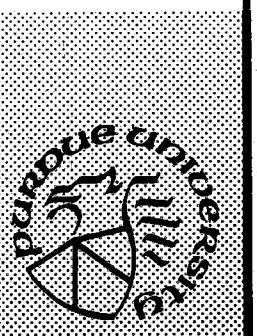
3-1-1989

Adding SLIP Support to UNIX System V

K. B. Sheets
Purdue University



Adding SLIP Support to UNIX* System V

K. B. Sheets

TR-EE 89-20
March, 1989

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

*UNIX is a registered trademark of AT&T.

Adding SLIP Support to UNIX[†] System V

by
Kitrick Sheets

Agriculture Computer Network
Purdue University
West Lafayette, Indiana 47907

TR-EE89-20

March, 1989

[†] UNIX is a registered trademark of AT&T.

ACKNOWLEDGEMENTS

I would like to thank the other members of ACN for their patience and support during the testing of this package.

TABLE OF CONTENTS

CHAPTER

ABSTRACT

SLIP (Serial Line Internet Protocal)[ROMK88][KARE86] is a means by which a serial line may be used as the data link interface for TCP/IP communication [POST80]. This document describes software which can be used to enable a computer running UNIX† System V Release 3 and equipped with the WIN3B [TWG86] software to utilize SLIP. This software is necessary to our organization since we have several UNIX machines for which no ethernet interfaces are available and from which we would like to be able to access the campus network.

† UNIX is a registered trademark of AT&T.

1. Introduction

Serial Line Internet Protocal (SLIP)[ROMK88][KARE86] is a means by which a serial line may be used as a data link interface for TCP/IP communication [POST80]. This document describes software which enables a computer running UNIX† System V Release 3 and equipped with the WIN3B [TWG86] software to utilize SLIP. Since the WIN3B package has no provision for utilizing a serial line as a network interface, software had to be developed which would accomplish this task. This software is necessary to our organization since we utilize several UNIX machines for which no ethernet [METC76] interfaces are available and from which we would like to be able to access the campus network. By connecting these machines via SLIP to a 3B2 with an ethernet interface, we are able to gateway through the 3B2 onto the ethernet and thus have access to the entire campus network.

What follows is a discussion of the SLIP implementation for System V and how it is used in our environment. As background for this information, however, I will first begin with an overview of our network configuration at ACN. I will then discuss briefly the other software packages involved (WIN3B and ka9q). Finally I will describe our SLIP implementation and how it fits together with the other packages to provide cohesive network solution.

2. Agriculture Computer Network Configuration

The Agriculture Computer Network (ACN) consists of several AT&T 3B2/310 computers and one 3B2/500 computer all of which are connected to the campus network via an ethernet interface. In addition to these, we also maintain several AT&T 3b1s and UNIXPCs

† UNIX is a registered trademark of AT&T.

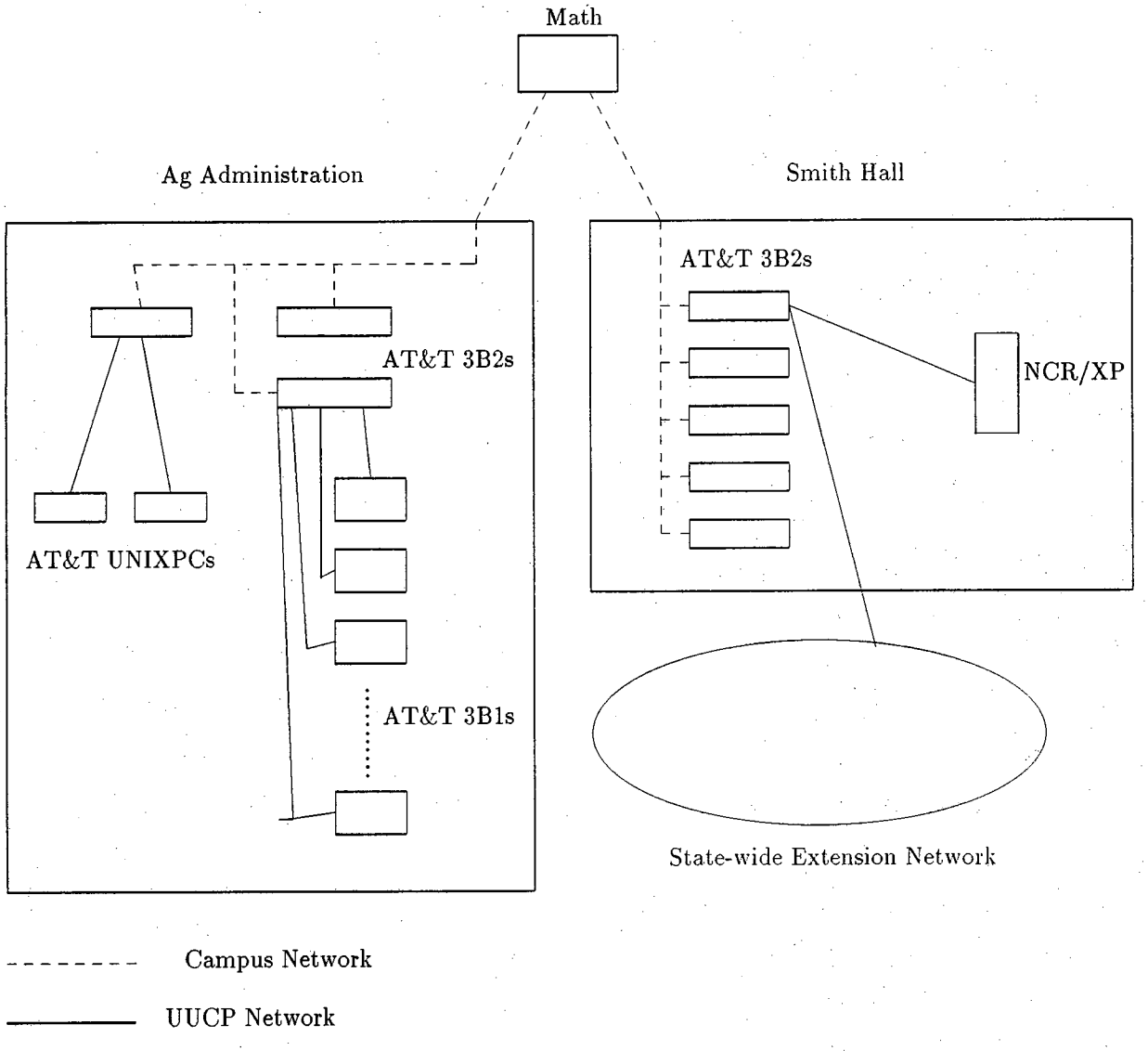
which are used as single user workstations. Finally there is one NCR Tower XP which is used for software development and testing. The 3b1s, UNIXPCs, and the NCR are all connected to the 3B2s via UUCP over RS232 connections.

In addition to supporting these machines on campus, ACN also maintains computers in each of the counties throughout the state which consist of a smattering of the above systems. At the present time all of these systems are connected via UUCP to the 3B2/500.

Although users at remote county sites rarely need access to the campus network at the present time, we see this changing in the near future and plan to expand the software described in this document to support dial-up IP [LANZ89]. However, a more pressing need was to connect the on-campus resources that we currently maintain more closely to the campus network.

3. Software Overview

The software described in this paper interfaces with two other software packages which are currently available to provide TCP/IP support in a System V environment. The first of these which is available for the AT&T 3B2 computer is the WIN3B package from The Wolongong Group. This package provides an implementation TCP/IP over ethernet. It is this package which provides the upper layers of TCP/IP which will be used in our SLIP implementation. The other package which is used is the ka9q internet software package [KARN87] which is in the public domain. For our purposes, this package acts essentially as a user level daemon which communicates using internet frames over a serial link. This package is used on our AT&T 3b1s, AT&T UNIXPCs, the NCR Tower XP to provide telnet and ftp access to/from these systems. This package was written primarily for use on PCs running



Agriculture Computer Network

DOS but with very little effort was ported to our System V UNIX environment.

In order to understand our implementation of SLIP, some knowledge of the structure of WIN3B, System V STREAMS [RITC84], and the UNIX tty subsystem [BACH86] is needed. The following sections briefly describe each of these. For more background on these see the referenced material.

3.1. System V STREAMS

STREAMS was first developed at Bell Labs as an alternative to the character I/O subsystem of UNIX. The principle behind STREAMS is to allow layering of drivers and protocol modules in such a way that a logical software organization can be achieved. Using this model, a user could build a "stream" of successive modules which would each handle a logical chore in the processing of data. TCP/IP is a good candidate for this since it has several layers each of which need to manipulate the incoming and outgoing data. In ISO reference model terms [TANE81], at the bottom of the stream would be the device driver, on top of that would be the transport module, then the network module, etc. Each of these modules would be responsible for manipulating data going both up and down the stream as required for that layer of software. In STREAMS, each of these protocol modules is called a "service provider."

Figure 1 shows a logical representation of the STREAMS mechanism using TCP/IP as an example.

3.2. Structure of WIN3B

The WIN3B package uses this layering mechanism in its TCP/IP implementation. At the bottom of the stream is a network interface driver which controls the ethernet hardware. Above that is an *ARP* module which is responsible for the TCP/IP address resolution protocol. Layered on top of this is an *IP* module and then a *TCP* module. Since STREAMS allows

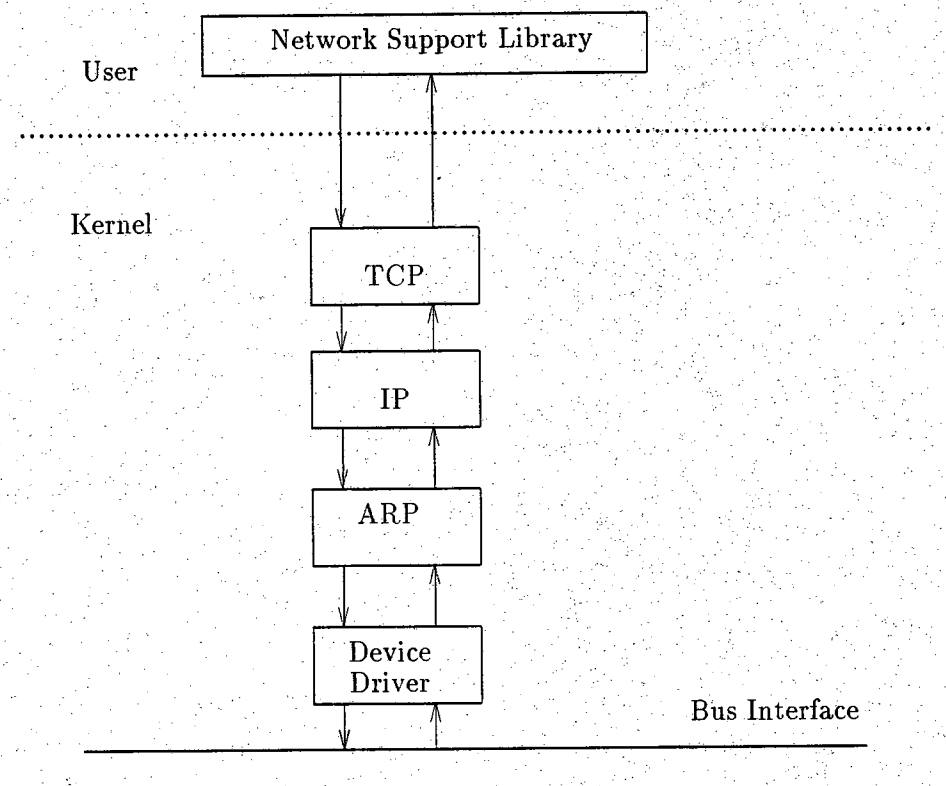


Figure 1. Example Streams Configuration

multiple connections to a module, a gateway can be formed by connecting two network interfaces at the bottom of the stream. It is also possible to connect users directly to the *IP* module instead of the *TCP* module thus allowing UDP and TCP connections in the same configuration.

3.3. Streams != Clists

STREAMS modules pass "messages" up and down the stream in order to send or receive data. At the bottom level, the device driver takes these messages and transmits them onto

the device. It also takes input data and sends it up stream. The problem with this is that serial devices in System V use clists (a linked list of small buffers of characters (cblocks)) to communicate with the device. Therefore, some mechanism must be employed to convert the messages coming down the stream into clists for transmission on the device. The opposite must be done when input is received since all input characters are also stored in clists. Although eventually all System V character device drivers will utilize STREAMS, they all currently use clists. When this transition takes place, the conversion from STREAMS messages to clists will be unnecessary.

4. SLIP Implementation

This SLIP implementation consists of three parts: 1) a STREAMS service provider, 2) a SLIP tty line discipline, and 3) a daemon process which is used to configure the serial lines to be used in SLIP communication. The following sections describe each of these in detail.

4.1. /etc/slipdaemon

In order to give the administrator some control over the configuration of the serial lines used, there is a daemon process which is used for this purpose. Its primary purpose is to set baud rates for the lines which will be used for communication. When the process starts up, it reads a configuration file which lists the lines which are to be used and the baud rate which will be used for communication. It then configures each of the lines at the requested baud rate and hangs thus keeping the configurations static for when the TCP/IP software utilizes the device. An example configuration will be explained in more detail in the next section.

4.2. SLIP Service Provider

The SLIP service provider has three major functions; to associate a stream with a physical channel, to translate STREAMS messages to/from clist format so that they can be transmitted on the device or moved up stream, and to add and remove the SLIP encoding used to mark the end of incoming and outgoing frames.

The association of a stream with a physical tty device is done in the *slipopen()* routine. This open routine is accessed through a device entry which has a major number associated with the SLIP module and which has encoded in the minor number the major/minor pair for the target tty device. For example, if the major number for the SLIP module is 86 and we want to use */dev/tty11* for a SLIP channel, the device entries would look like the following:

```
crw-r--r--  1 root     other     1,  0 Mar 14 10:35 /dev/tty11
crw-r--r--  1 root     other    86, 10 Mar 14 10:34 /dev/slip11
```

In this example, an open of */dev/slip11* would cause an association to be established between the newly created stream and the device */dev/tty11*. This is done by changing the line discipline associated with the tty to the SLIP line discipline (discussed later). Therefore any data received on this stream will be transmitted on the associated device. Also any data received on */dev/tty11* will be sent up the associated stream. This completes the logical link between the stream and the device. This link will remain intact until the stream is closed at which time all output pending to the device is flushed and the line discipline associated with the tty is reset.

Once the association between the stream and the tty is established, data can flow up/down stream as though a normal streams device were attached at the bottom.

As data enters the SLIP service provider from upstream, the message type is checked so that the proper action can be taken. There are three main types of messages which can be received by this module. They are control messages, protocol messages, and data messages. Each of these messages has a different meaning in the STREAMS context and requires a distinct action.

Control messages are used to send control and status information to/from the module. When this type of message is received, it is an indication that the stream has been assembled and is ready for communication. The service provider can simply discard this message since it is essentially just notification that the stream is assembled.

Protocol messages are used by upper levels to find out specific information about a service provider or device driver. It inquires about information such as the minimum and maximum packet sizes accepted by the device, the size of an address, type of network provided (e.g. point-to-point, bus, token ring), the current state of the connection, etc. When a protocol message is received, a reply to the specific information requested is assembled and sent back up stream. This information will be used by the upper layers of software to assemble messages to be sent over this link.

In STREAMS there is the notion of immediate and deferred processing of messages. Important messages such as control and protocol messages require immediate action and are serviced when received. Data messages on the other hand are placed on a queue and serviced at a lower priority. The processing of this data in STREAMS is done before a return to user mode. The kernel will see if there are any queues which need serviced and call the appropriate

service routine for that queue. In STREAMS there is a read side and a write side of a module and as such there is an associated read and write side service routine.

In this implementation the write side service routine *slipwsrv()* is responsible for most of the data manipulation chores. It takes each queued message and performs two actions on it. The first is to encode the incoming data in SLIP format. This involves placing a "frame end" (FREND) character at the end of each message. It must also escape any FRENDs which are contained within the data of the message so that when it is coalesced at the other end the message will be received in one piece. The second job that *slipwsrv()* performs is to take the translated data and place it onto the output clist for the device. This is done by allocating cblocks (small buffers to hold characters) as needed, filling them with the data and linking them onto the clist. This clist is then "drained" at interrupt level by the line discipline as described in the next section.

As part of the SLIP service provider there is also a read side service routine *sliprsrv()*. This routine is responsible for passing the data received by the line discipline up stream to the upper layers of software. It also provides a flow control mechanism by keeping data on its queue until the upper layers of software can handle it.

4.3. SLIP Line Discipline

The SLIP line discipline is the portion of the software which is responsible for transmission and reception of the data to/from the physical line. It is also responsible for decoding incoming data into frames which can be sent up stream. There are two routines which make up the line discipline portion of the module, *slipin() and slipout()*.

The *slipin()* routine (or any line discipline input routine for that matter) is called by the device driver each time an input interrupt is received. It is the job of the line discipline to process this data in some manner expected by the user. In our case this would be docoding the data from SLIP format and reconstructing the frame which was sent from the remote host. Since the data received on an interrupt is in a cblock buffer, it must be copied to a temporary buffer which is used to collect an entire frame to be sent up stream. As soon as the end of the frame is encountered, it is sent up stream and the frame building procedure starts all over. This procedure continues until the device is closed.

The *slipout()* routine is also called at interrupt level. This time, however it is on the receipt of an output interrupt. This signifies that the last cblock which was queued for output has been transmitted and the hardware is ready for another one. Since the output clist has already been built by the *slipwsrv()* routine, all that needs to be done is to take a cblock off this list and queue it for transmission by the hardware. This procedure also continues until the output is exhausted or the device is closed.

Since there is no direct interaction with the user at this level, these two routines are all that are necessary to complete the line discipline.

5. SLIP Configuration

There are several steps which must be performed in order to configure SLIP into a system equipped with the WIN3B software. The first of these is to configure the SLIP service provider/line discipline into the system. This requires that both the /etc/system file and /etc/master.d/kernel files must be updated. The /etc/system file is a list of drivers and modules which are to be configured when the system boots. A line must be added to this file

which tells the system to configure the SLIP module at boot time. This line will look like this:

INCLUDE: SLIP

The next step is to add the SLIP line discipline to the list of available line disciplines in /etc/master.d/kernel. These are kept in the *linesw[]* table which appears as in Figure 2.

By adding slip as a valid line discipline, it is now possible for the driver to access it at interrupt level by indexing through *linesw[]*. As soon as this file is updated we need to run the command **mkboot -k /boot/KERNEL** in order to update the kernel data structures contained in this module so that the change will be present after the next boot cycle.

The next thing that needs to be done is to modify the */usr/etc/inetinit.cf* file which contains the streams configuration which is built for TCP/IP. By adding the following lines:

```
sl0   /dev/slip11   /dev/ip0   2   remote    local
sl1   /dev/slip12   /dev/ip0   2   remote1   local1
```

we are configuring two serial devices to be used for communication.

In this example, the device */dev/tty11* will be used to communicate between hosts **remote** and **local**. According to the WIN3B documentation, **local** and **local1** need to be distinct aliases for the local host if more than one network interface is used. The **2** in the above line specifies to WIN3B that this is a point-to-point network. The **/dev/ip0** in the above line tells WIN3B which module to push on top of SLIP. Finally, the **sl0** is used as an identifier for WIN3B to keep track of distinct streams.

Next we need to create the */etc/slipconfig* file which will give a description of how to configure the SLIP channels we will be using. An example of this file is the following:

```
* Line Discipline Switch Table
*   order: open close read write ioctl rxint txint modemint

linesw (%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l%l
        ={
*       tty -------------
        &ttopen,
        &ttclose,
        &ttread,
        &ttwrite,
        &ttioctl,
        &ttin,
        &ttout,
        &nulldev,
*       xt -------------
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &xtin,
        &xtout,
        &nulldev,
*       sxt -------------
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &sxtin,
        &sxtout,
        &nulldev,
*       slip -------------
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &nulldev,
        &slipin,
        &slipout,
        &nulldev,
        }
linecnt (%i) ={4}
```

Figure 2. Example *linesw[]* table from /etc/master.d/kernel.

```
/dev/tty11    9600
/dev/tty12    19200
```

At the present time, this file contains only the device and baud rate. However, it will be expanded in the future to include another field which specifies whether the link is permanent or temporary (i.e. if it is a hard link or if it is to be used for dial-up IP).

Finally, we need to add a script in */etc/rc2.d* which will be used to start the *slipdaemon* used to configure the serial lines which will be used. An example of this script is shown in figure 3.

This script will only start a daemon if there is not one currently running.

```
PROC=slipdaemon

pid=`ps -ef | grep "${PROC}" | sed -e "/grep/d" | sed 's/[ ]*[^ ]*[ ]* ]*).*/1/'` ;

case "$1" in
        start)
                if [ ! "${pid}" ]
                then
                        /etc/slipdaemon > /dev/console 2>&1 &
                fi
                ;;
        stop)
                if [ "${pid}" ]
                then
                        kill ${pid}
                fi
esac
```

Figure 3. Example startup script for */etc/slipdaemon.*

Now that we have all of this done, we are ready to reboot the system. We need to make sure that */etc/system* is booted since the kernel has been updated as well the addition of SLIP to the desired configuration. This can be done by either executing *shutdown -i6* or by executing *shutdown -i5* and booting /etc/system manually from firmware mode. Once the system is rebooted, we should now be able to communicate over the newly configured serial link.

6. Closing Remarks

6.1. Future Directions

Work is currently under way to expand this software to support dial-up IP. This will allow us to pull the rest of our systems (i.e. those which reside in counties throughout the state) into the campus network.

6.2. Conclusion

By implementing SLIP on the AT&T 3B2 systems, we were able to organize our on-campus resources in a much more cohesive fashion. We are now able to communicate with any host on the campus network from our workstations without having to use multiple communication protocols thus simplifying data transmission greatly.

REFERENCES

[BACH86]
Bach, M. J., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[KARE86]
Karels, M. J., "Changes to the Kernel in 4.3BSD," *Unix System Manager's Manual (SMM)*, April, 1986.

[KARN87]
Karn, P., "The ka9q Internet Software Package"

[LANZ89]
Lanzillo, L., Partridge, C., "Implementation of Dial-up IP for UNIX Systems," *Proceedings of the 1989 Winter USENIX Technical Conference*, San Diego, CA, Jan. 1989, pp. 201-207.

[METC76]
Metcalfe, R. and Boggs, D., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM,* 19(7), 395-404, July 1976.

[POST80]
Postel, J. (ed.), "DOD Standard Transmission Control Protocol," *ACM Computer Communication Review,* Vol. 10, No. 4, Oct. 1980, pp. 52-132.

[RITC84]
Ritchie, D. M., "A Stream Input Output System," *AT&T Bell Laboratories Technical Journal,* Oct. 1984, Vol 63, No. 8, Part 2, pp. 1897-1910.

[ROMK88]
Romkey, J., *A Nonstandard for Transmission of IP Datagrams Over Serial Lines: Slip; RFC 1055,* Internet Working Group, Requests for Comments, No. 1055, DDN Network Information Center (NIC) at SRI International, Menlo Park, California, June 1988.

[TANE81]
Tanenbaum, A., *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[TWG86]
The Wolongong Group, Inc., *Enhanced TCP/IP WIN/3B Administrator Guide for the AT&T 3B2 Computers*, 1987.