;login:
THE MAGAZINE OF USENIX & SAGE
February 2003 • volume 28 • number 1

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

the tclsh spot

by Clif Flynt

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

clif@cflynt.com

Analyzing Network Usage

The previous Tclsh Spot article (October 2002 *;login:*) described using SWIG to build a Tcl extension that could build and transmit datagrams over an Ethernet.

Before I'd finished the article, a friend requested a package that could transmit various datagrams over an Ethernet, but it needed to be fast. He was concerned that an interpreted language like Tcl wouldn't be able to put datagrams onto the Net fast enough.

When I test a system, I prefer to use some other platform to test it from. Since I'm generating the packets with a Tcl script on one computer, I prefer to analyze the output on another piece of hardware.

Fortunately, I have a Spirent/AdTech AX-4000 broadband analyzer handy, and it can be programmed using Tcl.

This article will briefly describe the Spirent/AdTech AX-4000, and the AdTech Tcl extension, and show how to use the equipment to check how fully a network is being utilized.

The AX-4000 (*http://www.adtech-inc.com/*) is a configurable piece of hardware that can generate and analyze data packets on four different transmission technologies (IP, ATM, Ethernet, and Frame Relay) simultaneously at speeds up to 10Gbps.

A simple system will include a controller card set and an interface card set. The interface card set contains the circuitry to generate and analyze packets for a transmission medium. A controller card set can control multiple interface cards, which can be mixed and matched to work with Ethernet, fiber, etc.

For this example, the AX-4000 is equipped with a controller and an Ethernet interface card set.

The AX-4000 comes with a nice GUI for performing bench testing, and it also includes a Tcl extension. The primary purpose for the Tcl extension is to support automated testing, but it's also useful for folks who prefer to work outside the GUI.

The general flow for an AdTech Tcl script is:

1. Load the AxTcl extension.
2. Initialize the connection to the AX-4000 controller.
3. Reserve an interface card set.
4. Create a generic interface object attached to the card set.
5. Create an analyzer or generator attached to the interface.
6. Configure the analyzer or generator.
7. Run the test.
8. Analyze the results.

Compiled Tcl extensions can be loaded with the Tcl load command:

Syntax: load *libFile.so ?name?*

Load a shared library extension into the Tcl interpreter.

libFile.so The name of the library to load. The filename suffix will depend on the base operating system.

?name? An optional name for the Tcl initialization function.

The AxTcl extension is available for Solaris, Linux, or Windows platforms. One trick for writing a script that will load on all platforms is to use the catch command to see if the extension loads correctly, and step on to the next possibility if the load fails.

```
if {[catch {load $base/tclwin/libax4k.dll ax4kpkg}]} {
    catch {load $base/tclclib/libax4k.so ax4kpkg}
}
```

Once the AxTcl extension is loaded, it creates several new Tcl commands, each of which has several subcommands. The new commands include:

ax	Interacts with an AX-4000 system.
interface	Establishes and configures a connection to the generic interface.
enet	Interacts with an Ethernet connection.
analyzer	Establishes and configures a connection to the analyzer.

The ax commands provide the high-level control needed for the interactions with the AX-4000 equipment.

One of the features that make the AX-4000 series so fast is that they make heavy use of programmable logic. This feature allows

the hardware to be configured for specific tasks, which are then hardware driven rather than software driven. Programming the hardware allows the AX-4000 to do things like saturate the largest optical fiber with packets and analyze them in real time.

The programmability of the AdTech hardware also means that special hardware configuration files must be available to program the boards for the various tests to be performed.

The ax hwdir command tells the AxTcl extension where to find the hardware configuration files.

Syntax: ax hwdir *path*

ax hwdir *directory*	Identifies the directory for the AX-4000 BIOS files. Default is: ../bios

Once the system knows where to find the BIOS files with the programmed logic definitions, you can initialize the connection to the AX-4000 with the ax init command.

Syntax: ax init *?-option value?*

Initialize internal tables in the AX library.
Options include:

-remote *IP*	The IP address of an AX-4000 accessed via an Ethernet port.
-user *name*	A username that will be used to identify who is using this AX-4000.
-nobios *1/0*	By default ax init will download BIOS to a freshly powered-on AX-4000. Setting this to 1 will inhibit that download.
-forceload *1/0*	Forces the AX-4000 to get a new BIOS upload. When working with multiple revisions of AxTcl, this is recommended.

Initializing the connection to an AX-4000 can be done with these two lines of Tcl code:

```
ax hwdir $base/bios

ax init -remote $ipAddress -user clif -forceload 0
```

The AX-4000 can support multiple users and multiple interface cards on a chassis, but only one user at a time can use an interface card set. To avoid having two applications fighting for control of a card set, the AxTcl extension allows an application to lock (and release) the physical card set for an application's use.

The two commands that control this for an Ethernet card set are enet lock and enet unlock.

Syntax: enet lock *LogicalID ControllerIndex DeviceID*

	Locks a device for this script's use and assigns a logical ID to that device. NOTE: Throws an error if device is already locked.
LogicalID	A value provided by the script to use to reference this locked device.
ControllerIndex	The IP Address/Hostname of this AX-4000.
DeviceID	The position of the card being locked (counting from 1).

Syntax: enet unlock *LogicalID*

enet unlock	Unlocks a device identified by LogicalID from a previous enet lock command.
LogicalID	The device identifier assigned in a previous enet lock command. If this parameter is left out, all devices previously locked in this session are unlocked.

The enet lock command will throw an error if another user has locked a card set. Once the lock has been successful, however, a script can create an interface to the card set.

Syntax: interface create *Name Device ?-key value?*

interface create	Create a new interface object.
Name	The name to assign to the new interface.

Device	The device to attach this interface name to.
?-key value?	Option and value pairs to control how the interface behaves or to configure the card set.

These options vary from card set to card set, and may include:

-interface A|B	For dual interfaces, selects the left (default) as A or right B interface.
-ifmode *type*	Defines the type of data to be used on this interface. Values for interface type include:
POS	Sonet Packets.
IPoETHER	Internet protocol datagrams encapsulated in Ethernet frames.
IPoPPP	Internet protocol datagrams encapsulated in PPP frames.
IPoATM	Internet protocol datagrams encapsulated in ATM frames.
IPoFR	Internet protocol datagrams encapsulated in Frame Relay frames.

The interface create command creates a new command with the same name as the interface you've created. Your script will use this new command to interact with the interface. Two of the main subcommands for the new interface are set (to set device-specific options) and run (to start the interface).

The code to create, configure, and start an interface looks like this:

```
interface create int1 $logicalID -ifmode IPoETHER
int1 set -mode normal -dataRate MBS10
int1 run
```

The next step is to create an analyzer and/or generator object attached to the interface object. Creating the analyzer or generator follows the same pattern as creating the interface.

Syntax: analyzer create *Name Device*

analyzer create	Create a new analyzer object.
Name	The name to assign to the new analyzer.
Device	The device to attach this analyzer name to. This is the logical device that was locked in a previous enet lock command.

The analyzer create command will create a new analyzer object and a new command to use to interact with that object. The analyzer command supports many subcommands, including:

analyzerName set *Name Device*
: Sets one or more configuration options for this analyzer. Configuration options vary for different analyzer cards.

analyzerName display
: Returns a list of the current settings.

analyzerName run
: Starts the analyzer running.

analyzerName reset
: Stops the analyzer and clears all the statistics the analyzer can gather.

analyzerName stop
: Stops the analyzer but does not clear any values.

analyzerName destroy
: Destroys the analyzer, freeing it for other use.

analyzerName stats
: Returns a set of keyword-value pairs as a list. The exact return depends on the analyzer being used.

The analyzer can do lots of interesting things, including capturing packets, generating histograms of the data, and much more. For this application, all we need is to look at the statistics that the AX-4000 analyzer gathers whenever it's running.

This code resets the analyzer, runs it for two seconds, collects the runtime statistics, and releases the device for other users.

```
# Reset the statistics to 0
ana1 reset.
```

```
# Pause until the AX-4000 completes its action
after 400
# Start the analyzer
ana1 run
# Wait 2 seconds and get the statistics
after 2000
set anaStats [ana1 stats]
# Stop the hardware and
# destroy the software object
ana1 stop
ana1 destroy
# Finally, unlock the device for the next user
enet unlock $logicalID
```

Most AxTcl commands return their results as a list of keyword and value pairs. The Tcl foreach command makes this data format easy to use.

Syntax: foreach *varList dataList body*

	Evaluate *body* for each of the items in *dataList*.
varList	A list of variable names. Data values will be extracted from the *dataList* and assigned to these variables.
dataList	A list of data values to step through.
body	The body of code to evaluate on each pass through the loop$

The data will be easier to read if it's formatted as columns. The Tcl format command implements the same string formatting rules as the C library sprintf command.

Syntax: format *formatString value1 ?value2?...*

This code will display a table of keywords and values from the analyzer:

```
puts "ANALYZER STATS"
foreach {key1 val1} $anaStats {
    puts [format "%-30s %12s" $key1 $val1]
}
```

The output looks like this:

```
ANALYZER STATS
-elapsedTime                  2081
-totalPackets                19110
-totalPacketBytes          1949220
-goodPackets                 19110
-goodPacketBytes           1949220
-goodDatagramBytes         1605240
-totalPacketRate             10160
-goodPacketRate              10160
-goodPacketBitRate            8291
-goodDatagramBitRate        6827.5
-lineRatePerc                111.70
-tcpPackets                      0
-tcpRatio                     0.00
-tcpChecksumErrors               0
-udpPackets                      0
-udpRatio                     0.00
-udpChecksumErrors               0
-icmpPackets                19110
-icmpRatio                    1.00
-ipPackets                 19110.00
-ipChecksumErrors             0.00
-avgDatagramLength              84
-minDatagramLength              84
-maxDatagramLength              84
-avgPacketLength               102
-minPacketLength               102
-maxPacketLength               102
-substreamCount                  1
-substreamErrorCount             0
-filterCount                     2
```

Dividing the -totalPacketBytes value (1,949,220) by the 2.081 seconds that the test ran gives 936,674 bytes/second, which is fairly close to 100 percent usage of the 10 megabit/second theoretical bandwidth of the network.

This provides a quick introduction to the AX-4000 and AxTcl. The next few articles will discuss generating different types of Ethernet frames, verifying the generator with the AX-4000, and using those frames to validate a Linux-based firewall.

As usual, the code for these examples is available at *http://www.noucorp.com*.