TCP MSS will actually decrease throughput. It is proposed that testing be done to find the ideal TCP MSS value for the current architecture. It is also suggested that addition of an error correction and detection checksum (such as the 16-bit CRC-CCITT) to the Sytek header be considered. Even though this would be increasing the header size, it would allow TCP MSS to be increased, and the resultant gain in throughput may be substantial.

4.2.1.3 Improvements to Unit Discovery

As previously mentioned, the Unit Discovery protocol's primary weakness is that it is inherently centralized. Problems arise when the UD server(s) become unavailable due to system crashes, or simply because they are busy with other users. The current UD design, however, is most likely sufficient for a small user base. This section deals with extensions and improvements to UD which will make it suitable for a larger user base.

Currently the UD database is a simple linked list with the server(s) in front. As the number of Sytek users increases, the number of UD entries in the gateway's database will increase. It is recommended that if the

Sytek user base becomes fairly large, the UD database be migrated to a hash table, which will allow for faster UD table lookups.

Another shortcoming of the current UD design is that information in the database is always assumed to be valid. If we manage to get an incorrect IP/Sytek address pair in our database, we're basically stuck with it. The only way to eradicate it is by receiving a UD_RESPONSE from the named Sytek address, with a new IP address which UD will use to correct the erroneous entry. To keep consistency in the UD database, it is therefore recommended that the UD protocol be modified to remove any UD entries which have not been used for some period of time, or to which repeated connection attempts have failed.

Another possible way to keep the UD databases up-to-date, at least in the UD server(s), is to have hosts send unsolicited UD_RESPONSE frames to the UD Server(s) upon connecting to the Network.

As we have said, UD in it's current incarnation is fairly centralized. By making the UD algorithm more distributed, we can hope to combat this somewhat. The easiest way to do this is by implementing multiple UD servers. The functionality for this already exists in the

software; all that need be done is to administratively assign the UD server task to a number of hosts which have a high degree of availability, and make the proper entries in the UD databases. Future developers may decide to augment this scheme somewhat by modifying the server types to create two classes of servers, primary and secondary. This is left to the consideration of future developers.

Currently in the CERL, the prototype gateway is configured to also act as the UD server. This is a distinctly bad arrangement, since the gateway is single-session only. For example, if a Sytek user has established a session through the gateway, no other Sytek user can resolve an IP address. This may stop Sytek user A from communicating with Sytek user B (n.b., except for UD they wouldn't need the gateway) if Sytek user C has a gateway session established.

It is recommended that if the prototype gateway is put into operation, the UD server be implemented on some other "always connected" host (if one is available). If a future gateway had multi-session capability (as is anticipated) the UD server function could be migrated back into the gateway. Recall that each Sytek host will have the capability to be a UD server. In fact, the software is

written such that each host has no idea if it is a server or not, each simply answers any UD request frames it receives. UD servers are identified in the databases of other systems.

4.2.1.4 General Improvements

This section is provided to briefly discuss some other possibilities for improving the prototype gateway. The refinements presented here are nonessential to proper operation of the gateway, but may be considered as possible development projects.

Currently the Connection Handler database is a simple linked list. As the Sytek session capacity of the gateway increases, the number of entries in the CH database will increase. It may be prudent to migrate the CH database to a faster access-time data structure such as a hash table.

As previously mentioned, KA9Q's strengths lie in it's usefulness for development systems and non-end-user systems such as gateways. Another possible development project would be to migrate the protocol stack in the Sytek workstations into another public domain TCP/IP software package. If a package such as NCSA Telnet were selected, users could get all the fancy user features absent from a

KA9Q workstation, such as a VT100 emulator. As another approach, future developers may consider making improvements to the KA9Q user interface.

It should be mentioned that the whole protocol stack could not be easily implemented with NCSA Telnet, since it was designed to be an "end system", not an "open system" (i.e., it has no routing code).

Finally, the following simplification is proposed: Eliminate the "Passive Listener Daemon" entirely and have asy_rx discard data heard during passive mode, until a FR_END. Any messages from the PCU (e.g., "CALL RECEIVED FROM E905,0") will not have FR_END's and will be discarded. The first frame sent on a new connection should be of type SYTEK_CONNECT_FRAME (which the "connect_daemon" routine, should be modified to do). Additional handling of UD frames would be necessary (i.e., a UD request frame is implicitly a SYTEK_CONNECT_FRAME).

4.2.2 Expanding to Multiple Sessions

In Section 4.1.3 we concluded that a "production-quality" Sytek-to-Ethernet gateway must have multiple Sytek session capability. It is assumed that making the Sytek Network a viable subnetwork of the UA Campus Internet would

require the connectivity of such a "production-quality" gateway.

The multiple session capability we need can be achieved in three ways:

Implementing Multiple Single-Session Gateways

This approach will give the desired multiple Sytek session capability, but it has definite drawbacks. Firstly, it is very expensive; Each additional Sytek session "costs" a PC, an Etherlink card and a PCU box.

Secondly, it is not the "plug-and-play" solution it might seem to be. Generally when two networks are connected by two or more gateways, they are present to help avoid congestion. That is, a user trying to use heavily loaded gateway A might be instructed by A to re-direct his route to gateway B (via an ICMP re-direct message). This is impossible for our prototype gateways, since they have two states (from a host's perspective): idle or unreachable; A prototype gateway which is being used cannot send ICMP re-direct messages to subsequent users. It would take some fairly complex extensions to the current driver to give hosts the ability to make intelligent routing decisions based on the state of the gateways.

For these reasons, this technique is not recommended for a solution to this problem.

Increase Sessions Per Gateway PCU

Another possible solution is to increase the number of sessions for the prototype gateway's single PCU port. Sytek PCU's have the capability to manage several simultaneous sessions [38], and we could re-design our Sytek driver to utilize this, giving the gateway multi-session capability.

Unfortunately, this approach does has some serious drawbacks. For instance, the modifications to the Sytek driver would be many. It would require a major re-write. The resulting architecture would have to ensure fair treatment of all sessions. Also, this solution sets an upper bound on the number of sessions the gateway can have. A LocalNet 20/100 PCU has the capability for managing four simultaneous Sytek sessions. A LocalNet 20/200 PCU is better, allowing 16 sessions, but this still represents an absolute upper bound on this approach.

For these reasons, this technique is also not recommended.

Increase Number of Gateway PCUs

The third approach to giving the gateway multi-session capability is to add many serial ports and PCUs to the gateway machine. The Sytek driver software could then be extended to manage these. The proposed software architecture would be to layer a "generic interface" on top of the multiple "physical interfaces" associated with each single-session PCU box. These physical interfaces would implementations of the current driver.

From the standpoint of IP addressability, the gateway would have one IP address on the Sytek subnetwork, associated with the generic interface. From the standpoint of Sytek addressability, the group of PCUs associated with the gateway's generic interface could be formed into a "rotary group", which permits it's participating PCUs to share the same Sytek address. A call to the address of this "rotary group" would establish a session with the gateway through any of its PCU which had a session available. See p.3-5 of the "LocalNet 20 Reference and Installation Guide", [38], for more details concerning the Sytek rotary dialing scheme.

It should be noted that just such an architecture was developed for the KA9Q package to implement a "NET/ROM"

generic interface over multiple "AX25" packet radio physical interfaces. The details of this design may be found in the original KA9Q code (the code specific to packet radio networks was removed during the course of this research to save memory).

Incidentally, it is not recommended that TCP sessions be mapped to Sytek sessions; This is a dangerous violation of the layering paradigm. Rather, future versions continue to allow idle Sytek sessions to timeout. This is more economical (idle TCP sessions will not waste valuable Sytek sessions). Such an arrangement might very well lead to sporadically active TCP sessions which "float" among the gateway's physical interfaces, but such an arrangement, while seemingly bizarre, shouldn't present a problem.

The approach presented in this section is recommended to expand the gateway to multiple sessions. It should require the least software development, and it has no absolute upper bound on the number of sessions which can be added to the gateway.

4.2.3 Porting Gateway to a High Performance Platform

The prototype gateway software, KA9Q, includes it's own operating system kernel, the Network Operating System. NOS

itself is small, fast and efficient. Unfortunately, under the current implementation it is layered on top of MS-DOS, which is large, slow and inefficient. This seriously hampers the gateway's performance.

A production-quality gateway should have as little OS overhead as possible. Under the current architecture, this overhead is tremendous. Fortunately KA9Q was designed to be highly portable. In particular, it has been ported to numerous flavors of Unix with a great deal of success. There is even a coordinator for Unix ports of the KA9Q code, Bob Hoffman at Pittsburgh University. He is available via electronic mail as <hoffman@vax.cs.pittsburgh.edu>.

A Unix environment for the gateway software is preferable for a number of reasons. Unix is small, fast, and efficient. It was designed to be a multi-tasking OS, whereas MS-DOS was not (and NOS had to implement multi-tasking primitives on top of it). Since KA9Q has already been ported to it, the transition from the current platform to a Unix-based one should be relatively painless.

It would be possible to purchase a Unix OS kernel for the current gateway machine (a 386-class PC AT), but this is not recommended for two reasons: It has a fairly

limited multiple-session capability (expandability to three serial ports, at most), and it may very well be unable to handle the heavy throughput which will be demanded of the production gateway.

Rather, it is recommended that while porting the gateway software to a Unix environment, it also be moved to a more robust platform, such as a 25 MIPS Unix Workstation, like the ones made by Sun and DEC.

4.2.4 Extension to Other LANs

The end goal of the generic gateway research project is to approach comprehensive campus interconnectivity. Such a scheme would involve designating each campus LAN as a subnetwork of the UA Campus Internet and providing an IP-router between that LAN and a participating campus Ethernet. KA9Q provides a robust platform for the development of the new network drivers which would be necessary to implement these IP-routers.

Incidentally, KA9Q may provide more interconnectivity than just Sytek-to-Ethernet. KA9Q provides support for a generic network hardware interface called a "packet driver". The "packet driver" concept was inspired by KA9Q, and developed by FTP Software, Inc. A "packet driver" is

a specific network interface driver which can be installed in memory by running a TSR (Terminate and Stay Resident) program. It's interface to the networking software is defined in terms of non-subnet specific primitives. Thus the networking software sees only the generic packet driver interface, with no understanding of the particular subnet needed.

A library of packet drivers for myriad hardware network interfaces has been developed. The packet driver library in the public domain includes drivers for numerous Ethernet cards, the IBM Token Ring Adapter card and the various flavors of AT&T's StarLAN. What this translates to in terms of UA campus connectivity is that the KA9Q package may be used to implement additional IP-routers, such as one between an Ethernet LAN and a Token Ring LAN, as shown in Figure 3.2.

To implement an IP-router between Ethernet and any candidate network for which there is a public domain packet driver, no hardware or software design would be necessary. It would simply be a matter of configuring the KA9Q software to include packet driver interfaces for the candidate network and Ethernet, and specifying the gateway's routing tables properly.

To implement an IP-router between Ethernet and any other LAN, a developer should implement a packet driver for this network interface which meets the "PC/TCP Version 1.08 Packet Driver Specification", by FTP Software, Inc. This specification is reproduced in Section 8.4 of "The KA9Q Internet Software Package", [10]. To implement the desired router, it would then be a matter of configuring the KA9Q software to include packet driver interfaces for the candidate network and Ethernet, and specifying the gateway's routing tables properly.

APPENDIX A

USER MANUAL

This Appendix is provided as a user manual for the KA9Q software package which has been extended for use as a TCP/IP implementation on a Sytek Network. There are sections for the new Internet users on the Sytek Network, and the prototype gateway administrator.

A.1 For the Sytek User

This section is provided for a Sytek user of the KA9Q Internet Software Package. The original (non-NOS) KA9Q package was documented in "The KA9Q Internet Software Package", [10]. The user interface was not changed very much for the newer NOS version, so a newer KA9Q user's manual has never been written.

As part of this research, the packet radio code was removed from KA9Q to conserve memory. Also, a new network interface driver class was added - the "sytek" driver. These changes have been incorporated into the text presented here, which borrows heavily from "The KA9Q Internet Software Package", [10].

A.1.1 Getting Started

In order for a Sytek Network User to use the KA9Q Internet Software package, he/she must have an IBM PC compatible microcomputer running MS-DOS Version 3.3 or higher, 512 KBytes of main memory, an RS-232C serial port and one Sytek LocalNet 20/100 PCU. KA9Q can be run from a floppy disk or a hard disk. In either case, all the relevant files should be in the \NET subdirectory of the current drive.

There are five files of interest in this software package. Each are discussed below:

NET.EXE - This is the executable file for the KA9Q package. It includes all the code for the TCP/IP protocol stack and any non-packet driver network interfaces.

AUTOEXEC.NET - When NET.EXE is executed without arguments, it attempts to open the file "AUTOEXEC.NET". If it exists, it is read and executed as though its contents were typed on the console as commands. This feature is useful for setting the local IP address and host name, initializing the IP routing table, and starting the various Internet services. If NET.EXE is invoked with an argument, it is taken to be the name of an alternate startup file; it is read instead of AUTOEXEC.NET. An example of an AUTOEXEC.NET file that a Sytek user might use is discussed in A.1.4.

FTPUSERS - Since MS-DOS was designed as a single-user operating system, it provides no access control; all files can be read, written or deleted by the local user. It is usually undesirable to give such open access to a system to remote network users. The KA9Q FTP server therefore provides its own access control mechanism. The file

"\net\ftpusers" is used to control remote FTP access. The default is NO access; if this file does not exist, the FTP server will be unusable. A remote user must first "log in" to the system, giving a valid name and password listed in \net\ftpusers, before he or she can transfer files.

Each entry in \net\ftpusers consists of a single line of the form:

username password path1 permissions1

SYDOMAIN.TXT - This file is the Sytek domain database used by UD. It contains the seed entries for the UD address resolution database read in at startup. When "net" is exited, the most current UD database is written to this file. The entries in this text file are of the following format:

<ip-address> <Sytek-address> [<serv-indicator>]

where the first two parameters denote an IP/Sytek address pair, and the optional third parameter is a string denoting entries which are servers. UD Servers will have the string "SERVER" as the third field.

DOMAIN.TXT - This file is the local domain cache for the domain name resolver of KA9Q. It is used by the resolver alone - it's contents is not necessarily the same as you would put in a server database.

The only file required at startup is NET.EXE. FTPUSERS is necessary only if you wish to act as an FTP server. UD will build a SYDOMAIN.TXT file, if necessary. The domain resolver will likewise build DOMAIN.TXT. And AUTOEXEC.NET is not required, but highly recommended if you don't like typing.

When running KA9Q, the command line accepts several arguments that are best illustrated by example:

net /m300 /s40 /d\net \net\autoexec.net /t60

In this case, net is told to grab 300K of memory from MS-DOS for the heap, to allow a maximum of 40 active sockets, to make the directory prefix for all files (e.g. ftpusers, domain.txt) "\net", and set the Sytek session idle timeout timer to 60 seconds. These particular values all happen to be the defaults for their corresponding parameters, so of course you'd get the same effect by just typing "net".

A.1.2 Console Mode

The console may be in one of two modes: command mode and converse mode. In command mode, the prompt "net>" is displayed and any of the commands described in the next section may be entered. In converse mode, keyboard input is processed according to the "current session", which may be either a Telnet, or FTP connection. In a telnet session, keyboard input is sent to the remote system and any output from the remote system is displayed on the console. In an FTP session, keyboard input is first examined to see if it is a known local command; if so it is executed locally. If

not, it is "passed through" to the remote FTP server.

The keyboard also has "cooked" and "raw" states. In cooked state, input is line-at-a-time; the user may use the line editing characters ^U, ^R and backspace to erase the line, redisplay the line and erase the last character, respectively. Hitting either return or line feed passes the complete line up to the application. In raw mode, each character is immediately passed to the application as it is typed. The keyboard is always in cooked state in command mode. It is also cooked in converse mode on an AX25 or FTP session. In a Telnet session it depends on whether the remote end has issued (and the local end has accepted) the Telnet "WILL ECHO" option.

On the IBM-PC, the user may escape back to command mode by hitting the F10 key.

A.1.3 KA9Q Command Reference

This section describes each of the commands recognized while in command mode. Note that certain FTP subcommands, (e.g., put, get, dir) are recognized only in converse mode with the appropriate FTP session; they are not recognized while in command mode. The notation "<hostid>" denotes a host or gateway, which may be specified in one of two ways: as a symbolic name which is an ARPA-style domain name (e.g., london.ece.arizona.edu), or as a numeric IP address in dotted decimal notation enclosed by brackets, e.g., [128.196.28.12].

<cr> - Entering a carriage return (empty line) while in command mode puts you in converse mode with the current session. If there is no current session, net remains in

command mode.

! - An alias for the "shell" command.

- Commands starting with the hash mark (#) are ignored. This is mainly useful for comments in the AUTOEXEC.NET file.

arp - With no arguments, displays the Address Resolution Protocol table that maps IP addresses to their subnet (link) addresses on subnetworks capable of broadcasting. For each IP address entry the subnet type (e.g., Ethernet), subnet address and time to expiration is shown. If the link address is currently unknown, the number of IP datagrams awaiting resolution is also shown.

arp add <hostid> ether <ether addr> - The add subcommand allows manual addition of address resolution entries into the table. This is useful for "hard-wiring" paths that are not directly resolvable.

arp drop <hostid> ether - The drop subcommand allows removal of entries from the table.

arp flush - Removes all entries from our ARP tables.

arp publish <hostid> ether <ether addr> - The publish subcommand allows you to respond to arp queries for some other host. This is commonly referred to as "proxy arp", and is considered a fairly dangerous tool. The basic idea is that if you have two machines, one of which is on an Ethernet, and the second one of which is connected to the first with a slip link, you might want the first machine to publish it's own Ethernet address as the right answer for arp queries addressing the second machine. This way, the rest of the world doesn't know the second machine isn't really directly connected. Use arp publish with caution.

asystat - Gives a brief status summary of all asynchronous network interfaces (slip and sytek).

attach asy <address> <vector> slip|sytek <label> <snd_bufsiz> <rcv_bufsiz> <speed> [<ip_addr>] - Configure and attach an asynchronous hardware interface to the system. There are two types: SLIP and Sytek, selectable by the third parameter. <address> is the base address of the control registers for the device. <vector> is the interrupt vector number. Both the address and the vector must be in

hexadecimal. (You may put "0x" in front of these two values if you wish, but note that they will be interpreted in hex even if you don't use it). <label> gives the name by which the interface will be known to the various commands, such as "route" and "trace". <snd_bufsiz> and <rcv_bufsiz> specify the size of the ring buffers in bytes to be statically allocated to the sender and receiver, respectively; incoming bursts larger than this may (but not necessarily) cause data to be lost. <speed> indicates the baud rate to which the controller should be initialized. <ip_addr> is an optional parameter which can be used to assign this interface an ip address different than that specified in a previous "ip addr" command.

attach packet <int#> <label> <buffers> <mtu> [<ip_addr>] - Configure and attach an interface for a "packet driver" to the system. <int#> is the software interrupt number which the packet driver has been installed to use. <label> gives the name by which the interface will be known to the various commands, such as "route" and "trace". For an Ethernet packet driver (the most common type), <buffers> specifies how many PACKETS may be queued on the receive queue at one time; if this limit is exceeded, further received packets will be discarded. This is useful to prevent the system from running out of memory should another node suddenly develop a case of diarrhea. <mtu> is the Maximum Transmission Unit size, in bytes. Datagrams larger than this limit will be fragmented at the IP layer into smaller pieces. <ip_addr> is an optional parameter which can be used to assign this interface an ip address different than that specified in a previous "ip addr" command.

cd [<dirname>] - Changes directory on the local machine.

close [<session #>] - On a FTP or Telnet session, this command sends a FIN (i.e., initiates a close) on the session's TCP connection. This is an alternative to asking the remote server to initiate a close ("QUIT" to FTP, or the logout command appropriate for the remote system in the case of Telnet). When either FTP or Telnet sees the incoming half of a TCP connection close, it automatically responds by closing the outgoing half of the connection. Close is more graceful than the "reset" command, in that it is less likely to leave the remote TCP in a "half-open" state.

dir [<dirname>] - List the contents of the specified directory on the console. If no argument is given, the

current directory is listed.

disconnect [<session #>] - An alias for the "close" command.
delete [<filename>] - Deletes a file from the current local directory.

detach [<interface>] - Detaches a network interface. This command is the opposite of the attach command (c.f.).

domain addserver [<ip address>] - Adds a domain nameserver to our list, specified by it's IP address.

domain dropserver [<ip address>] - Drops the indicated domain nameserver from our list.

domain listservers - Displays the list of domain servers along with various statistics measured for each.

echo [accept|refuse] - Displays or changes the flag controlling client Telnet's response to a remote WILL ECHO offer.

The Telnet presentation protocol specifies that in the absence of a negotiated agreement to the contrary, neither end echoes data received from the other. In this mode, a Telnet client session echoes keyboard input locally and nothing is actually sent until a carriage return is typed. Local line editing is also performed: backspace deletes the last character typed, while control-U deletes the entire line.

Many timesharing systems (e.g., UNIX) prefer to do their own echoing of typed input. (This makes screen editors work right, among other things). Such systems send a Telnet WILL ECHO offer immediately upon receiving an incoming Telnet connection request. If "echo accept" is in effect, a client Telnet session will automatically return a DO ECHO response. In this mode, local echoing and editing is turned off and each key stroke is sent immediately (subject to the Nagle tinygram algorithm in TCP). While this mode is just fine across an Ethernet, it is clearly inefficient and painful across slow paths like SLIP links. Specifying "echo refuse" causes an incoming WILL ECHO offer to be answered with a DONT ECHO; the client Telnet session remains in the local echo mode. Sessions already in the remote echo mode are unaffected. (Note: Berkeley Unix has a bug in that it will still echo input even after the client has refused the WILL ECHO offer. To get around this problem, enter the "stty -echo" command to the shell once you have logged in.)

eol [unix|standard] - Displays or changes Telnet's end-of-line behavior when in remote echo mode. In standard mode, each key is sent as-is. In unix mode, carriage returns are translated to line feeds. This command is not necessary with all UNIX systems; use it only when you find that a particular system responds to line feeds but not carriage returns. Only Sun UNIX release 3.2 seems to exhibit this behavior; later releases are fixed.

exit - Exit the "net" program and return to MS-DOS.

finger [<user>]@<hostid> - Issues a finger query to the remote finger daemon at <hostid>. The data returned by that daemon is printed. If <user> is specified, the remote daemon returns only information specific to this user. Otherwise, information about all logged in users at that host are returned.

ftp <hostid> - Open an FTP control channel to the specified remote host and enter converse mode on the new session.

When in converse mode with an FTP server, everything typed on the console is first examined to see if it is a locally-known command. If not, the line is passed intact to the remote server on the control channel. If it is one of the following commands, however, it is executed locally. (Note that this generally involves other commands being sent to the remote server on the control channel.) When actively transferring a file, the only acceptable command is "abort"; all other commands will result in an error message. Responses from the remote server are displayed directly on the screen.

ftp abort - Aborts a get, put or dir operation in progress. When receiving a file, abort simply resets the data connection; the next incoming data packet will generate a TCP RST (reset) in response which will clear the remote server. When sending a file, abort sends a premature end-of-file. Note that in both cases abort will leave a partial copy of the file on the destination machine, which must be removed manually if it is unwanted. Abort is valid only when a transfer is in progress.

ftp dir [<file>|<directory> [<localfile>]] - Without arguments, "dir" requests that a full directory listing of the remote server's current directory be sent to the terminal. If one argument is given, this is passed along

in the LIST command; this can be a specific file or sub-directory that is meaningful to the remote file system. If two arguments are given, the second is taken as the local file into which the directory listing should be put (instead of being sent to the console). The PORT command is used before the LIST command is sent.

ftp get <remote_file> [<local_file>] - Asks the remote server to send the file specified in the first argument. The second argument, if given, will be the name of the file on the local machine; otherwise it will have the same name as on the remote machine. The PORT and RETR commands are sent on the control* channel.

ftp ls [<file>|<directory> [<localfile>]] - ls is identical to the "dir" command except that the "NLST" command is sent to the server instead of the "LIST" command. This results in an abbreviated directory listing, i.e., one showing only the file names themselves without any other information.

ftp mkdir <remote_directory> - Creates a directory on the remote machine.

ftp put <local_file> [<remote_file>] - Asks the remote server to accept data, creating the file named in the first argument. The second argument, if given, will be the name of the file on the remote machine; otherwise it will have the same name as on the local machine. The PORT and STOR commands are sent on the control channel.

ftp rmdir <remote_directory> - Deletes a directory on the remote machine.

ftp type [a|i|l<bytesize>] - Tells both the local client and remote server the type of file that is to be transferred. The default is 'a', which means ASCII (i.e., a text file). Type length lines of text in ASCII separated by cr/lf sequences; in IMAGE mode, files are sent exactly as they appear in the file system. ASCII mode should be used whenever transferring text between dissimilar systems (e.g., UNIX and MS-DOS) because of their different end-of-line and/or end-of-file conventions. When exchanging text files between machines of the same type, either mode will work but IMAGE mode may be somewhat faster. Naturally, when exchanging raw binary files (e.g., executables) IMAGE mode must be used. Type 'l' (logical byte size) is used when exchanging binary files with remote servers having oddball

word sizes (e.g., DECSYSTEM-10s and 20s). Locally it works exactly like IMAGE, except that it notifies the remote system how large the byte size is. <bytesize> is typically 8. The type command sets the local transfer mode and generates the TYPE command on the control channel.

help - Display a brief summary of top-level commands.

hostname [<name>] - Displays or sets the local host's name (an ASCII string such as "fred@ece-sytek.ece.arizona.edu", NOT an IP address). Currently this is used only in the greeting messages from the SMTP (mail) and FTP (file transfer) servers.

icmp echo [on|off] - Sets the local ICMP's "echo response" flag according to the parameter. If none is given, this flag's current value is reported.

icmp status - Prints out a summary of ICMP messages sent and received by the local ICMP.

icmp trace [on|off] - Turns ICMP tracing on and off. If no parameter is given, the current trace setting is reported.

ifconfig [<iface>] - Lists the interface specified by <iface> along with its parameters (e.g., IP address, send and receive packet counts). If no parameter is provided, this command lists all the interfaces on the system.

ifconfig <iface> ipaddr [<ip addr>] - Changes the IP address for the interface specified in <iface> to the value specified by <ip addr>.

ip address [<hostid>] - Displays or sets the local IP address.

ip rtimer [<val>] - Displays or sets the default rtimer parameter for this IP implementation.

ip status - Displays Internet Protocol (IP) statistics, such as total packet counts and error counters of various types. Also displays statistics about the Internet Control Message Protocol (ICMP), including the number of ICMP messages of each type sent or received.

ip ttl [<val>] - Displays or sets the default time-to-live value placed in each outgoing IP datagram. This limits the

number of switch hops the datagram will be allowed to take. The idea is to bound the lifetime of the packet should it become caught in a routing loop, so make the value somewhat larger than the diameter of the network.

log [stop|<file>] - Without arguments, indicates whether server sessions are being logged. If "stop" is given as the argument, logging is terminated (the servers themselves are unaffected). If a file name is given as an argument, server session log entries will be appended to it.

memstat - Displays the internal free memory list in the storage allocator.

more <file> - Browse or page through a text file, similar to the Unix "more" command.

param <interface> [param] - Param invokes a device-specific control routine. On a SLIP or Sytek interface, the param command allows the baud rate to be read (without arguments) or set. The implementation of this command for the various interface drivers is incomplete and subject to change.

ping <hostid> - Sends an ICMP echo request packet to the specified host. Pings are used to establish reachability.

ps - Displays the status of current NOS processes.

pwd [<dirname>] - An alias for the cd command.

record [<filename>|off] - Opens <filename> and appends to it all data received on the current session. Data sent on the current session is also written into the file except for Telnet sessions in remote echo mode. The command "record off" stops recording and closes the file. This command is not supported for FTP sessions.

rename <file1> <file2> - Renames a local file from <file1> to <file2>.

reset [<session>] - If an argument is given, force a local reset (deletion) of the TCP Control Block (TCB) belonging to the specified session. The argument is first checked for validity. If no argument is given, the current session, if any, is used. This command should be used with caution since it does not inform the remote end that the connection no longer exists. A reset (RST) message will be

automatically generated should the remote TCP send anything after a local reset has been done. This is used to get rid of a lingering half-open connection after a remote system has crashed.

rip - These commands support RIP, the interior routing protocol made popular by Berkeley UNIX. To enable the reception of RIP broadcasts, merely say "start rip" (this starts the RIP "server"; c.f. "start").

If you're not acting as a gateway to anyone, this is all you need do -- your system will begin to passively monitor its interfaces for broadcast routing packets and it will automatically add routes to the routing table. It may take up to 30 seconds on an ethernet for the table to be built (this assumes a broadcast rate of 30 seconds, which is standard on Ethernet). If you want to get started faster, you can give an optional IP address to the command: "start rip <ip addr>". This sends a RIP "request" packet to <ip addr> which triggers that gateway (or gateways if a broadcast ip address is given) to send you its routing tables.

rip accept <hostid> - Used for RIP filtering, see "rip refuse" below.

rip add <hostid> <interval> <flag> - This command generates periodic "unsolicited RIP responses". It means "send your routing tables once every <interval> seconds to <hostid>". The last parameter, <flag>, is the sum of the following possible flag values:

1 - Include a host-specific route to yourself in each update. (Not needed if you're already advertising a route to the network you're on.)

2 - Use split horizon updating; that is, omit all routing entries that point to the interface being used for the broadcast. (This reduces the chances of routing loops forming).

4 - Generate triggered updates as necessary on this interface, i.e., whenever a metric changes in the routing table, immediately generate a broadcast on this interface with the changed entries. If split horizon (bit 2) is also set, use "poisoned reverse", i.e., for any routing table entries that point to this interface, include them with an infinite metric

(RIP defines 16 to be infinity) instead of leaving them out as happens during a normal routing broadcast when split horizon is set. Triggered updates helps spread the word faster when links fail, reducing the chances of a temporary loop forming.

rip drop <hostid> - Removes a RIP periodic update list entry. This command counteracts a "rip add".

rip merge [on|off] - Enables/Disables route merging. If you say "rip merge on", then an incoming route that is more specific than one you already have in your table is ignored if they both point to the same gateway. For example, if you already have a default route that points to gateway "foobar", then any route that arrives from gateway foobar will be ignored because to put it in the table would not cause any change in the routing of packets -- they'd still go to foobar anyway. Properly used, this should save a lot of routing table space.

rip refuse <hostid> - If you want to ignore routing broadcasts from a certain gateway (e.g., because it can't hear you), use this command, where <hostid> identifies the offending gateway. To reverse this, use the "rip accept" command.

rip request <hostid> - This sends a RIP "request" packet to <ip addr> which triggers that gateway (or gateways if a broadcast ip address is given) to send you its routing tables.

rip status - Generates a summary of RIP traffic sent and received locally.

rip trace [0|1|2] - You can trace the automatic routing messages and controls by the "rip trace" command; it takes a numeric parameter. "rip trace 0" disables tracing, "rip trace 1" generates messages only when routes change, and "rip trace 2" shows you everything, even when things are stable.

route - Displays the IP routing table.

route add <dest hostid>[/bits]|default <interface> [<gateway hostid> [<metric>]] - Adds an entry to the routing table. This command requires at least two more arguments, the host

id of the target destination and the local name of the interface to which its packets should be sent. If the destination is not local, the gateway's host id should also be specified. (If the interface is a point-to-point link, then <gateway hostid> may be omitted even if the target is non-local because this field is only used to determine the gateway's link level address, if any. If the destination is directly reachable, <gateway hostid> is also unnecessary since the destination address is used to determine the interface link address).

The optional "/bits" suffix to the destination host id specifies how many leading bits in the host id are to be considered significant in the routing comparisons. If not specified, 32 bits (i.e., full significance) is assumed. With this option, a single routing table entry may refer to many hosts all sharing a common bit string prefix in their IP addresses. For example, ARPA Class A, B and C networks would use suffixes of /8, /16 and /24 respectively; the command "route add [44]/8 sl0 [44.64.0.2]" causes any IP addresses beginning with "44" in the first 8 bits to be routed to [44.64.0.2]; the remaining 24 bits are "don't-cares".

When an IP address to be routed matches more than one entry in the routing table, the entry with largest "bits" parameter (i.e., the "best" match) is used. This allows individual hosts or blocks of hosts to be exceptions to a more general rule for a larger block of hosts.

The special destination "default" is used to route datagrams to addresses not in the routing table; it is equivalent to specifying a /bits suffix of /0 to any destination hostid. Care must be taken with default entries since two nodes with default entries pointing at each other will route packets to unknown addresses back and forth in a loop until their time-to-live (TTL) fields expire. (Routing loops for specific addresses can also be created, but this is less likely to occur accidentally).

route addprivate - Same as "route add" except the IP routing table entry created is marked as "private". "Private" routing table entries are not included in RIP responses.

route drop <hostid> - Deletes an existing entry from the IP routing table.

route flush - Deletes all entries from the IP routing table.

route lookup <hostid> - Displays the entry in the IP routing

table which would be used for traffic to the specified host.

session [<session #>] - Without arguments, displays the list of current sessions, including session number, remote TCP address and the address of the TCP control block. An asterisk (*) is shown next to the "current" session; entering <cr> at this point will put you in converse mode with that session. Entering a session number as an argument to the session command will put you in converse mode with that session.

shell - Suspends "net" and executes a "command processor" shell under MS-DOS). When the sub-shell exits, net resumes. Note that background activity, such as an FTP server, is also suspended while the sub-shell executes.

smtp gateway [<hostid>] - Displays or sets the host to be used as a "smart" mail relay. Any mail sent to a hostid not in the host table will instead be sent to the gateway for forwarding.

smtp kick - Run through the outgoing mail queue and attempt to deliver any pending mail. This command is periodically invoked by a timer whenever net is running; this command allows the user to "kick" the mail system manually.

smtp maxclients [<val>] - Displays or sets the maximum number of simultaneous outgoing SMTP sessions that will be allowed. The default is 10; reduce it if network congestion is a problem.

smtp timer [<val>] - Displays or sets the interval, in seconds, between scans of the outbound mail queue. For example, "smtp timer 600" will cause the system to check for outgoing mail every 10 minutes and attempt to deliver anything it finds, subject of course to the "maxclients" limit. Setting a value of zero disables queue scanning altogether, note that this is the default! This value is recommended for stand alone IP gateways that never handle mail, since it saves wear and tear on the disk drive.

smtp trace [<val>] - Displays or sets the trace flag in the SMTP client, allowing you to watch SMTP's conversations as it delivers mail. Zero (the default) disables tracing.

socket [<socket #>] - With no parameter, this command displays a summary of the current TCP sockets. If a

particular socket is specified by it's socket number, a more detailed summary of that socket and it's usage is displayed.

start - Starts the specified Internet server, allowing remote connection requests. A "start ?" command will list the servers currently available.

stop - Stops the specified Internet server, rejecting any further remote connect requests. Existing connections are allow to complete normally. A "stop ?" command will list the servers currently available.

tcp irtt [<val>] - Display or set the initial round trip time estimate, in seconds, to be used for new TCP connections until they can measure and adapt to the actual value. The default is 5 seconds. Increasing this when operating over slow channels will avoid the flurry of retransmissions that would otherwise occur as the smoothed estimate settles down at the correct value. Note that this command should be given before servers are started in order for it to have effect on incoming connections.

tcp kick <tcp_addr> - If there is data on the send queue of the specified tcb, this command forces an immediate retransmission.

tcp mss [<size>] - Display or set the TCP Maximum Segment Size in bytes that will be sent on all outgoing TCP connect request (SYN segments). This tells the remote end the size of the largest segment (packet) it may send. Changing MSS affects only future connections; existing connections are unaffected.

tcp reset <tcb_addr> - Deletes the TCP control block at the specified address.

tcp rtt <tcp_addr> <rtval> - Replaces the automatically computed round trip time in the specified tcb with the rttval in milliseconds. This command is useful to speed up recovery from a series of lost packets since it provides a manual bypass around the normal backoff retransmission timing mechanisms.

tcp status [<tcb_addr>] - Without arguments, displays several TCP-level statistics, plus a summary of all existing TCP connections, including TCB address, send and receive queue sizes, local and remote sockets, and connection state.

If <tcb_addr> is specified, a more detailed dump of the specified TCB is generated, including send and receive sequence numbers and timer information.

tcp window [<val>] - Displays or sets the default receive window size in bytes to be used by TCP when creating new connections. Existing connections are unaffected.

telnet <hostid> - Creates a Telnet session to the specified host and enters converse mode.

trace [<interface> [<flags>]|allmode|cmdmode] - Controls packet tracing by the interface drivers. Specific bits enable tracing of the various interfaces and the amount of information produced. Tracing is controlled on a per interface basis; without arguments, trace gives a list of all defined interfaces and their tracing status. Output can be limited to a single interface by specifying it, and the control flags can be change by specifying them as well. The flags are given as a hexadecimal number which is interpreted as follows:

```
TIO
|||--- Enable tracing of output packets if 1, 0=disable
 ||---- Enable tracing of input packets if 1, 0=disable
  |----- Controls type of tracing:
   0 - Protocol headers are decoded, data is not
       displayed.
   1 - Protocol headers are decoded, and data (but not the
       headers themselves) are displayed as ASCII
       characters, 64 characters/line. Unprintable
       characters are displayed as periods.
   2 - Protocol headers are decoded, and the entire packet
       (headers AND data) is also displayed in
       hexadecimal and ASCII, 16 characters per line.
```

There is an additional option for tracing, that allows you to select whether traced packets are always displayed, or only displayed when you are in command mode. Having tracing only happen in command mode sometimes provides the right mix between "knowing what's going on", and "keeping the garbage off the screen" while you're typing. To select tracing all the time (the default mode), use "trace allmode". To restrict tracing to command mode, use "trace cmdmode".

udp status - Displays the status of all UDP receive queues.

upload [<filename>] - Opens <filename> and sends it on the current session as though it were typed on the terminal. Valid only on Telnet sessions.

? - Same as the "help" command.

A.1.4 Example Configuration File

The following is an example configuration file for a Sytek PC running KA9Q:

```
Line    Text
1       hostname coventry.ece-sytek.arizona.edu
2       ip address [128.196.68.1]
3       attach asy 0x3f8 4 sytek sy0 2048 2048 19200
4       route add [128.196.68]/24 sy0
5       route add default sy0 [128.196.68.2]
6       domain addserver [128.196.28.12]
7       ip ttl 32
8       tcp mss 576
9       tcp window 432
10      log \net\net.log
11      start discard
12      start echo
13      start finger
14      start ftp
15      start ttylink
```

The first line defines our hostname for greeting messages from the SMTP and FTP servers. The second line defines our IP address. The third line attaches a Sytek interface called "sy0". This interface allocates send and receive buffers of 2 Kbytes. The associated asynchronous controller chip (probably an 8250) is identified as having an I/O address of 3F8, uses hardware interrupt #4, and runs

at 19200 baud.

Lines 4 and 5 define our routing tables. Quite simply, we direct any traffic for the [128.196.68] class C subnetwork to the sy0 interface, with no gateway necessary. The "sy0" interface knows how to deliver "locally". Any other traffic is routed by our default entry to the sy0 interface, with a gateway of identified as [128.196.68.2].

Line 6 adds a nameserver to our list. Any unresolved domain-type hostnames will be resolved via consultation to this system.

Lines 7-9 define some TCP/IP parameter values. Line 10 indicates a logfile for server session log entries to be appended to. Lines 11-15 start the various Internet servers this host implements.

Note that this file has no comment lines (those beginning with '#'). This was done here for brevity, but most configuration files will have comments.

A.2 For the Prototype Gateway Administrator

This section is provided as further information for the administrator of the Sytek-to-Ethernet gateway. This administrator must manage the Sytek Network namespace and IP address space, insuring that users are not assigned

conflicting names or IP addresses. He/she must also designate the address of the gateway system and UD server (if different) and let these be know to all Sytek users. Finally, he/she must configure the gateway system appropriately.

As an example of how the gateway might be configured, the configuration file from the prototype gateway test system is given below and discussed. The gateway was between an Ethernet class C subnetwork (128.196.28.0) and the Sytek class C subnetwork (128.196.68.0). It IP addresses were 128.196.28.25 and 128.196.68.2 respectively. The AUTOEXEC.NET file used:

```
Line    Text
1       hostname gengw.ece.arizona.edu
2       ip address [128.196.28.26]
3       attach packet 0x60 eth0 5 1500
4       route addprivate [128.196.28]/24 eth0
5       attach asy 0x3f8 4 sytek sy0 2048 2048 19200
6       route add [128.196.68]/24 sy0
7       ifconfig sy0 ipaddr [128.196.68.2]
8       route addprivate default eth0 [128.196.28.1]
9       domain addserver [128.196.28.12]
10      rip merge on
11      start rip
12      rip request [128.196.28.1]
13      arp add [128.196.28.0] ether ff:ff:ff:ff:ff:ff
14      rip add [128.196.28.0] 30 6
15      ip ttl 32
16      tcp mss 576
17      tcp window 432
18      tcp irtt 1000
19      log \net\net.log
20      start discard
21      start echo
22      start finger
```

```
23      start ftp
24      start ttylink
```

The first line defines our hostname for greeting messages from the SMTP and FTP servers. The second line defines our IP address. The third line attaches a "packet driver" network interface which is assumed to be for an Ethernet card. The forth line adds a routing table entry for the Ethernet subnetwork. It is private because we don't want to announce ourselves (via RIP) as a gateway to this system (we will not use RIP on the Sytek side). The fifth line attaches a Sytek interface called "sy0". This interface allocates send and receive buffers of 2 Kbytes. The associated asynchronous controller chip (probably an 8250) is identified as having an I/O address of 3F8, uses hardware interrupt #4, and runs at 19200 baud. Line 6 adds the routing table entry for the Sytek subnetwork. It is <u>not</u> private, because we do wish to advertise ourselves as a gateway to the network (via RIP broadcasts on the Ethernet).

At the time of their attachment, both interfaces are given the IP address assigned in line 2. Actually, the "sy0" interface should have an address on the Sytek subnetwork, since the gateway is a "multi-homed" host. Line 7 accomplishes this.

Line 8 adds our default route, with is to the Ethernet

using [128.196.28.1] as a gateway to the rest of the Internet. Line 9 identifies our DNS nameserver.

Lines 10-14 setup RIP for our gateway system. Line 10 enables RIP merging, to keep our routing tables small. Line 11 starts the RIP server. Line 12 sends an immediate RIP request to [128.196.28.1]. Line 13 sets up the [128.196.28.0] IP address as an Ethernet broadcast address, for use in line 14. Line 14 schedules the RIP broadcasting characteristics for the gateway itself (to notify external systems that it is the gateway to the Sytek subnetwork). It means "broadcast your routing tables every 30 seconds on the interface named "eth0", using IP destination address 128.96.160.0 (broadcast). Generate triggered updates as necessary, and use the split horizon method."

Lines 15-18 define some TCP/IP parameter values. Line 19 indicates a logfile for server session log entries to be appended to. Lines 20-24 start the various Internet servers the gateway implements.

APPENDIX B

ADVICE TO DEVELOPERS

This appendix is provided to assist with future development using the KA9Q code and the Sytek driver presented in this thesis. It attempts to disclose pertinent details which were learned in the course of this research, so as to save future developers from "re-inventing the wheel". There is a section outlining the Internet resources that may be of interest to a developer. There is also a brief overview of the NOS KA9Q Software architecture. Finally, the problems encountered in this research project are discussed.

B.1 Internet Resources

Most of the written information about the Internet, including it's architecture, protocols, and history, can be found in a series of reports known as Request For Comments or RFCs. An informal, loosely coordinated set of notes, RFCs are unusually rich in information and color [8]. Many RFCs were used as primary references for this thesis. Developers of TCP/IP software should be intimately familiar with those RFCs which pertain, however slightly, to their

work. This point cannot be over-stressed. For an excellent guide to the RFCs, consult "Internetworking with TCP/IP", [8], Appendix 3.

In the Internet community there are also many forums of discussion on a variety of topics. In Internet parlance, these are called "interest groups" (they are called "newsgroups" in the USENET world). An "interest group" is an association of Internet users who wish to discuss a particular topic. In order to do so, they subscribe to a mailing list, and any mail sent to the group is forwarded to them (and anyone else on the mailing list). The master list of Internet interest groups is available via FTP access to NIC.DDN.MIL as NETINFO:INTEREST-GROUPS.TXT.

In the course of this research, two applicable interest groups were of invaluable assistance. Their entries from the interest groups master list are reproduced below:

PCIP@UDEL.EDU

Discussion group for the various sets of TCP/IP implementations for personal computers. Bugs are reported here and help bringing up a new environment may be forthcoming from members of this list. In the past, discussions have included the MIT package, the Stanford TCP modifications and work at Wisconsin and Maryland.

Archives are available via an electronic mail server. Details about its use can be obtained by sending a request to PCIP-REQUEST@UDEL.EDU.

All requests to be added to or deleted from this list, problems, questions, etc., should be sent to

PCIP-REQUEST@UDEL.EDU.

List Maintainer: James Galvin <galvin@UDEL.EDU>

TCP-IP@NIC.DDN.MIL

The NIC has taken over the responsibility for the periodic update of the TCP-IP implementations (the latest update can be obtained via FTP by ANONYMOUS login from SRI-NIC file NETINFO:VENDORS-GUIDE.DOC). We are particularly interested in the addition and expansion of TCP services. In addition to this function, it is hoped that this distribution list can aid in the following areas:
To act as an on-line exchange among TCP developers and maintainers, and to announce new and expanded services in a timely manner.

Archives are kept on SRI-NIC in files:
TS:<TCP-IP>TCP-IP.*
where the "*" is a wild-card character.

All requests to be added to or deleted from this list, problems, questions, etc., should be sent to TCP-IP-REQUEST@NIC.DDN.MIL. Please do not send such requests to TCP-IP@NIC.DDN.MIL, as this address is self forwarding to the entire list membership.

Coordinator: Vivian Neou <Vivian@NIC.DDN.MIL>

There are also two people who should realistically fall into the category of "Internet Resources", at least where KA9Q is concerned. They are:

Phil Karn, <karn@ka9q.bellcore.edu>, and

Bob Hoffman, <hoffman@vax.cs.pittsburgh.edu>

Phil is the primary author of the KA9Q package, without whom this research would have been, at the very best,

difficult. He is the ultimate repository of KA9Q knowledge, and, thankfully, he answers his e-mail promptly. Any significant bugs in or major difficulties with KA9Q should be forwarded to him.

Bob is the official coordinator of Unix ports of the KA9Q code. If the prototype gateway design is migrated to a Unix-based workstation, contact with him will be very beneficial.

Finally, it should be related where the various software mentioned in this thesis is available. The latest (and ever-changing) version of nos KA9Q is available via anonymous FTP to flash.bellcore.com in the /pub/ka9q directory. The source code is in src.arc, the executable is in net.exe, the relevant documentation is in userman.arc and nosdoc.txt, and the BM Mailer (a SMTP mailer system for use with KA9Q) is in bm_src.arc and bmdist.arc.

The NCSA Telnet code can be FTP'd anonymously from omnigate.clarkson.edu in the /pub/ncsa2.2tn subdirectory. The latest collection of public domain packet drivers, all of which should be usable with KA9Q, is available via anonymous FTP from the /pub/ka9q directory of sun.soe.clarkson.edu. There are two files of interest, drivers.arc (just the executables) and driverss.arc

(executables + source code).

B.2 Overview of the NOS KA9Q Software Architecture

Conventionally, protocols have been implemented as processes. This approach returns good protocol modularity, but it results in high system overhead and poor system responsiveness [44]. We need responsiveness to bursty, real-time events (frame arrivals), and protocol modularity in a gateway. KA9Q uses processes to implement protocol layers, with hardware interrupts to allow priority processing of real-time events (i.e., frame arrivals).

The NOS version of KA9Q implements it's own operating system kernel "on top of" MS-DOS. NOS (Network Operating System) is discussed further in B.1.2.

The version used for this research was modified in that the code used for packet radio interfaces was removed. This was done to reduce the amount of memory necessary to run KA9Q. Even with this, the resulting package contains more than 850 KBytes of source code. This appendix is not intended as a complete architectural overview of this package. It merely tries to indicate some of what was learned in the development of the Sytek driver.

B.2.1 Network Operating System (NOS) Overview

NOS is a very small multi-processing operating system kernel that was designed for use with network software applications in mind. It is extremely simple: it is non-pre-emptive and all processes have equal priority. It does, however, allow for dynamic process creation and termination (i.e., processes can fork off other processes). It also allows shared memory structures and provides a crude rendezvous mechanism, but no explicit inter-process communication primitives.

The basic NOS kernel user routines are in the "kernel.c" file. Each are outlined below:

mainproc (name)
    name = pointer to a character string

Create a process descriptor for the main function. Must be actually called from the main function! Returns a pointer to the process' control block.

newproc (name,stksize,pc,iarg,parg1,parg2)
    name = pointer to a character string
    stksize = size of new process' stack
    pc = Initial execution address (pointer to function)
    iarg = Integer argument to pc (argc)
    parg1 = Generic pointer argument #1 (argv)
    parg2 = Generic pointer argument #2 (session ptr)

Create a new, ready process and return pointer to description. The general registers are not initialized, but optional args are pushed on the stack so they can be seen

by a C function. Returns a pointer to the process' control block.

killproc (pp)
    pp = pointer to the process' control block.

Free resources allocated to specified process. If a process wants to kill itself, the reaper is called to do the dirty work. This avoids some messy situations that would otherwise occur, like freeing your own stack. No return value.

killself ()
    no parameters

Terminate current process by sending a request to the killer process. Automatically called when a process function returns. Does not return. No return value.

suspend (pp)
    pp = pointer to the process' control block.

Inhibit a process from running. No return value.

resume (pp)
    pp = pointer to the process' control block.

Restart suspended process. No return value.

alert (pp,val)
    pp = pointer to the process' control block.
    val = integer

Wakeup waiting process, regardless of event it's waiting for. The process will see a return value of "val" from its pwait() call. No return value.

pwait (event)

event = Generic pointer for rendezvous location

Post a wait on a specified event and give up the CPU until it happens. The null event is special: it means "I don't want to block on an event, but let somebody else run for a while". It can also mean that the present process is terminating; in this case the wait never returns. Pwait() returns 0 if the event was signaled; otherwise it returns the arg in an alert() call. Pwait must not be called from interrupt level. Returns an int.

psignal (event,n)
event = Generic pointer for rendezvous location
n = integer

Make ready the first 'n' processes waiting for a given event. The ready processes will see a return value of 0 from pwait(). Note that they don't actually get control until we explicitly give up the CPU ourselves through a pwait(). Psignal may be called from interrupt level. It returns the number of processes that were woken up. Returns an int.

chname (pp,newname)
pp = pointer to the process' control block.
newname = pointer to a character string

Rename a process. No return value.

B.2.2 KA9Q Memory Management

One of the most notable peculiarities of KA9Q is that it defines it's own memory management subsystem (see "alloc.c" for the code). These memory allocation routines are adapted from the malloc and free routines in Kernighan and Ritchie, with memory statistics and interrupt protection added for use with net package. These routines are functionally equivalent to the standard Turbo-C library routines, but must be used instead because the latter check for stack/heap collisions. This causes erroneous failures because NOS process stacks are allocated off the heap.

This can lead to some rather odd behavior, since malloc's allocate memory from a NOS-declared heap (who's size in KBytes is configured by the value of HEAPSIZE in "config.h"). Special care should be taken to insure that you have a sufficiently large heap. During this research, the heapsize had to be increase to 300K, from the 200K which the original net used, to accommodate the memory allocated by the Sytek driver. If you configure HEAPSIZE too large, KA9Q will simply "grab what it can" at startup, but this will leave no memory for other applications (i.e., a DOS subshell).

B.2.3 Significant KA9Q Data Structures

The KA9Q package was written to be highly reconfigurable, depending upon an individual application's needs. A slew of option flags are in the "config.h" file. There are parameters there to allow or inhibit the inclusion of chunks of code (e.g., TCP servers, packet tracing code, RIP, individual drivers) in net during compilation. You can also set such parameters as the memory subsystem heapsize (c.f. B.2.2), the number of available TCP sockets, the number of interactive clients and others. A KA9Q developer should familiarize himself/herself with these parameters.

The various modules of KA9Q pass data in chained structures called mbufs, with the following format:

```
/* Basic message buffer structure */
struct mbuf {
     struct mbuf *next;
     struct mbuf *anext;
     int16 size;
     int refcnt;
     struct mbuf *dup;
     char *data;
     int16 cnt;
};
```

"next" links mbufs belonging to the a single packet, "anext" links packets on queues, "size" is the size of associated data buffer, "refcnt" is the reference count for this mbuf, "dup" is a pointer to duplication of this mbuf if any, "data" points to the actual data buffer for this mbuf, and "cnt" is the number of bytes in this buffer.

Although somewhat cumbersome to work with, mbufs make it possible to avoid memory-to-memory copies that limit performance. For example, when user data is transmitted it must first traverse several protocol layers before reaching the transmitter hardware. With mbufs, each layer adds its protocol header by allocating an mbuf and linking it to the head of the mbuf "chain" given it by the higher layer, thus avoiding several copy operations. When the linked list reaches the device driver, the driver can copy it into contiguous storage for transmission.

A number of primitives operating on mbufs are available in "mbuf.c". The user may create, fill, empty and free mbufs himself with the alloc_mbuf and free_mbuf primitives, or at the cost of a single memory-to-memory copy he/she may use the more convenient qdata() and dqdata() primitives.

TCP and IP require timers. A timer package is included, in "timer.c" and "timer.h". Future developers should familiarize themselves with the "timer" structure in timer.h, given here:

```
struct timer {
     struct timer *next;
     struct timer *prev;
     int32 start;
     int32 count;
     void (*func) __ARGS((void *));
     void *arg;
     char state;
#define   TIMER_STOP     0
```

```
#define   TIMER_RUN      1
#define   TIMER_EXPIRE   2
};
```

"next" and "prev" are pointers for a Doubly-linked-list used to store the timers, "start" is the period of this counter (load value), "count" ticks to go until expiration of this timer, "func" is the function to call at expiration of this timer, "arg" is the argument to pass to this function, "state" is the timer's current state (stopped, running or expired).

There are a timer primitive routines available to the user in "timer.c", include those to start, stop, pause and read timers. KA9Q developers (particularly those working at the transport level and below) should be very familiar with this timer package.

The heart of KA9Q packet routing is the "network" process, whose source code is the function "network" in "config.c". This process handles all packets that pass through the system via a packet queue called "Hopper". "Hopper" is nothing more than a pointer to an mbuf. If a network interface receives a packet, it pre-pends an internal packet header to it and enqueues it in the Hopper. The internal packet header is defined by "phdr" structure in "iface.h". It identifies the interface this packet

arrived on.

The network process infinitely loops, processing any packets that appear in the Hopper Queue. It is responsible for passing them to the appropriate network interface send and receive routines. For more information about this architecture, aspiring developers should peruse the code.

LIST OF REFERENCES

[1] 3Com, "EtherLink II Adapter Guide", 3Com Corporation, 1989.

[2] Bauerfeld, W., "A Tutorial on Network Gateways and Internetworking of LANs and WANs", Computer Networks and ISDN Systems, Volume 13, 1987.

[3] Borland, "Turbo C User's Guide - Version 2.0", Borland International Inc., 1988.

[4] Borland, "Turbo C Reference Guide - Version 2.0", Borland International Inc., 1988.

[5] Borland, "Turbo Debugger User's Guide - Version 2.0", Borland International Inc., 1988.

[6] Chay, J.K, Seltzer, J., and Siddique, N., "A Simple Gateway for ODD Networks", Computer Design (USA), Vol. 23 #2, February 1984.

[7] Clarkson, "NCSA Telnet for the PC - Version 2.2TN and Version 2.2D", Clarkson University, July 1988.

[8] Comer, D., "Internetworking with TCP/IP", Prentice Hall, 1988.

[9] Frank, D.M., "Transmission of IP Datagrams over NET/ROM Networks", Proceedings from the 7th ARRL Computer Networking Conference, 1988.

[10] Garbee, B., "The KA9Q Internet Software Package - Updated for the 890421.1 Revision", May 8, 1989.

[11] Hedrick, C., "Routing Information Protocol", RFC 1058, DARPA Network Working Group, June 1988.

[12] Hinden, R. and Sheltzer, A., "The DARPA Internet Gateway", RFC 823, Bolt Beranek and Newman Inc., September 1982.

[13] Kernighan, B.W. and Ritchie, D.M., "The C Programming Language Second Edition, Prentice Hall, 1988.

[14] Kirton, P., "EGP Gateway Under Berkeley Unix 4.2", RFC 911, USC/Information Sciences Institute, August 1984.

[15] Martinez, R., et. al., "Interconnection of Sytek LOCALNET 20 Networks Through the Defense Data Network Using IP Gateways", IEEE International Phoenix Conference on Computers and Communications, 1987 Conference Proceedings, 1987.

[16] Martinez, R., "Proposal: Generic Gateway Development for the University of Arizona Campus", Computer Engineering Research Laboratory, University of Arizona, July 1989.

[17] Mills, D.L., "Exterior Gateway Protocol Formal Specification", RFC 904, DARPA Network Working Group, April 1984.

[18] Mogul, J. and Postel, J., "Internet Standard Subnetting Procedure", RFC 950, August 1985.

[19] Mohamed, S.A., "Automated Document Distribution with Signature Release Authority Using AI-Based Workstation and Knowledge Base Servers", Computer Engineering Research Laboratory, University of Arizona, December 1988.

[20] Narten, T., "Internet Routing", Computer Communication Review, September 1989.

[21] Nelson, R., "User Documentation for the Packet Driver Collection", Clarkson University, August 1989.

[22] Padlipsky, M.A., "Gateways, Architectures, and Heffalumps", RFC 874, The MITRE Corporation, September 1982.

[23] Perkins, D., "The Point-to-Point Protocol: A Proposal for Multi-Protocol Transmission of Datagrams Over Point-to-Point Links", RFC 1134, November 1989.

[24] Plummer, D.C., "An Ethernet Address Resolution Protocol", RFC 826, November 1982.

[25] Postel, J., "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[26] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[27] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," RFC 793, USC/Information Sciences Institute, September 1981.

[28] Postel, J., "Multi-LAN Address Resolution", RFC 925, USC/Information Sciences Institute, October 1984.

[29] Postel, J., " Requirements for Internet Gateways", RFC 1009, USC/Information Sciences Institute, June 1987.

[30] Reynolds, J. and Postel, J., "Assigned Numbers", RFC 1010, May 1987.

[31] Romkey, J., "A Non-Standard For Transmission of IP Datagrams Over Serial Lines: SLIP", RFC 1055, June 1988.

[32] Rutgers. "Introduction to the Internet Protocols", Computer Science Facilities Group, Rutgers University, July 1987.

[33] Son, C.W., "Functional Description and Formal Specification of a Generic Gateway", Computer Engineering Research Laboratory, University of Arizona, August 1988.

[34] Stallings, W., "Handbook of Computer Communications Standards - Volume 2: Department of Defense Protocol Standards", Howard W. Sams & Company, 1987.

[35] Stix, G., "Technology 90: data communications", IEEE Spectrum, January 1990

[36] Stuck, B.W., "Gateways: Marketing Trends and Design Considerations," LOCALNET '85 Conference Proceedings, 1985.

[37] Sunshine, C., "Network Interconnection and Gateways", IEEE Journal on Selected Areas in Communications, January 1990.

[38] Sytek, "LocalNet 20, Reference and Installation Guide", June 1 1984.

[39] Tanenbaum, A., "Computer Networks", Second Edition, Prentice Hall, 1988.

[40] Tao, J. and Martinez, R., "Internetworking ISDN with LANs", Ninth Annual International Phoenix Conference on Computers and Communications, 1990 Conference Proceedings, 1990.

[41] Turner, K.J., "Gateways for Networking in the Framework of Open Systems Interconnection", Proceedings of the Seventh International Conference on Computer Communication, The New World of the Information Society, 1985.

[42] Von Taube, E., "Gateways Link Assorted Networks", Computer Design (USA), Vol. 24 #2, February 1985.

[43] Wilcox, R.M. and Martinez, R., " An Interactive PC-Based Network Management and Control Package Using a Database Management System", University of Arizona, December 1988.

[44] Zhang, L., "How to Build a Gateway: C-Gateway, An Example", IEEE International Conference on Computers and Applications (Beijing), 1987.

Notes:

References 7 and 21 can be acquired via FTP access to omnigate.clarkson.edu

Reference 9 can be acquired via anonymous FTP access to louie.udel.edu

Reference 10 can be acquired via anonymous FTP access to flash.bellcore.edu

References 11, 12, 14, 17, 18, and 22-31 can be acquired via anonymous FTP access to sri-nic.arpa

Reference 32 can be acquired via anonymous FTP access to topaz.rutgers.edu