Overview of the Control Software for the Washington University Gigabit Switch

Presented by:

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

Applied Research Laboratory
http://www.arl.wustl.edu/arl/

WUGS Kits Program
http://www.arl.wustl.edu/~jst/gigatech/kits.html

Washington University
July 13-24, 1998
August 3-14, 1998

Washington
WASHINGTON • UNIVERSITY • IN • ST • LOUIS

Washington University Gigabit Switch Software Interfaces

Software completely separate from Switch.

No processing engine on switch.

Switch Control Interface is through ATM Cells.

Same Interface can control software switch simulator.

Low Level Script Interface for Programmable Test Scripts.

Object Oriented API used by Connection Management.

GSMP Interface present but not fully functional and tested.

Object Oriented API used by Clients for UNI Signaling Protocol Interface.

Object Oriented API used by Connection Management for NNI Signaling Protocol Interface.

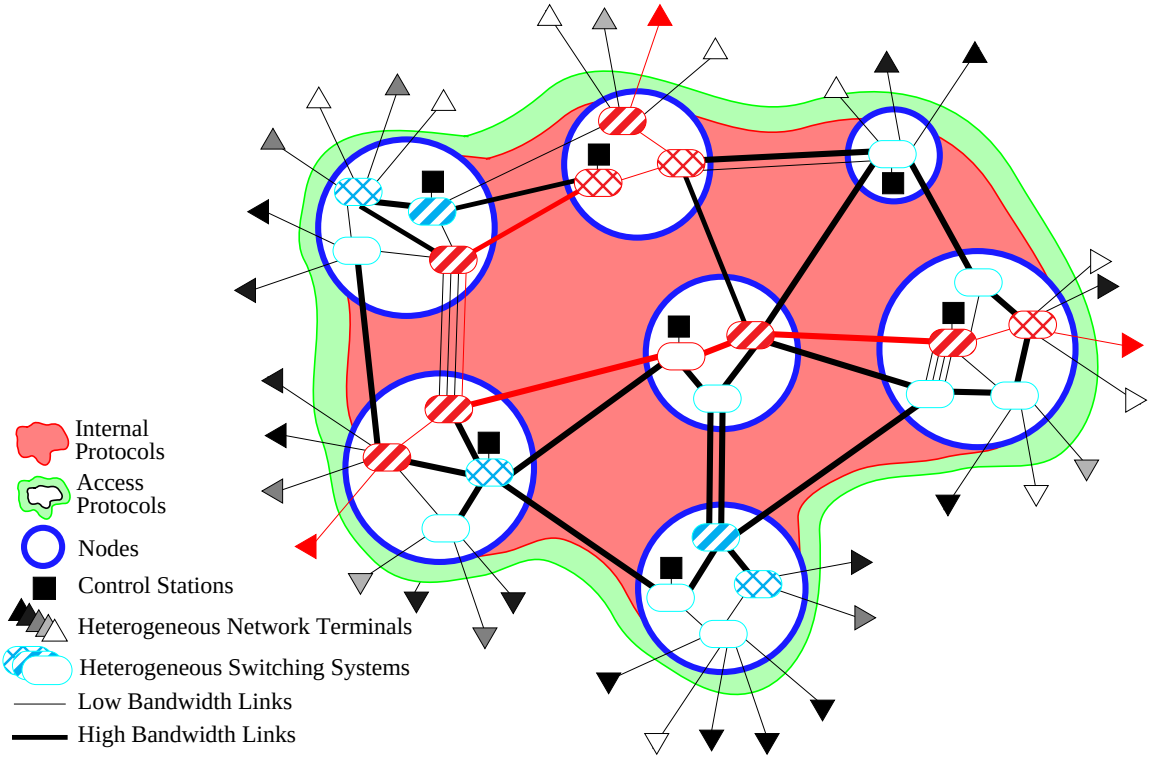
Software Status

- **Functional Today**
- **Simplified over the past year. (Removed about 25K lines of code)**
- **Lines of Code: Total = 135,100 lines of C++ (not counting comments)**
  - GBNSC: 21000
  - Jammer: 15300
  - NodeSim: 2800
  - Libraries: 46000.
  - CMAP: 25000
  - CM: 25000
- **Platforms:**
  - NetBSD - uses Efficient Networks Inc. adapter and WU driver (preferred!)
  - Solaris - uses Fore Systems adapter and driver (not well tested)
  - Others: ???

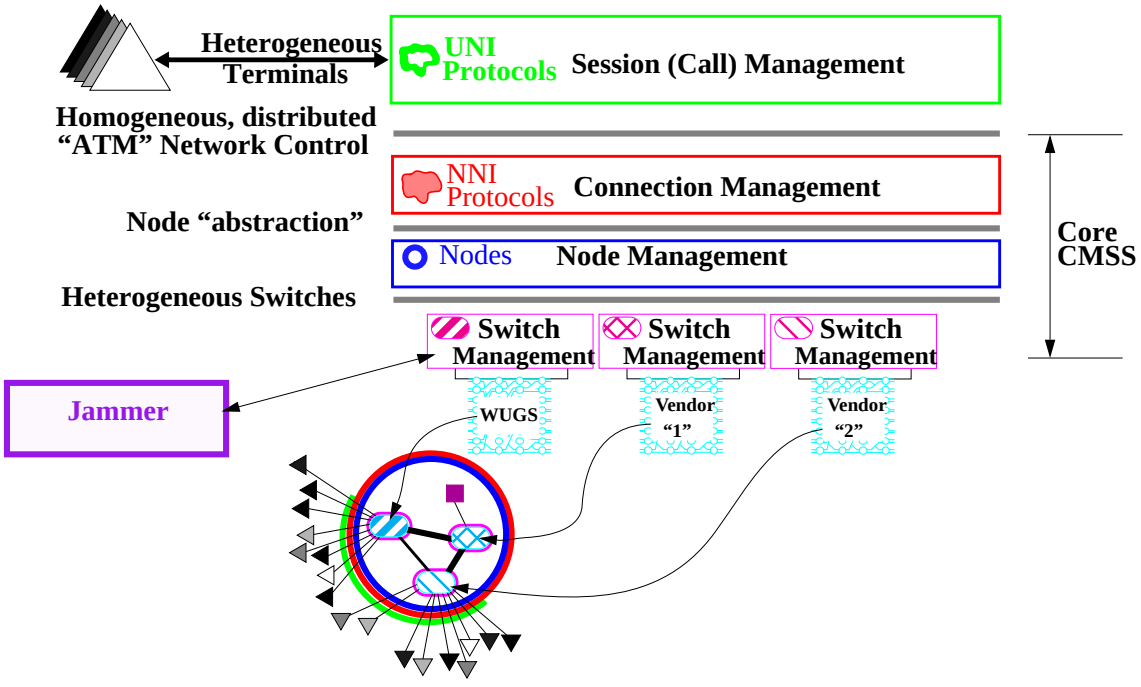
History

- **Started in 1989**
- **Research Associates who have contributed:**
  - Rick Bubenik (1990-1993)
  - Ken Cox (1993/4-1994/5)
  - John DeHart (1989-1998)
  - Mike Gaddis (1989-1993)
  - Dakang Wu (1993-1998)
  -
- **Graduate Students who have contributed:**
  - Pete Flugstad
  - Ian Flanigan
  - Matt Beal
  -
- **Industry/University visitors who have contributed:**
  - Sam Choi (Goldstar)
  - Fatoh Yap (UMKC)
  -

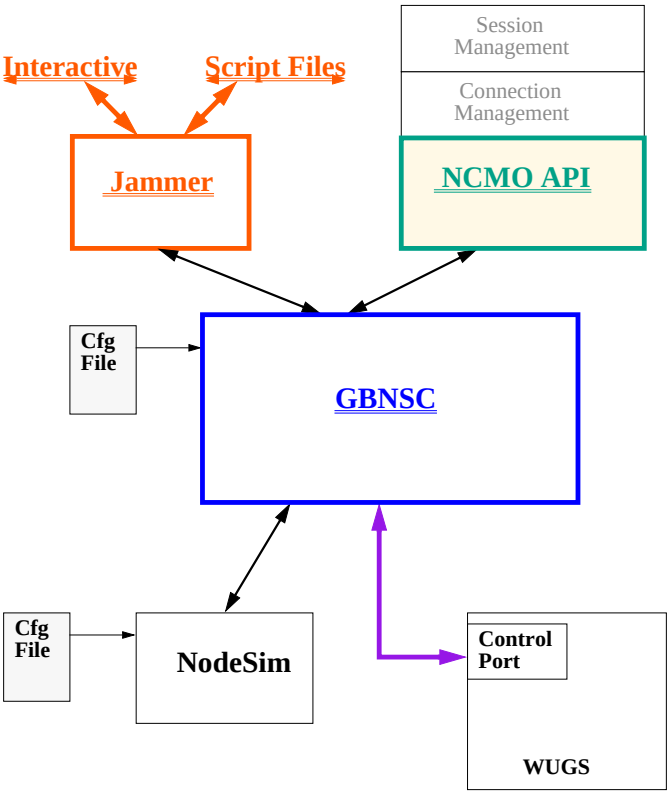
Subdividing the Control Problem



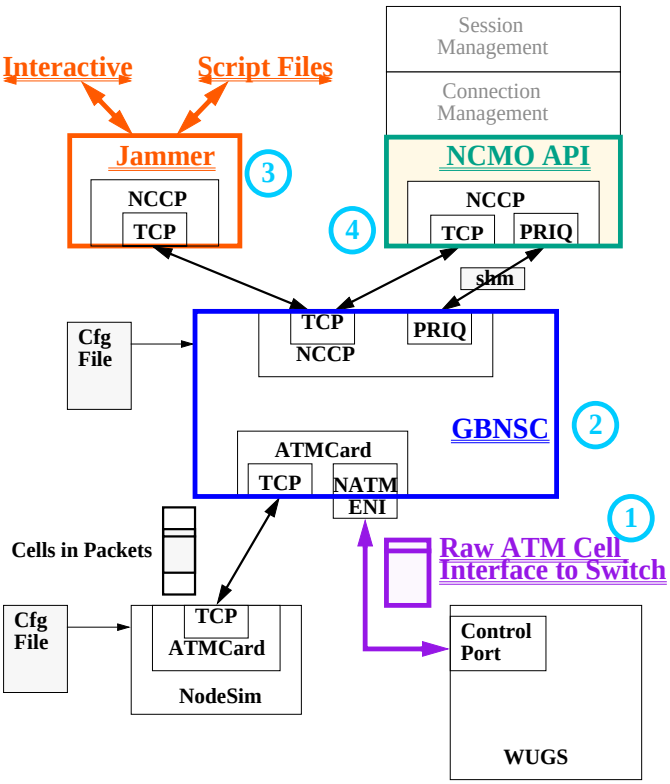
Subdividing the Control Problem



WUGS Software Overview



WUGS Software Overview



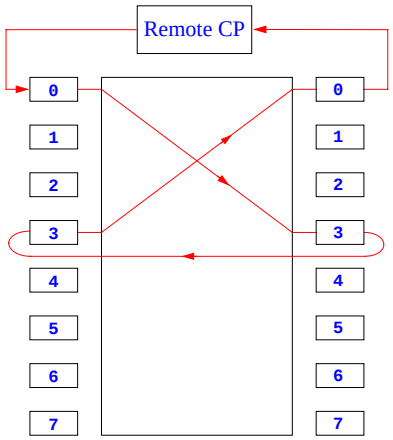
WUGS Hardware Control Interface

Access to Switch is through ATM Cells

Control Cell Formats are defined for updating and testing the switch

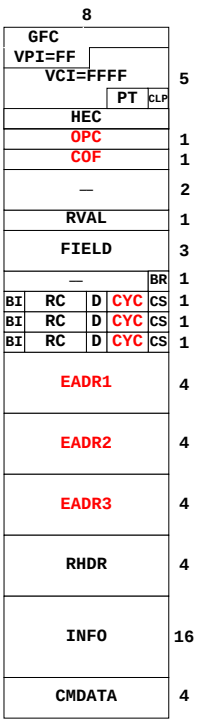
Control Cells sent on VPI=0 VCI=32

Physical jumper on Port indicates whether it is valid for receiving Control Cells



Control Cell Fields

External Control Cell
CP to Switch

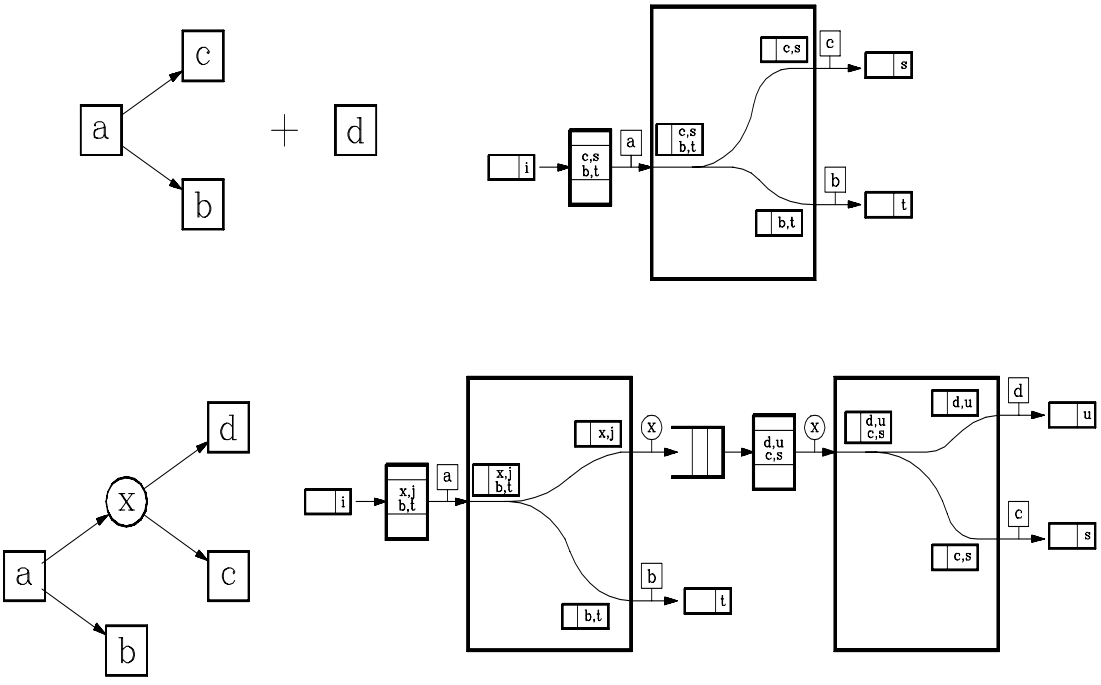


Field	Description
ATM Header	Standard ATM Header Fields
OPC	**Opcode**
COF	**Control Offset - count for which switch chip is to perform control operation.**
RVAL	Return Value - code describing success or failure of operation.
FIELD	Target table entry
BR	Bypass Resenquencer - for immediate release of cells at output port
BI	Busy/Idle - distinguish between idle and control or data cells
RC	Routing Control - specific path, Port1, Port2, both or range copies.
D	Data Cell - distinguish between control and data cells
CYC	**Recycling**
CS	Continuous Stream
EADRi	**External Routing Addresses**
RHDR	Return Header - for routing control cell back to CP
INFO	Information read or to be written
CMDATA	Connection Management Data - used by CP to identify returning control cells.
LT	Local Time - local switch cell clock. Can be used by CP to compute traffic statistics. (Return cells only)

Control Cell Opcodes

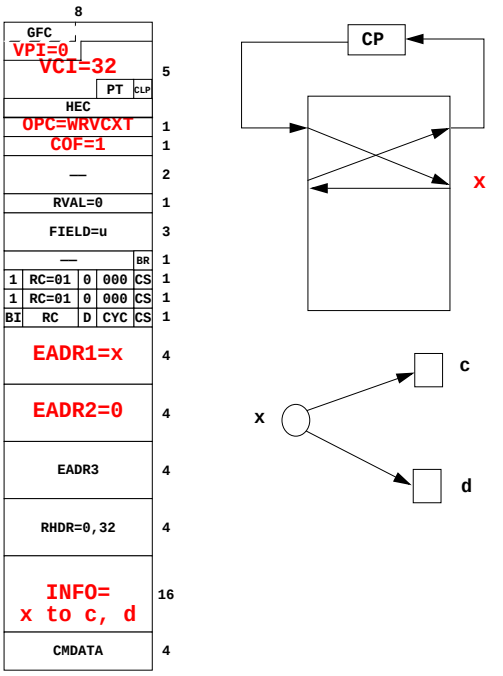
Opcode	Command	Description
0	**NOP**	**No operation (used for cells that test switch operation and internal paths)**
F0 (hex)	**RST**	**Hard reset of all chips. The opcode is F0 instead of 1 so that a single bit error in any other control cell does not transform it into a RST.**
2	CLRERR	Clear all error flags in all chips.
3	RDVPXT	Read virtual path table entry from VXT (everything except CC field)
4	RDVCXT	Read virtual circuit table entry from VXT (everything except CC field)
5	RDVPXTCC	Read cell counter (CC) from virtual path table.
6	RDVCXTCC	Read cell counter (CC) from virtual circuit table.
7	WRVPXT	Write virtual path table entry into VXT (does not write CC field)
8	**WRVCXT**	**Write virtual circuit table entry into VXT (does not write CC field)**
9	WRVPXTTR	Write virtual path table entry into VXT and start transitional time stamping
10	**WRVCXT TR**	**Write virtual circuit table entry into VXT and start transitional time stamping**
11	WRVCXTCC	Write cell counter (CC) to virtual path table (for testing only)
12	WRVCXTCC	Write cell counter (CC) to virtual circuit table (for testing only)
13	ERRORS	Return a cell only if error conditions exist
14	RDMR	Read maintenance register field
15	**WRMR**	**Write maintenance register field**

Adding An EndPoint to a Connection

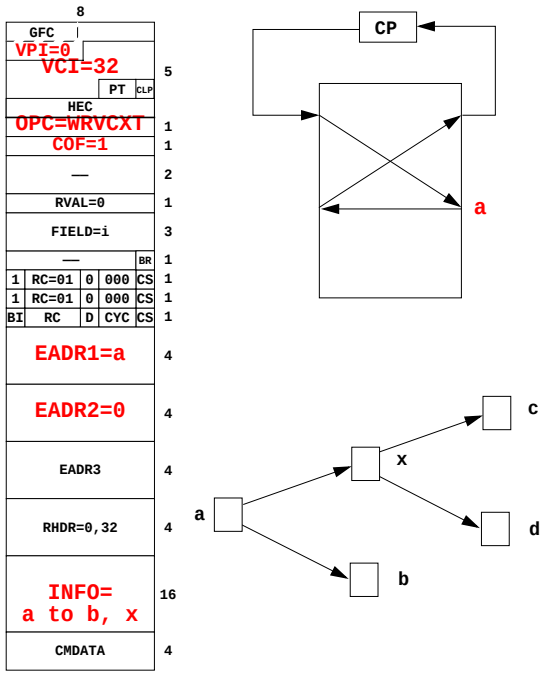


Control Cells That Add *d* into Existing Connection

Control Cell to Port x



Control Cell to Port a



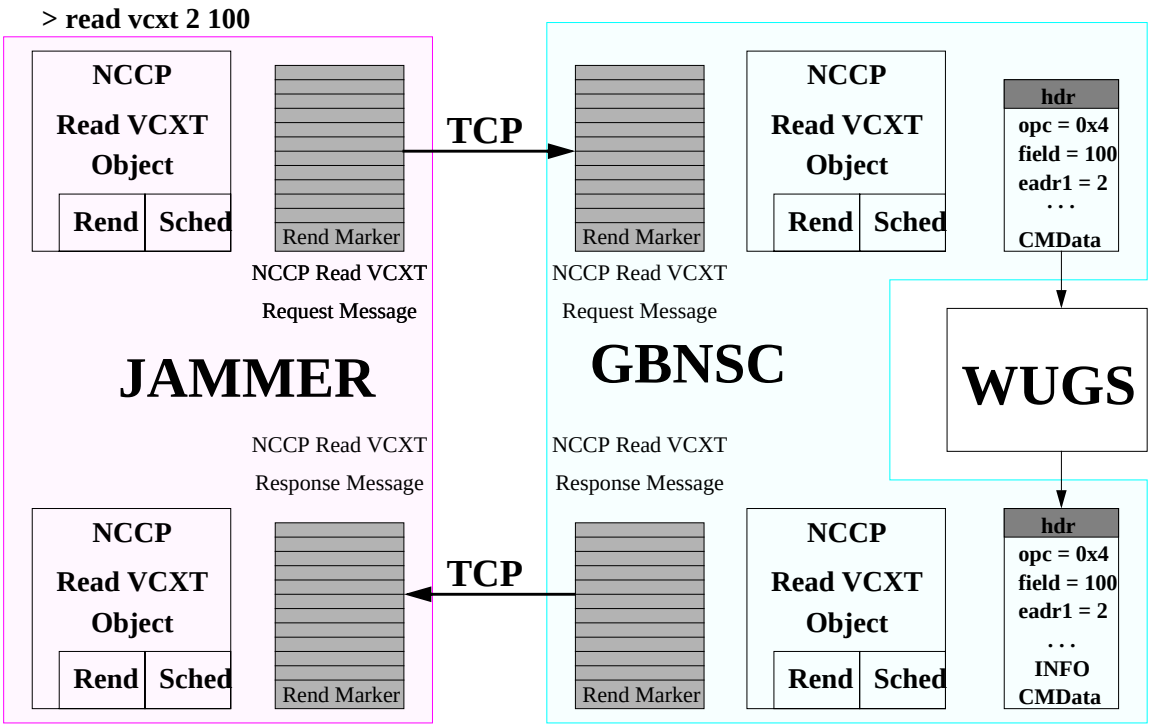
Reading a VCXT Entry

Enter command: read vcxt 2 100
Read VCXT Operation Completed Successfully

bi = 1
rc = 3
d = 1
cyc1 = 1
cyc2 = 0
cs = 1
ud1 = 0
ud2 = 0
sc = 0
vpt = 0
rco = 0
br = 0
mapt1vpi = 0
mapt1vci = 100
bdi1 = 0
mapt2vpi = 0
mapt2vci = 101
bdi2 = 0
adr1 = 4
adr2 = 5

Enter command:

Reading a VCXT Entry



Code Layout

wugs/
wugs/bin
wugs/bin/NetBSD
wugs/bin/solaris
wugs/etc
wugs/include
wugs/include/NCMO
wugs/include/NCCP
wugs/lib
wugs/lib/NetBSD
wugs/lib/solaris

wugs/src
wugs/src/CM
wugs/src/CMAP
wugs/src/GBNSC
wugs/src/JAMMER
wugs/src/NodeSim

wugs/src/common_code
wugs/src/common_code/ATMCARD
wugs/src/common_code/BANDWIDTH
wugs/src/common_code/BYTEBUFFER
wugs/src/common_code/COMMON_UTILITIES
wugs/src/common_code/CONTEXT_SWITCHER
wugs/src/common_code/ERROR_HANDLER
wugs/src/common_code/GBNCAC
wugs/src/common_code/NCCP
wugs/src/common_code/NCMO
wugs/src/common_code/OBJECT_JOIN
wugs/src/common_code/OBJECT_RENDEZVOUS
wugs/src/common_code/PRIORITY_QUEUE
wugs/src/common_code/PRIQUE_PAIR
wugs/src/common_code/SCCOMMON
wugs/src/common_code/SCHEDULER
wugs/src/common_code/SEMAPHORES
wugs/src/common_code/SIGNAL_HANDLER
wugs/src/common_code/TCPOBJECT
wugs/src/common_code/VXIMANAGER

Libraries

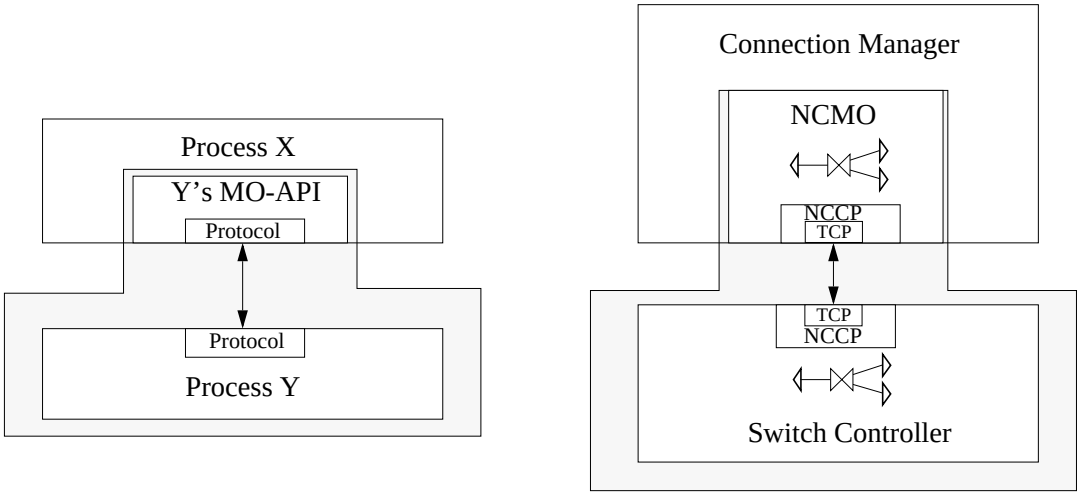
ATMCARD	: API for ENI and Fore driver interfaces. Provides access to NATM cell interface to switch, reserving and releasing VPI/VCIs.
BANDWIDTH	: Simple bandwidth object with peak, average and burst.
BYTEBUFFER	: Byte array with shift operators used for formatting protocol messages an encapsulating machine byte order dependencies.
COMMON_UTILITIES	: Miscellaneous common utilities including ones relating to paths, sorting, shared memory access and time.
CONTEXT_SWITCHER	: Originally developed for SunOS 4.x when context switching was not terribly efficient. Now, under Solaris and NetBSD those problems do not seem to be as bad. Still used for its general interface to select().
ERROR_HANDLER	: Common interface to print functions in the hope of provide more uniform output formats.
GBNCAC	: Call Admission Control. Not fully integrated into CMSS.
NCCP	: Node Control Communication Protocol. Definitions and interface routines for messaging protocol used between CM/NC/SC.
NCMO	: Node Control Managed Object. API for building and manipulating general multipoint connection objects. Provides the interface between CM/NC/SC. Utilizes the NCCP for communicating between the associated processes.

Libraries (continued)

OBJECT_JOIN	: Base classes for building and manipulating parent/child and sibling relationships between objects.
OBJECT_RENDEZVOUS	: Base class for providing a reliable, robust rendezvous mechanism for objects which perform or wait for asynchronous operations.
PRIORITY_QUEUE	: Interprocess communication mechanism implemented as a shared memory queue (unidirectional). Provides several variations including out of band for high priority messages. (Being phased out.)
PRIQUE_PAIR	: Priority_Queues are typically used in pairs. This provides and interface for allocating them as a pair.
SCCOMMON	: Some common type definitions and classes relating to the WUGS SC.
SCHEDULER	: Global object and base class for object scheduling. This allows an object performing or waiting on an asynchronous event or operation to schedule a time-out for a certain time in the future.
SEMAPHORES	: Wrapper for semaphore related system calls.
SIGNAL_HANDLER	: Wrapper for signal related system calls
TCPOBJECT	: Wrapper for TCP socket system calls.
VXIMANAGER	: Object interface for managing VPIs and VCIs for a switch port.

Managed Object - API Model

- **Model used for object interface between CM/NC/SC processes.**
- **API in Process X provides interface and access to objects to be managed in Process Y**
- **API and accompanying library provide object interface, protocol and means to initiate process Y.**



Makefiles and Building

- **global Makefile and make.defs.* files in wugs/src**
- **local Makefile in each source directory**
- **all Makefiles provide make {'all', 'clean', 'install', 'depend'...}**
- **all Makefiles have similar structure:**

```
#
CMS = ..
include $(CMS)/make.defs.common

OBJECTS=  $(OSTYPE)/GBNSC_ReadErrors.o $(OSTYPE)/GBNSC_ClearErrors.o \
        $(OSTYPE)/GBNSC_WriteVXT.o $(OSTYPE)/GBNSC_CellManager.o \
        $(OSTYPE)/GBNSC_SwitchController.o \

LOCAL_LIBS =
LOCAL_EXEC = $(OSTYPE)/GBNSC   $(OSTYPE)/GSMP_Client   $(OSTYPE)/buildHardwareInfo

#DEBUG =
#DEBUG = -g
#DEBUG = -O4
#DEBUG = -g -DTIMING
DEBUG = -g  -D_ERR_DEBUG
#DEBUG = -D_ERR_DEBUG


$(OSTYPE)/%.o: %.C
        $(CC) -c $(DEBUG) -I. -I$(OSTYPE) $(CMINC) -DYY_SKIP_YYWRAP -o $@ $<

$(OSTYPE)/%.o: $(OSTYPE)/%.C
        $(CC) -c $(DEBUG) -I. -I$(OSTYPE) $(CMINC) -DYY_SKIP_YYWRAP -o $@ $<
```

all: $(LOCAL_EXEC)

clean:
	rm -f $(LOCAL_EXEC) $(OBJECTS) $(DEBUG_OBJECTS) $(OSTYPE)/GBNSC_grammar.C $(OSTYPE)/GBNSC_grammar.h $(OSTYPE)/GBNSC_lex.C $(OSTYPE)/buildHardwareInfo $(OSTYPE)/buildHardwareInfo.o $(OSTYPE)/GSMP_Client.o $(OSTYPE)/GSMP_CLib.o

install: all
	cp install.$(OSTYPE) install.temp
	chmod a+rwx install.temp
	install.temp
	rm install.temp

$(OSTYPE)/GBNSC_grammar.o $(OSTYPE)/GBNSC_lex.o: $(OSTYPE)/GBNSC_grammar.h GBNSC_SwitchController.h GBNSC_RecyclingManager.h GBNSC_Link.h

$(OSTYPE)/GBNSC_lex.C: GBNSC_lex.l
	lex GBNSC_lex.l
	@sed -f stupidDuplicateDefines < lex.yy.c > $(OSTYPE)/GBNSC_lex.C
	@/bin/rm lex.yy.c

$(OSTYPE)/GBNSC_grammar.h $(OSTYPE)/GBNSC_grammar.C: GBNSC_grammar.y
	yacc -d GBNSC_grammar.y
	mv y.tab.c $(OSTYPE)/GBNSC_grammar.C
	mv y.tab.h $(OSTYPE)/GBNSC_grammar.h

NetBSD/GBNSC: $(OBJECTS) $(COM_LIB_DEPENDS)
	$(CC) $(DEBUG) -o $(OSTYPE)/GBNSC $(OBJECTS) -L$(OSTYPE)/ $(COM_LIBS)

solaris/GBNSC: $(OBJECTS) $(COM_LIB_DEPENDS)
	$(CC) $(DEBUG) -o $(OSTYPE)/GBNSC $(OBJECTS) -L$(OSTYPE)/ $(COM_LIBS) -L /project/gbn_sw/fore/lib -latm

purify: $(OBJECTS) $(COM_LIB_DEPENDS)
	purify $(CC) $(DEBUG) -o solaris/GBNSC.purify $(OBJECTS) -L$(OSTYPE)/ $(COM_LIBS) -L /project/gbn_sw/fore/lib -latm

$(OSTYPE)/buildHardwareInfo: $(OSTYPE)/buildHardwareInfo.o $(OSTYPE)/GBNSC_HardwareInfo.o $(OSTYPE)/GBNSC_MR.o $(OSTYPE)/GBNSC_ATMCells.o $(COM_LIB_DEPENDS)
	$(CC) $(DEBUG) -o $(OSTYPE)/buildHardwareInfo $(OSTYPE)/buildHardwareInfo.o $(OSTYPE)/GBNSC_HardwareInfo.o $(OSTYPE)/GBNSC_MR.o $(OSTYPE)/GBNSC_ATMCells.o -L$(OSTYPE)/ $(COM_LIBS)

$(OSTYPE)/GSMP_Client: $(OSTYPE)/GSMP_Client.o $(OSTYPE)/GSMP_CLib.o $(OSTYPE)/GSMP_Engine.o
	$(CC) $(DEBUG) -o $(OSTYPE)/GSMP_Client $(OSTYPE)/GSMP_Client.o $(OSTYPE)/GSMP_Engine.o \
	$(OSTYPE)/GSMP_CLib.o -L$(OSTYPE)/ $(COM_LIBS) -lERR

depend:
	mv -f $(OSTYPE)/Make.depends $(OSTYPE)/Make.depends.bak
	touch $(OSTYPE)/Make.depends
	$(CMS)/../bin/g++dep -fO $(OSTYPE)/Make.depends $(OSTYPE) $(DEBUG) -I. -I$(OSTYPE) $(CMINC) *.C

include $(OSTYPE)/Make.depends

Downloading Software

- **To download software, visit the Software page located off Gigabit Networking Technology Distribution Program web page:**

  http://www.arl.wustl.edu/~jst/gigatech/kits.html

- **There is a link on that page for downloading the tar file which contains all the source code and a current set of binaries.**

Reporting Bugs

- **To report a bug in the software, visit the Gigabit Networking Technology Distribution Program web page:**

  http://www.arl.wustl.edu/~jst/gigatech/kits.html

- **There is a link on that page for Known Bugs. Check there first to see if anyone else has already reported the problem and if there is perhaps already a solution**
- **If not, go to Bug Reports and fill out the form.**

Miscellaneous Things You Should Know

- TCP bind problems when you quit things in the wrong order.
- Jammer occasionally leaves bogus file in /tmp/__JAM*
- VPI/VCI allocation (i.e. one set of VCIs per port!!)
  - VPI/VCI 5/100 is the same table entry as VPI/VCI 0/100
- IPC - semaphores and shared memory.
  - If processes do not exit cleanly you may need to manually remove semaphores and/or shared memory
  - ipcs(1) and ipcrm(1) are good commands to know about.

    > ipcs -p is good for seeing if there are any left behind
  - I use this script to clean up ones that I own when I am debugging:

```
shm=‘/usr/bin/ipcs | grep jdd | grep "^m" | /bin/cut -c2-9‘
echo $shm
sems=‘/usr/bin/ipcs | grep jdd | grep "^s" | /bin/cut -c2-9‘
echo $sems
for i in $shm
  do
    echo "Removing shared memory segment $i"
    /usr/bin/ipcrm -m $i
  done
for i in $sems
  do
    echo "Removing semaphore $i"
    /usr/bin/ipcrm -s $i
  done
```

- It may take as many as 3 operations to activate a connection:
  - write vcxt 2 100
  - write vpxt 2 0
  - write mr 2 2

Errata
or
What I Learned in the last Session

- ‘test cell’ commands are not described in the Jammer documentation
- Need EOL at EOF
- LT should be included in all operations but isn’t
- Jammer prompts are sometimes confusing. Think:

  ADR1 == COPY1 == PORT1
  ADR2 == COPY2 == PORT2
- some missing default indications in Jammer
- filenames beginning with numbers do not work.
- Long timeouts for ‘get cells’ may have strange behavior
- ‘read mr 9 1’ seemed to crash GBNSC (9 being out of range of ports)
- Jammer should really have commands like:

  change vpxt vpt <port> <index> <value>
- ‘write/clear mr’ - review which bits are writeable and which are not.
- some range checking seems to be done when it shouldn’t
- ‘write vcxt’ with RC = {1,2} may appear to fail when given non-zero ADR1 and ADR2.