```
                7920 pts/17   0:01 BTPengin
```

5. Tear Down

To tear down and kill all the processes, first quit Jammer but giving it the ‘quit’ command. Then perform the following commands relative to the ps(1) listing above:

```
> kill 7918 # kill fakeCM, which in turn kills GBNSC and some other support processes

> kill 7896# kill NodeSim, which in turn kills atm_card_sim and other support processes

> kill 7885# kill gbnOverseer, which in turn kills some other support processes
```

After killing the processes listed above, it is a good idea to do a ps(1) to check to see that there are not any stray processes, like BTPengine, left running.

6. References

[1] Turner, J.S., “Fast Packet Switching System,” U.S. Patent 4 494 230 , January 15, 1985.

[2] Turner, J.S., “Design of a Broadcast Packet Switching Network,” in IEEE Transactions on Communications, 36 (6): 734-743, June 1988.

[3] Turner, J.S., ARL Staff, ANG Staff, “A Gigabit Local ATM Testbed for Multimedia Applications,” ARL Working Note-94-11, January 1995.

[4] Beal, O. Matthew., “Jammer Language Description: A Script Language for GigaBit Switch Testing,” ARL Working Note 96-01.

[5] DeHart, John D., “Connection Management Software System (CMSS) Architecture,” ARL Working Note 95-03, July 1996.

[6] Wu. D. and DeHart J., “GBNSC: The GigaBit Network Switch Controller,” Washington University, Applied Research Laboratory Working Note ARL-94-12, Version 1.2, July 1996.

4.1.3 etc/peer_file

This file describes what other hosts are attached to the switch being simulated so the NodeSim process can set up the necessary structures to communicate with those hosts. In this environment, the peer_file will contain the name of the name of the host that is running the control software.

Here is a sample file (this file has only one line):

```
"ackbar" 1 0 1 0
```

4.1.4 etc/gbn-overseer-*

This file is generated by the gbnOverseer process and stores information to be used by other processes. Do not remove or modify his file.

4.1.5 switch-*.*

This file is generated by the GBNSC and stores information about the switch to be used by other processes. Do not remove or modify this file.

4.2. Process Execution

To operate or test a GBN switch there are five processes that need to be started from the command line. These five process will start others. Here are the commands that can be used to start everything.

```
> cd /project/gbn_sw/switch/bin

> ./gbnOverseer &

> ./NodeSim -f ../etc/configNS -p ../etc/peer_file -s 1 4 &

> ./fakeCM ../etc/configSC 1 &

> ./Jammer 0.1 4
```

Here is a view obtained from ps(1) of the relevant processes that have been started when these commands are run:

```
PID TTY      TIME COMD
7897 pts/17   0:01 BTPengin
7924 pts/17   0:01 BTPengin
7916 ?        0:01 BTPengin
7915 ?        0:00 atm_card
7919 pts/17   0:02 GBNSC
7885 pts/17   0:00 gbnOvers
7923 pts/17   0:01 Jammer
7918 pts/17   0:00 fakeCM
7896 pts/17   0:00 NodeSim
```

```
            VXT_size        1024;
        }
        LINK {  }
        CP_port : SW 1, 0;
}
~
```

4.1.2 etc/configSC

This files specifies the configuration of a GBN switch. It is passed by fakeCM to the GBNSC so it can set up its internal structures for controlling its switch, simulated or real.

Here is a sample file:

```
SWITCH 1 GBN            --
{ -- Switch hardware section
        BTPPORT 3550            -- use this port for BTP communications
        PORTS 8                 -- 8 ports, numbers 0 to 7
        CHIPS 1 2               -- Default chips, IPP and OPP
                IPP CHIP 3      -- Overrides for specific ports: chip number,
                        6       -- then ports
         END
        PARAMS
            VPT 1                           -- This VPI is VP-terminated
            CONFIGURATION TIMEOUT 60        -- 60 seconds to configure
            POLLING TIMEOUT 60              -- 60 seconds between polling
        END
        CONTROL
            0 0             -- CP is connected to these two ports
            0/32            -- control VXI (cells from CP)
            0/32            -- control return VXI (cells arriving at switch)
        END
        LINKS
        -- Port Direction Type Speed   Resys-bw       Client
            0     <=>     UNI  @155   @2200         "r2d2.arl.wustl.edu"   1
            1     <=>     UNI  @155   @2200         "owen.arl.wustl.edu"  1
            2     <=>     NNI  @620   @1780    0 5  “jabba.arl.wustl.edu”
            3     <=>     NNI  @620   @1780
            4     <=>     NNI  @620   @1780
            5     <=>     NNI  @620   @1780
            6     <=>     NNI  @620   @1780
            7     <=>     NNI @2400   @0
        END
        INIT
            IPP 0 VPCOUNT 255           -- Use this value for VPCount in IPP 0
            IPP 1 VPCOUNT 127
            IPP 2 VPCOUNT 255
            IPP 3 VPCOUNT 255
            IPP 4 VPCOUNT 255
            IPP 5 VPCOUNT 255
            IPP 6 VPCOUNT 255
            IPP 7 VPCOUNT 255
        END
        PRESET
        --              Input                   Output
        --      Port    VPI     VCI     Port    VPI     VCI
                 0       1       34      4       0       32
                  4       0         0     32      1       34
        END
}
```

3.4. BTPengine

When a process creates a BTPengine object, a second process named BTPengine is created. This process opens a UDP port on its host and listens for messages on that port. A communications channel between the creator and BTPengine processes is established. The creator process can then use the BTPengine object to send messages to, and receive messages from, the BTPengine process. Messages that are sent contain a destination address which consists of an Internet host address and port number. The BTPengine process attempts to send the message to another BTPengine process on that host and port, using a connectionless but reliable protocol called BTP. Any message which the BTPengine process receives is forwarded to the creator process.

3.5. TIMER

Timers are implemented using a shared-memory segment containing the current time. A dedicated process updates the shared-memory segment, which other processes can read. The advantage of this is that the usual mechanisms for getting the time (*e.g.*, gettimeofday) force a context switch. By using the shared memory, the process is able to obtain the time without context switching.

3.6. atm_card_sim

The GBN switch will be controlled over an ATM interface from a workstation. The interface to the ATM link may be hardware-dependent. The atm_card_sim process provides a uniform API for this link. This process currently communicates over ethernet with NodeSim. When an ATM link is later used to communicate with the GBN switch, the interface will be changed.

3.7. fakeCM

In the standard software architecture for doing dynamic network control, a Connection Manager (CM) would control the operation of an ATM node made up of several switches. In the test architecture we have provided a “fakeCM” to fill this role so that other software processes do not have to be modified to perform in both environments.

4. Start Up

4.1. Configuration Files

4.1.1 etc/configNS

This file gives the node configuration for NodeSim to be able to set up its simulated node.

Here is a sample file:

```
node 0 {
    SW 1 {
        SW_size 8;
        Control_port    0;
        Input_buf_size 128;
        Recycling_size  32;
```

1. Introduction

This document describes the installation, start up and tear down of the GigaBit Network (GBN) software. This software has been designed and is being implemented to control and test the prototype GBN switch[3] being designed and built at Washington University in St. Louis. At the time of this version of this document, the software is still under development, and so this document is subject to change as the software matures.

2. Installation

The software should be installed in a set of directories located in /project/gbn_sw/switch. The initial implementation, relies on this directory path. It is planned that future versions will have this constraint removed. Below is a Unix ls(1) listing of the software to be installed.

```
-r-xr-x---   1   jdd   gbn   2862600  Mar 10 1996    /project/gbn_sw/switch/bin/GBNSC*
-r-xr-xr-x   1   jdd   gbn   2547016  Mar 10 1996    /project/gbn_sw/switch/bin/Jammer*
-r-xr-x---   1   jdd   gbn   599404   Mar 10 1996    /project/gbn_sw/switch/bin/NodeSim*
-r-xr-xr-x   1   jdd   gbn   76264    Mar 10 1996    /project/gbn_sw/switch/bin/TIMER*
-r-xr-x---   1   jdd   gbn   508028   Mar 10 1996    /project/gbn_sw/switch/bin/atm_card_sim*
-r-xr-x---   1   jdd   gbn   1061204  Mar 10 1996    /project/gbn_sw/switch/bin/fakeCM*
-r-xr-x---   1   jdd   gbn   106896   Mar 10 1996    /project/gbn_sw/switch/bin/gbnOverseer*

-rw-rw----   1   jdd   gbn   50       Mar 10 1996    /project/gbn_sw/switch/etc/atmcard-ackbar
-rw-rw----   1   jdd   gbn   2389     Mar 14 1996    /project/gbn_sw/switch/etc/configSC
-rw-rw----   1   jdd   gbn   146      Mar 10 1996    /project/gbn_sw/switch/etc/configNS
-rw-rw----   1   jdd   gbn   12       Mar 10 1996    /project/gbn_sw/switch/etc/gbn-overseer-ackbar
-rw-rw----   1   jdd   gbn   17       Mar 10 1996    /project/gbn_sw/switch/etc/peer_file
-rw-rw----   1   jdd   gbn   125      Mar 10 1996    /project/gbn_sw/switch/etc/switch-0.1
```

3. Software Overview

There are several processes involved in the GBN software [5]. They will each be briefly described in this section. The manner in which each is executed will be detailed in the next section.

3.1. GBNSC

This is the GBN Switch Controller [6]. This process understands all the hardware details of the switch and provides the mechanisms for updating and controlling the switch.

3.2. NodeSim

This is a software simulator which is able to functionally simulate the control operations performed by the GBN switch. It actually can simulate several GBN switches operating as a node, but initially the primary concern is to simulate one switch in order to test the other control software. Using Nodesim, software engineers and test engineers are able to prepare software and tests so that when the hardware switch is assembled they can very quickly be ready to test the actual hardware.

3.3. Jammer

Jammer [4] is an interactive script language processor designed to enable testing of the prototype ATM switches designed at Washington University. Jammer was originally created for testing of the first Washington University prototype ATM switch [2] and has been updated for testing of the new GBN switch. To provide the test engineer with as much flexibility and testing power as possible, Jammer enables full access to all tables and registers within the switch. It also provides basic programming constructs to allow iterative and conditional testing.