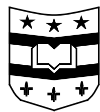
Using Jammer with the Washington University Gigabit Switch

Presented by:

John DeHart
jdd@arl.wustl.edu
http://www.arl.wustl.edu/~jdd
http://www.arl.wustl.edu/arl

Applied Research Laboratory

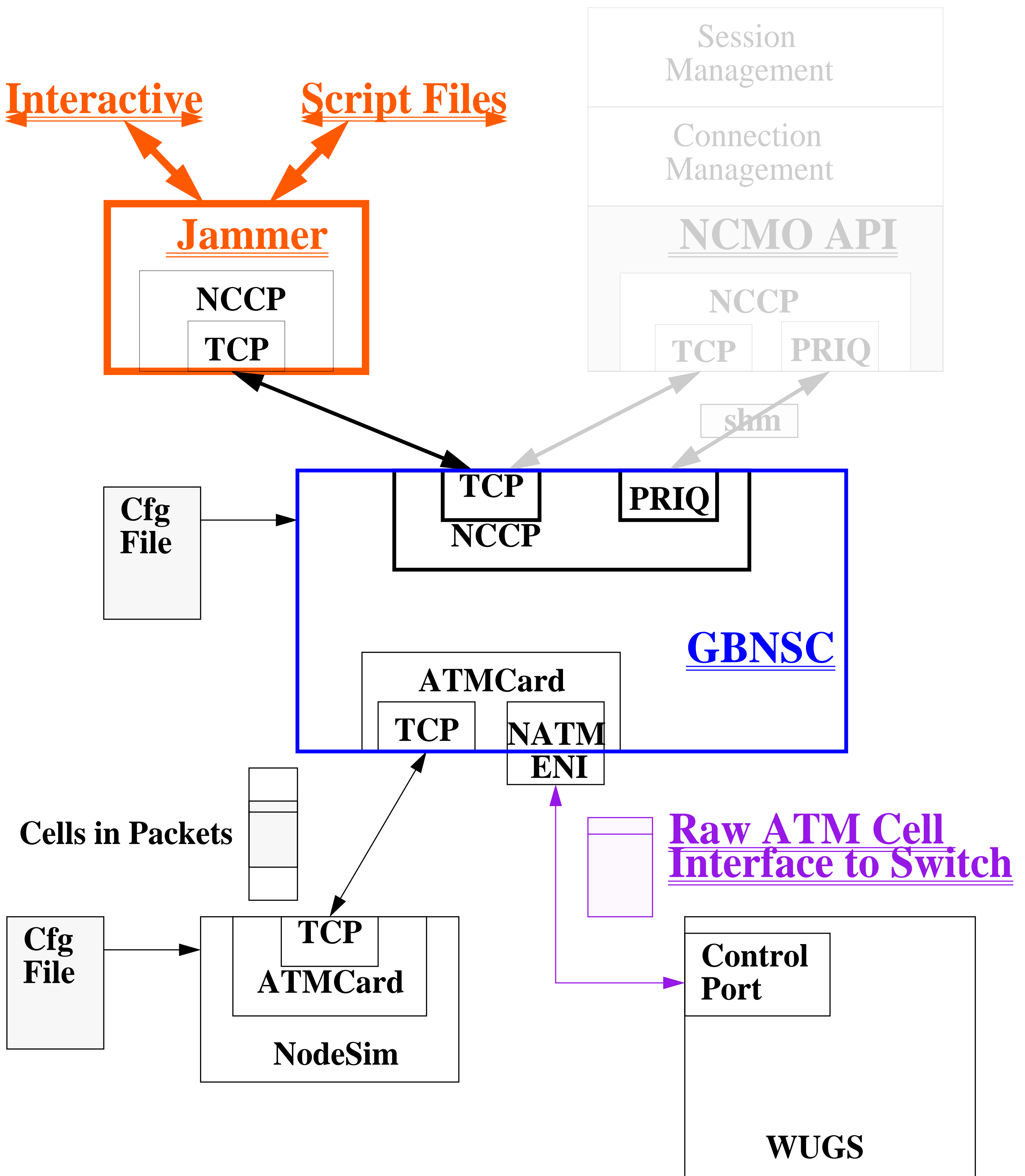
WUGS Kits Program
Washington University
July 13-24, 1998
August 3-14, 1998



Washington

WASHINGTON • UNIVERSITY • IN • ST • LOUIS

WUGS Software Overview



Jammer: Purpose

- **Complete access to all the bits in switch tables and registers.**
  - **some parts are very switch specific.**
- **No range checking**
  - **allows for testing error conditions in switch.**
- **Interactive and Batch oriented input.**
- **Programming constructs**
  - **support iterative and repeated tests.**

Jammer: Command Line Arguments

options:

- -[hH] get the usage message
- -[sS] small option, i.e. do no buffering at all
- -f buffer in a temporary file as opposed to memory
- -D set Debug_ON level. larger value means more debug output
- -d set JAM_base_debug level. larger value means less debug output
- -[gG] forces core dump for SIGBUS and SIGSEGV.
- -T Set Maximum number of outstanding transactions before Jammer will pause to allow the switch controller to catch up
- -t Hysterisis setting for maximum number of outstanding transactions
- switch_id two digit (0.1) id of the switch controller to communicate with
- switch_host hostname on which the switch controller is running.
- switch_port TCP port on which the switch controller is listening for connections

Typical Startup:

> Jammer 0.1 derlin 3550

Switch Manipulating Commands

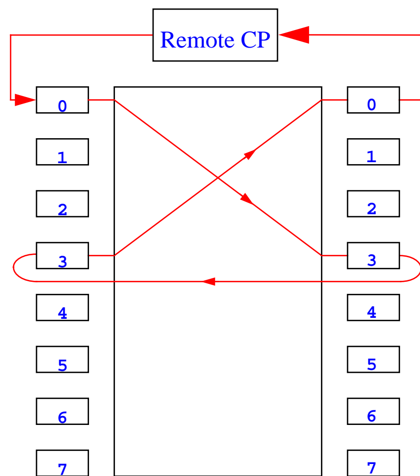
Command	Description
reset sm	reset the switch
write mr **write vcxt** **write vpxt**	Commands for writing entries into maintenance registers, virtual circuit tables and virtual path tables.
read vcxt **read vpxt** **read mr**	Commands for reading entries from maintenance registers, virtual circuit tables and virtual path tables.
clear errors	Clear all the error bits in all the maintenance registers
clear vpxt **clear vcxt**	clear a particular vcxt or vpxt entry.
clear vxt	clear ALL vcxt and vpxt entries at a port
build **merge** **receive** **recycle** **xmit**	A group of commands that make it easier (debatable) to build multipoint to multipoint recycling connection trees

Session Related Commands

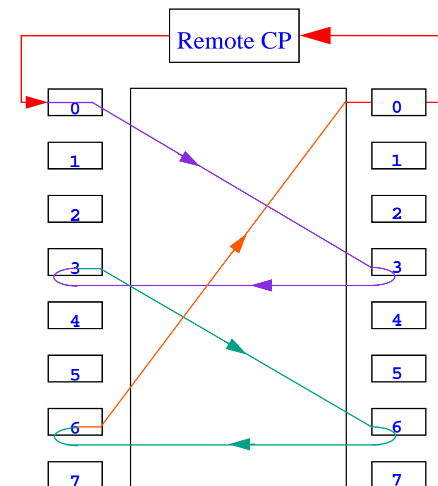
Command	Description
# comments	embed comments in scripts
help	print a terse list of available commands
quit	exit Jammer
shutdown	kill GBNSC that Jammer is connected to
echo "STRING"	print out a string included values of variables
pause N	pause for N seconds
upause N	pause for N microseconds
set print	set how verbose output should be
wait	wait until all outstanding operations are done or timed out
waiting	list all outstanding operations
prompt "STRING"	print out a string and wait for <CR> from user
<var> = prompt "STRING"	print out a string and wait for "value <CR> " from user, assign value to variable
date	print the current date and time (timestamp script output)
cd	change the current working directory
set switch	change the switch to which Jammer is communicating
<CTRL-C>	interrupt a Jammer script
!<system-command>	execute a Unix shell command

Testing Related Commands

Command	Description
test cell0	Send a "ping" cell through the switch (same as ping sm)
test cell1	Send a test cell with one recycling pass through the switch.
test cell2	Send a test cell with two recycling passes through the switch.
get cells	Receives cells from the switch and verifies specified payload
put cells	Injects cells into the switch with a specified payload
ping sm	Pings the switch to make sure it is alive
ping sc	Pings the switch controller to make sure it is alive
failed	keeps track of whether any commands have failed. can be set, tested and cleared



One Recycling Pass



Two Recycling Passes

Programming Structures

Command	Description
if-else-fi	Conditional block
proc-end	Procedure block
while-done	Loop
break	break out of a loop (currently unimplemented)
return	return from a procedure (no return value!)
include	include a file
int var	declare an integer variable
int var[range]	declare an integer array variable
int var[]	declare an integer array procedure parameter
$var	evaluate an integer variable
$var[index]	evaluate an integer array variable
<expressions>	+, -, *, /, ++, --, <, >, <=, >=, ==, !=, !, ( ) are all available for building expressions.

Jammer Example 1: Setting up Audio/Video Connections

```
# using dual SUNI OC-3 cards on ports 2 and 4
# Audio (vci 100), Video(vci 101) from port 2A to port 4B

                  #         C C                     V   V   B V V B A A
                  #         Y Y   U U   V R         P   C   D P C D D D
                  # B R     C C C D D S P C B       I   I   I I I I R R
                  # I C D 1 2 S 1 2 C T O R         1   1   1 2 2 2 1 2
                  # ------------------------------------------------------
write vcxt 2 100    1 2 1 0 0 1 0 0 0 0 0 0       128 100   0 0 0 0 4 0
write vcxt 2 101    1 2 1 0 0 1 0 0 0 0 0 0       128 101   0 0 0 0 4 0
write vpxt 2  0     1 0 1 0 0 0 0 0 0 1 0 0         0   0   0 0 0 0 0 0

                #     T R   R   V
                #     S C   C   P       S   H      S
                # T   O B   B   C   S R S   R      C
                # G   F D   H   N R L L L   E      L
                # I   F T   D   T E E E E   T      T
                # ------------------------------------------
 write mr   2 2   0 128 32  0 255 1 1 1 1 100000   0
```

Jammer Example 2: VCXT Test

```
proc vcxt_test(int switch_size, int vcxt_size, int abort_flag)

    set print quiet
    return_value = 0  # This is a Global variable

    if ($switch_size > 8)
        echo "vcxt_test: Illegal switch size -- $switch_size (must be <= 8)."
        return_value = 1
        return
    fi

    if ($vcxt_size > 1024)
        echo "vcxt_test: Illegal vcxt size -- $vcxt_size (must be <= 1024)."
        return_value = 1
        return
    fi


    echo "vcxt_test: Testing VCXT tables."

    int num_failures
    int entry
    int link
    num_failures = 0
    entry = 0
```

Jammer Example 2: VCXT Test (continued)

```
while ($entry < $vcxt_size)
   link = 0
   while ($link < $switch_size)
                                  #          C C                     V    V    B    V      V    B    A A
                                  #          Y Y    U U    V R       P    C    D    P      C    D    D D
                                  # B R      C C C  D D S  P C B     I    I    I    I      I    I    R R
                                  # I   C D  1 2 S  1 2 C  T O R     1    1    1    2      2    2    1 2
                                  #-------------------------------------------------------------------
      write vcxt $link $entry  1 1 1 0 0 0 0 0 0 1 0 0 0   0     0   0   0     0   0 0
      write vcxt $link $entry  1 2 1 0 0 0 0 0 0 1 0 0 0   0     0   85  21845 85  5 0
      write vcxt $link $entry  1 3 1 0 0 0 0 0 0 1 0 0 170 43690 170 17  43690 170 3 7
      write vcxt $link $entry  1 7 1 0 0 0 0 0 0 1 0 0 255 65535 255 255 65535 255 4 6

      link++
   done
   wait
```

Jammer Example 2: VCXT Test (continued)

```
        if (failed)
            echo "vcxt_test: write to VCXT index $entry FAILED:"
            clear failed
            num_failures++
            number_of_errors++
            return_value = 1
            if ($abort_flag != 0)
                return
            fi
        fi
        entry++
        if ($entry % 100 == 0)
            echo "vcxt_test: Tested VCXT table entries 0 - $entry at all links."
        fi
    done
    if ($num_failures == 0)
        echo "vcxt_test: All VCXT tables PASSED."
    else
        echo "vcxt_test: One or more VCXT tables FAILED."
        return_value = 1
    fi

    set print normal
end
```

Jammer Example 3: Data Cells

```
# If there are any failures due to bit errors, the Switch Controller will print out that it received
# a cell that did not match any of the payload patterns it was expecting. So keep an eye on the
# GBNSC output!!
proc test_cells (int inPort, int outPort, int inVCI, int outVCI, int cnt, int cyc, int pattern)
    int port
    port = 0
    int timeout
    timeout = 10
    while($port <= 7)
       write mr   $inPort   2 0 128 32 0 255 1 1 1 1 100000 0
       write mr   $outPort  2 0 128 32 0 255 1 1 1 1 100000 0
       write mr   $port     2 0 128 32 0 255 1 1 1 1 100000 0

                                   #       C C                     V    V       B    V    V B    A    A
                                   #       Y Y   U U   V R         P    C       D    P    C D    D    D
                                   # B R   C C C D D S P C B       I    I       I    I    I I    R    R
                                   # I  C D 1  2  S 1  2  C T O R  1    1       1    2    2  2   1    2
                                   #------------------------------------------------------------------------
       write vcxt $port $outVCI     1  2  1  0  0  1  0  0  0  0  0  0  128  $outVCI 0   0    0  0  $outPort0
       write vpxt $port      0      1  0  1  0  0  0  0  0  0  1  0  0    0    0     0   0    0  0    0    0

       write vcxt $inPort $inVCI 1 2 1 $cyc 0 1 0 0 0 0 0 0  0    outVCI    0   0    0   0  $port 0
       write vpxt $inPort 128 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
```

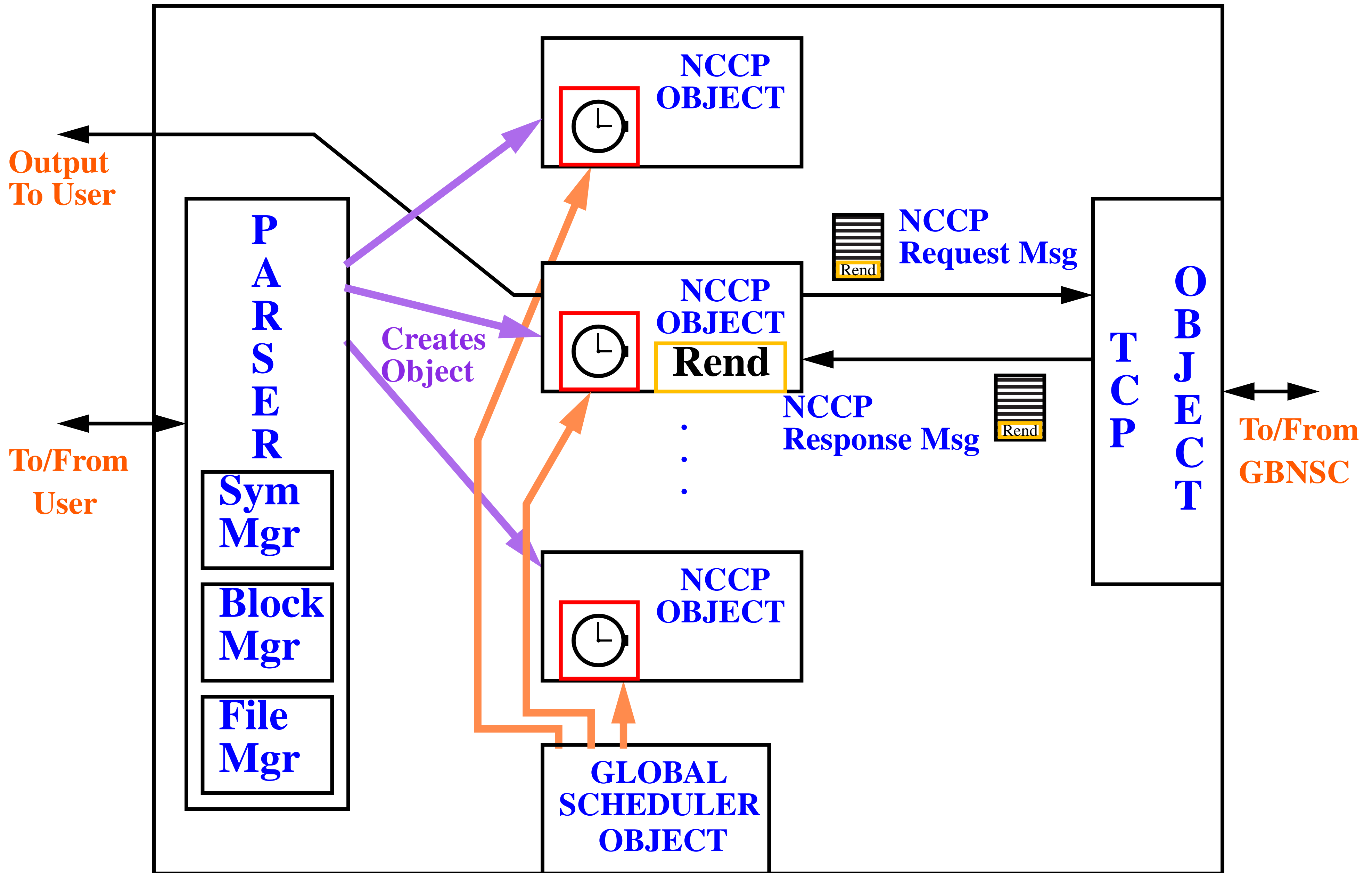
Jammer Example 3: Data Cells (continued)

```
        echo " Doing Port $port"

        get cells  0  $output_VCI    $cnt   $timeout  48   0  $pattern
        put cells  0  $input_VCI     $cnt             48   0  $pattern

        wait

        port = $port + 1
    done

    echo "Done"
end
```

Jammer Example 4: An Interactive Session

```
> ./Jammer 0.1 derlin 5550

Enter command:  ping sc
Ping Operation Completed Successfully

Enter command:  ping sm
Ping Operation Completed Successfully

Enter command:  write vcxt 2 100
      BI( 0(Discard) 1(Propagate))[0]: 1
      RC( 0(specific path) 1(Port2) 2(Port1) 3 (Both) 7 (Be-
tween)[1]: 2
      D( 0(ControlCell) 1(DataCell))[0]: 1
      CYC1(0,1): 0
      CYC2(0,1): 0
      CS(0,1): 0
      UD1(0,1): 0
      UD2(0,1): 0
      SC(0,1): 0
      VPT(0,1): 0
      RCO(0,1): 0
      BR(0,1): 0
      MAPT1VPI(0~255): 0
      MAPT1VCI(0 ~ (2^16-1)): 100
      BDI1(0 ~ 255)): 0
      MAPT2VPI(0~255): 0
      MAPT2VCI(0 ~ (2^16-1)): 0
      BDI2(0 ~ 255)): 0
      ADR1(0 ~ 16)): 4
      ADR2(0 ~ 16)): 0

Write VCXT Operation Completed Successfully

Enter command:  read vcxt 2 100
Read VCXT Operation Completed Successfully
      bi = 1
      rc = 2
      d = 1
      cyc1 = 0
      cyc2 = 0
      cs = 0
      ud1 = 0
      ud2 = 0
      sc = 0
      vpt = 0
      rco = 0
      br = 0
      mapt1vpi = 0
      mapt1vci = 100
      bdi1 = 0
      mapt2vpi = 0
      mapt2vci = 0
      bdi2 = 0
      adr1 = 4
      adr2 = 0

Enter command:
```

Internal Structure



To Add a New Command to Jammer

- Add to parser
  - add any need lexical symbols to jammer.l
  - add any grammar rules to jammer.y
- Add classes to NCCP for new command (assuming it is to send an operation to GBNSC)
  - find a similar existing operation
  - grep for that operation name in all the NCCP*.[cH] files
  - copy all the files that are specific to they operation to create files for new operation
  - modify copied files for new operation
  - modify general files (e.g. NCCP_OperationTypes.h) to include new operation
- Add code to GBNSC to handle new operation
  - use a similar copy/modify strategy...

Jammer Usage Hints

- Use at least THREE windows:
    - GBNSC
    - Jammer
    - Your favorite editor. Use this for saving commands. Then you can cut and paste.
- Useful counters in VCXTCC and MRs 4 and 15
- Error bits and counters in 3, 4, 5, 14 and 16