ND: A Comprehensive Network Administration and Analysis Tool

Ellen L. Mitchell, Eric Nelson, & David K. Hess – Texas A&M University

ABSTRACT

ND is a software tool developed by the Computing and Information Services Network Group at Texas A&M University (TAMU) to aid in the engineering and operation of the campus network. This tool was developed in response to the tremendous growth of the TAMU campus network over the last ten years. ND is designed to provide high-level application functionality while retaining the power and flexibility of a low level tool. It integrates remote network device analysis (via SNMP) with network defining databases (via SQL). ND is written in Python [1] and contains custom modules for interaction with SNMP and MySQL [2].

Introduction

ND is a tool that was developed to aid in the operation and, more importantly, the engineering of the TAMU campus network. With a network of more than 25,000 nodes and over 1000 network devices, a tool was required that could scale to a high level and provide useful functionality without the need to repeatedly develop special purpose scripting based solutions.

One of the major design goals of ND was to have a command line interface (CLI) syntax that was accessible remotely via Telnet/SSH. This is a very important feature because this type of remote access is the lowest common denominator that can usually be found on almost any host or network device in the field. It requires neither a graphical interface nor the installation of special software. In addition, the CLI syntax provides a common command structure independent of different vendors' software implementations due to the use of standard SNMP MIBs.

Another design goal was to integrate ND into the Unix environment in order to leverage and adopt some of the features of Unix that have given it the ability to scale. This integration also made it simple to use a number of freely available support tools and technologies such as Python, SNMP and SQL. The resulting synergies have resulted in a very powerful and useful tool.

This paper presents a background section on network management applications that explains why ND needed to be developed, a design section covering the architecture of ND, a section on how ND is used, and finally a section discussing what the resulting benefits have been.

Background

A quick search of the World Wide Web reveals that there are a remarkable number of packages available both commercially and as open source that purport to perform ''network management''. This leads one to ask, ''Why build yet another network management tool?''

Network management applications have evolved into four major categories: event management, performance management, policy management and finally ''generic'' network management. Event management applications are based on the monitoring and managing of network events such as the reception of SNMP traps and up/down state information based on ICMP echo request/replies (ping). Two tools in this category are Big Brother [3] and Micromuse's Netcool/Omnibus [4] product. Performance management tools collect, monitor, and present network performance information (usually collected via SNMP). InfoVista [5] and MRTG [6] are examples of tools in this category.

Policy management mainly refers to proprietary applications developed by network equipment vendors that attempt to manage complex network equipment functionality (such as quality of service and security) via a policy abstraction. Cisco and 3Com are examples of vendors that have developed tools of these types. Finally, the category of ''generic'' network management applications is the most recognizable one. Applications of this type tend to center around a Graphical User Interface (GUI), which displays information about network topology, network elements and status. Commercial versions of these tools typically incorporate an element manager, which provides a GUI that allows in-depth control of a network element. HP's OpenView [7] and Scotty/Tkined [8] are examples of this type of application.

One of the weaknesses of network management applications as a whole is that they tend to address only the operational aspect of ''network management''. Those who engineer networks tend to eschew these applications for their work because of their constrictive nature; while GUIs are useful for displaying complex information, they are typically implemented in such a way that limits the ability of users to perform complex operations on medium to large sets of network elements. They also do not take into account nor provide support for the specific network management practices that may be in use at a particular site (naming conventions, addressing conventions, network layouts, port numbering patterns, organizational structure, etc.).

True to their nature, network engineers tend to focus on network management tools rather than network management applications as categorized above. These tools are typically programming/scripting languages (Perl, Python, Expect, etc.), SNMP MIB browsers, libraries and standalone tools (SNMP++, UCD Snmp, etc), and databases (MySQL) The result is that engineers build special purpose applications on a case-by-case basis that bridge this gap between the tools and the applications. This software tends to be powerful and effective but special purpose and difficult to maintain and extend.

ND was developed to specifically address this problem; it has been designed to be a powerful tool built on SNMP MIB functionality and SQL databases that provides a useful and friendly application interface. While the Simple Network Management Executive [9] (SNMX) package provides a powerful scripting and SNMP tool structure and is thus closest in nature to ND, ND is unique in the level of functionality it provides in the form of a flexible tool.

ND Architecture

Python

ND is written in Python and contains custom modules for interaction with MySQL and SNMP. Python was chosen for its ease of programming and its modular and object-oriented design. There are over 40 modules in ND (approximately 10,000 lines of code) grouped into 8 families. Table 1 summarizes the module families. The module families closely follow individual MIBs (both standard and proprietary). Several of the standard Python modules have been customized while another has been newly written. Table 2 summarizes the modified or created Python modules.

To assist in creating new modules, a Python class was written from which all modules are sub-classed. The parent class provides methods for parsing user input, redirecting output to pipes or files, providing a command-line history, and providing a mechanism for methods common to all modules, e.g., help.

ND contains an extensive help system that is available at two levels. First, at each ND prompt, a question mark can be entered which will list all the commands available in that module. Second, the help command can be issued with a specific command which will cause the syntax and description of thecommand to be displayed.

Module	Description
mysql	Modified the initial connection code to use MySQL conf files
readline	Modified to allow for user defined history files
snmp++	A new module that allows Python access to the SNMP++ and UCD SNMP libraries

Table 2: Modified/new python modules.

Databases

ND makes extensive use of SQL tables and uses the MySQL server as a backend. Table 3 lists the major categories of SQL tables. Originally, ND used an early version of msql and db, which essentially had the ability to store simple flat files. To ensure that the database made sense from a higher network design perspective, ND was designed to keep databases consistent and free from typical types of errors as might be prone with manual input. This includes sanity checking of record relationships on the creation of new records and automatic generation of record id numbers (functionality that is now found in MySQL).

The Fiber Circuits and Twisted Pair Drops categories are the result of extensions that have been made to ND over time. These are accessed via the equivalent of small applications within ND that specialize in the management of the campus network's fiber optic plant and twisted pair wiring plant. This functionality was added to ND due to the CLI infrastructure ND provided and due to the usefulness of sharing this information with the rest of ND. Fundamentally, these small applications provide support for the processes used at TAMU related to plant management and demonstrate the ability of ND to support site-specific business practices.

Module Family	Description	MIB
fms	3Com FMS Repeater MIBs	Proprietary
ost	Alcatel Switching MIBs	Proprietary
bridge	Bridge MIB	RFC 1286 [10]
fr	Frame relay MIB	RFC 1315 [11]
repeater	Repeater MIB	RFC 1516 [12]
rmon	RMON MIB	RFC 1271 [13]
snmp	Basic SNMP Query	RFC 1213 [14]
netdb	Database applications	n/a

Table 1: ND module families.

Table Category	Description
Support	Various support data such as campus buildings and department descriptions.
Fiber Circuits	Defines fiber circuits
Twisted Pair Drops	Defines each installed drop on campus.
Device	Contains basic information about backbone devices

Table 3: Database architecture.

SNMP

SNMP is a fundamental aspect of ND [15]. Since Python contains no built-in support for SNMP, a new module was created. Initially, a set of functions that used the system method of the OS module was used to interface (externally as child processes) with the UCD SNMP [16] set of utilities. This proved inefficient and more importantly, resulted in a security risk since community strings would appear in the process table as arguments.

A search for lower-level tools was initiated and the SNMP++ [17] toolkit was discovered. This toolkit provides a very efficient interface into the SNMP GET, PUT and TABLE functions but it does not have any provision for parsing MIBs. For this reason, the interface to the UCD snmptranslate function was kept but an OID translation cache was added as one of the SQL tables. This cache all but eliminated the inefficiencies of using an external program.

It should be noted that the SNMP v1 protocol is insecure [18] by its nature since authentication tokens (community strings) are sent in plain text. Steps should be taken to guard against others snooping traffic on the network and discovering the community strings, which may provide the ability to change the configuration of a device. At TAMU, the campus network has been engineered in order to prevent eavesdropping.

Performance Issues

Since Python is used primarily to implement the user interface, the performance of ND is more closely tied to the performance of the underlying SNMP and MySQL toolkits rather than Python. It has turned out that the load on the MySQL database server, even with a network as large as TAMU's, has not been shown to be a problem.

The real bottleneck has turned out to be the network devices response times to SNMP queries. Depending on the CPU and memory of the network device, it may take tens of milliseconds to respond to an SNMP query. When large tables need to be traversed (which must be done serially under SNMP v1), this can result in a large delay before an ND command completes.

Another performance related issue is how ND reacts to poor network conditions. SNMP is a stateless UDP protocol; reliable communications to a network device must be ensured by the SNMP software. Typically, when SNMP responses fail to return to an SNMP management agent, the agent will retransmit the SNMP request. Under poor network conditions, it is important to be able to control this behavior based on the desires of the user and the severity of the network conditions. ND provides timeout and retries parameters that control how long it takes for an individual SNMP request to timeout and how many timeouts are allowed before communication with the device is considered to have failed.

Command Structure

ND is a hierarchical, command-line interface program. We made this choice so that shell scripts could be written to automate complex and repetitive tasks. Also, this tool is frequently used in the field where the only available interface is a Telnet/SSH session.

The hierarchical nature of ND follows a logical progression that we have found assists in problem tracking and security incident investigations. For example, the FMS family contains six sub-modules: address, ports, security, traps, users, and misc. Each of these in turn contains between five and ten individual commands. Similar structures exist for Alcatel, RMON, and bridge MIBs.

The current position in the hierarchy is reflected in the ND prompt. When first invoked, ND starts at the root level. To navigate deeper into the hierarchy, the user needs only type the name of a module available at that level. The quit command is used to leave a module and to return to a higher point in the hierarchy. The '..' notation is used as a mechanism to reference another module's commands without leaving the current module. When a hostname is required in a command, the '!' notation can be utilized to reference the host used in the previous command. This is very helpful when the command is related to the same host but is different to the point that command line history editing is not very helpful.

The GNU readline toolkit, which provides command line editing and history, has been incorporated into ND via the readline module found in Python. It has been modified to generate user defined command line history files, which allow the history to be persistent between ND sessions.

There are a number of commands common to most ND modules: list, show, set, add, delete, enable, disable and help. These commands have basically the same syntax but are tailored to the functionality of the particular module. The commonality of the syntax provides a user with a good base understanding of how commands within any module work. They need only use the help command to find the specific syntax.

One of the most powerful features of the command line interface is the ability to pipe output of any command to a Unix shell command or to a file. This is accomplished by using the normal shell characters of '|', '>', and '>>' at the end of an ND command. As an indication of this level of integration, ND does not have any output paging mechanism. The user is responsible for piping the output of any command to the paging program of their choice (e.g., more).

Using ND

All TAMU network operators and engineers use ND. The types of tasks that each group performs are quite different but ND is flexible enough for both. These tasks include troubleshooting, administration, monitoring, and maintenance.

Starting ND is as simple as typing nd. Before presenting the top level prompt of nd>, ND will execute any commands found in ~/.ndrc . Typically, some top level set commands are executed in order to establish certain operating parameters that ND needs. The set command allows you to specify the following:

sqluser	user name to connect to MySQL server
sqlpass	password to connect to MySQL server
sqlhost	host where MySQL server is running
timeout	number of seconds for SNMP timeouts
retries	number of retries for SNMP queries
snmplimit	maximum number of rows to return in SNMP queries
read	SNMP read community string
write	SNMP write community string
rows	maximum number of rows to output before reprinting column headings
debug	sets the internal debug level for MySQL and SNMP
historyfile	sets the file to use for the history file between ND sessions
echo	toggles ND echoing user commands

Another command commonly found in .ndrc files is the attach command. The attach command allows simple tags to be defined for long or complex host names and also allows for community strings to differ from the default. For example, hostnames of networking equipment at TAMU follow a set pattern but are often tedious to repeatedly type.

Once in ND, the user need only type the name of a module to navigate deeper into the hierarchy and quit to navigate back up the hierarchy. The prompt will change to indicate which module the user is currently in. Within each module the help command provides module and command specific help. There is no documentation or training material for ND other than the internal help system.

As mentioned previously, the ND command line interface is integrated with the Unix shell. Most commonly, users will need to take the output of commands and pipe them to an output pager of their choice. More experienced users will save the output to a file or pass it into a grep command looking for a particular pattern. Expert users will pass the output to awk or possibly even an external Perl or shell script. In support of this functionality, the rows parameter can be set such that no column headings are printed with a command's output.

Rather than give an exhaustive explanation of all ND modules and commands (which space will not allow), it is more useful to show by example, how ND is used at TAMU day-to-day. The following examples show how ND is used in a variety of both operational and engineering related situations.

The general format of common ND commands is:

```
<command> <host> <unit> <port> <options>
```

The units and ports can be specified using a scalar, list, or range notation.

Scenario 1: Ports 1, 3, 4, 5 and 8 of unit 1 of a new 3COM FMS have recently been activated. Before the users are informed that the ports are available, the configuration of the ports must be verified. The command shown in Figure 1 would be used.

Scenario 2: Ports have become scarce inside one building. It is suspected that some ports that are allocated are not actually being used. Viewing only the ports on unit 1 that have a total frame count greater than zero would indicate which ports are actually being used. The ND command and corresponding output are shown in Figure 5.

Scenario 3: Several users have called and complained that the network in their building has become unresponsive. It is suspected that someone is using too much bandwidth. By inspecting the frame counts for all ports on a device and then sorting the output, the ports using the most bandwidth are easily found. The output for this example is in Figure 6.

```
nd-fms-ports> show fms-device-1.domain.com 1 1,3-5,8 config
Unit Port  Status  EST filter  Part. trap  Link trap  Link pulse  DUD action
--------------------------------------------------------------------------------
  1    1     on       mac       enabled     disabled    enabled     N/A
  1    3     on       mac       enabled     disabled    enabled     N/A
  1    4     on       mac       enabled     disabled    enabled     N/A
  1    5     on       mac       enabled     disabled    enabled     N/A
  1    6     on       mac       enabled     disabled    enabled     N/A
  1    8     on       mac       enabled     disabled    enabled     N/A
```

Figure 1: Verifying port configuration.

```
RMON Stats  Variable: Packets Received
              3           4
  +------------------------
1 |    302414796           0
2 |            0    34631342

Help: ?
Host: host.domain.com
Mode: absolute
Rate: 0.5/sec Sample: 57
```

Figure 2: Dynamically monitoring RMON variables across all ports.

Scenario 4: A recent addition to ND has been the ability to continuously monitor a specific MIB variable across all units and ports of a network device. Python includes a Curses module that allows for simple screen control. This has been utilized to display a matrix of text where the columns represent slots or units, and rows represent ports. Figure 2 contains a single screen snapshot showing the RMON etherstats variable for the number of packets received. ND will query the device and update the screen every two seconds.

Scenario 5: There are over 500 3COM FMS type devices at TAMU. Our monitoring strategy is to have statistics that are summarized on a weekly basis. Clearing all the statistics on every FMS then becomes necessary. ND was specifically designed to be incorporated into shell scripts so that fairly complex tasks can be accomplished easily. Figure 3 shows how all the statistics on all FMS devices on campus can be cleared at the same time.

```
(
  echo "fms ports"
  grep -v '^[#%]' $FMSHOSTS |
  while read host
  do
    echo "clear $host of all statistics"
  done
  echo "exit"
) | nd
```

Figure 3: Shell programming with ND.

Scenario 6: The characteristics of all ports and units for a given device can be shown in a single table. This can be very time consuming when logging directly into a network device. Also, the device's software may not support tabular output of information. The ND command and resulting output for this example are shown in Figure 7.

Scenario 7: With over a thousand network devices at TAMU, new devices are constantly being added and old ones replaced. Using ND, a device can easily be configured to a known state. Figure 4 demonstrates how a set of specific ports can be configured easily using a shell script.

```
nd-fms-ports> show fms-device-1.domain.com 1 all counters | \
                                  awk -- '{if ($7 > 0) print $0}'

                  Uni  Mult Bcast    Tot Ucast Mcast Bcase    Total
Unit  Port  Util  frms  frms  frms   frms  octs  octs  octs    Octets  Colls  Runts
-------------------------------------------------------------------------------------
   1     2    0%   1286     0    11   1297     0     0     0    116943      2      0
   1     3    0% 145846     0  5966 151811     0     0     0  20298744    209      0
   1     6    0%   1638     0   764   2402     0     0     0    468142      0      0
   1     7    0%   6697     0  1001   7698     0     0     0    531474    105      0
   1     8    0%  16217     0   668  16885     0     0     0   1220390     16      0
   1    10    0%   3268     0    18   3286     0     0     0    229193      1      0
```

Figure 5: Discovering which ports on a device are active.

```
nd-fms-ports> show fms-device-1.domain.com 1 1-10 counters | sort -k 4 -r -n

                  Uni   Mult Bcast   Tot Ucast Mcast Bcase    Total
Unit  Port  Util  frms  frms  frms   frms  octs  octs  octs    Octets  Colls  Runts
-------------------------------------------------------------------------------------
   1     3    0% 205380     0  8453 213833     0     0     0  31735634    666      0
   1     8    0%  23148     0   951  24099     0     0     0   1726534     63      0
   1     7    0%   9413     0  1415  10828     0     0     0    746284    395      0
   1    10    0%   4627     0    18   4645     0     0     0    324124     13      0
   1     6    0%   2518     0  1090   3608     0     0     0    679204    184      0
   1     2    0%   1803     0    17   1820     0     0     0    164001      6      0
   1     9    0%      0     0     0      0     0     0     0         0      0      0
   1     5    0%      0     0     0      0     0     0     0         0      0      0
   1     4    0%      0     0     0      0     0     0     0         0      0      0
   1     1    0%      0     0     0      0     0     0     0         0      0      0
```

Figure 6: Finding which ports on a device are using the most bandwidth.

Scenario 8: A customer calls and wants to know the speed of the workstation connections in his office. He provides the building and room number. The caller also wants to know the location of the 100 MB connections in the building. Figure 8 shows the commands used to answer this query.

Scenario 9: A student in a residence hall has violated the acceptable use policy and is causing a

```
#!/bin/sh
# Find out which ports to configure as secure
snmptable -mPRODUCTMIB -R $TARGET public -H \
        mrmPortTable mrmPortCardIndex mrmPortIndex mrmPortInterfaceType |
        grep 'twistedPair' | sed -e 's/^ *\([0-9]*\) *\([0-9]*\).*/\1 \2/' |
(
echo set read commread
echo set write commwrite
echo fms security
while read PORT
do
        echo set $TARGET $PORT addresses 1
        echo set ! $PORT intrusion noAction
        echo set ! $PORT ntk NTKWithBcastAndMcast
        echo set ! $PORT mode continuousLearning
done
echo quit
echo exit
) | nd > /dev/null
```

Figure 4: Shell script for configuring a new device to a known state.

```
nd-fms-ports> show fms-device-1.domain.com 1 all diagnostics
           Total                Colls/ Runts/        Late
Unit Port    frms  Coll's Runts  frame  frame  Frags  colls
-----------------------------------------------------------
   1    1       0       0     0      -      -      0      0
   1    2    2700      12     0  0.44%  0.00%      0      0
   1    3  273442     916     0  0.33%  0.00%      0      0
   1    4       0       0     0      -      -      0      0
   1    5       0       0     0      -      -      0      0
   1    6    4656     283     0  6.08%  0.00%      0      0
   1    7   13857     620     0  4.47%  0.00%      0      0
   1    8   30582      93     0  0.30%  0.00%      0      0
   1    9       0       0     0      -      -      0      0
   1   10    5947      23     0  0.39%  0.00%      0      0
   1   11     815       8     0  0.98%  0.00%      0      0
   1   12    8095      42     0  0.52%  0.00%      0      0
   1   13   56349    2548     0  4.52%  0.00%      0      0
   1   14       5       0     0  0.00%  0.00%      0      0
   1   15       0       0     0      -      -      0      0
   1   16    1574       4     0  0.25%  0.00%      0      0
   1   17   30788      48     0  0.16%  0.00%      0      0
   1   18    1222      66     0  5.40%  0.00%      0      0
   1   19       0       0     0      -      -      0      0
   1   20     790       0     0  0.00%  0.00%      0      0
   1   21      21       0     0  0.00%  0.00%      0      0
   1   22       0       0     0      -      -      0      0
   1   23    2805      40     0  1.43%  0.00%      0      0
   1   24       0       0     0      -      -      0      0
   1   25 14047788 1032746    0  7.35%  0.00%      0      0
   1   26       0       0     0      -      -      0      0
```

Figure 7: Displaying diagnostics for all units and ports in one command.

disruption. An engineer can disable the data ports to the room and schedule a meeting with the owner to discuss the problem. Figure 9 shows the commands used to solve this problem.

Benefits

One of the most important benefits that has resulted from the development and deployment of ND is the availability of a command line interface that is the same regardless of which type and make of network equipment is being accessed. This has always been the potential embodied in standard MIBs, though most network management application software has never taken advantage of it outside of an operational setting. The result has been that it is easier to train users. A module's commands will be effective on numerous devices regardless of the type and make of the device.

Another benefit that engineers have expressed is the ease of accessing ND and the simplicity of its interface. ND can be invoked from any Telnet/SSH session and is a system wide tool. No software needs to be installed in a user's account or on a workstation in the office or in the field. Also, the online help and commonality of the syntax between modules makes it easy to remember commands and learn new modules quickly. Command line history files and command line editing also make it simple to look up recent commands or use a previous command as the starting point for a new one. The result is that engineers are very efficient when using ND.

One of the seemingly mundane features but considered very valuable by the operators and engineers at TAMU is the ability to view a condensed, full page of statistics with a single command. This makes it simple for operators and engineers to get a picture of the overall health of a network device and quickly identify a problem. Usually, this is an action that would require a cumbersome series of queries when using the vendor's user interface.

The integration of the Fiber Circuits and Twisted Pair Drops database functionality has proven to be beneficial also. Leveraging the investment of learning ND and tying the information back to ND commands where appropriate not only saves human resources but has made the databases more valuable than they would otherwise be. In many environments, network databases of these types are considered a burden and are many times discarded because they have a different user interface from other tools and because the data is trapped in a separate system and not available in other contexts.

Another important benefit of ND is its ability to pipe output to a shell command or to a file. This feature gives users a powerful means of extending the functionality of ND in the traditional Unix style. Unlike many network management applications, the designers of ND realized that they could not foresee all the desires that operators and engineers would have and made ND extensible in this way. This has saved users from having to develop new special purpose applications and again saved resources.

Conclusion

ND was designed to be a powerful network management tool and to provide user-friendly functionality and extensibility. The Python language facilitates maintainable modules and a flexible command-line interface allowing the simple addition of new vendor-specific and standard MIB functionality as the network grows. The ND CLI eliminates the need for operators and engineers to learn complex vendor-specific syntax by providing a new interface based on the standard MIBs implemented by all vendors. Unix authentication and authorization mechanisms provide a basis for managing access to network devices independently of the individual device's authentication and authorization mechanisms.

It is clear that this type of approach to network management software is a powerful one. The

```
nd-netdb-drops> show room BLDG 311              The building and room number are entered.

Drop number Type Room Length DB loss Dept. Media    Connected to  Unit Port
-----------------------------------------------------------------------------
BLDG:1164    CAT5 311    106   0.0   DEPT 100BaseTX  dev1-ost5-1   4    7
BLDG:1019    CAT5 311      0   0.0        10BaseTX    dev2-fms2-1  3    6

nd-netdb-drops> show device dev1-ost5-1         One drop is 10MB, the other is 100MB. Check the 100MB
                                                device and see in which rooms the drops are terminated.

  Device    Unit Port  Drop number Type Room  Length DB loss Dept.  Media
-----------------------------------------------------------------------------
dev1-ost5-1    4    1  BLDG:1136   CAT5 319    147     0.0   DEPT  100BaseTX
dev1-ost5-1    4    2  BLDG:1137   CAT5 312    157     0.0   DEPT  100BaseTX
dev1-ost5-1    4    3  BLDG:994    CAT5 011A     0     0.0   DEPT  100BaseTX
dev1-ost5-1    4    4  BLDG:1107   CAT5 011A     0     0.0   DEPT  100BaseTX
dev1-ost5-1    4    5  BLDG:1155   CAT5 107     91     0.0   DEPT  100BaseTX
dev1-ost5-1    4    6  BLDG:1064   CAT5 107      0     0.0   DEPT  100BaseTX
```

Figure 8: Helping a user Locate 100 MB ports.

development of ND began from the desire to build a friendly application that was also a powerful tool. A simple command line interface leveraging Unix was key to accomplishing this. More importantly, ND became a platform in which business practices could be embodied. At the same time, this embodiment can lead to weaknesses. Some business practices coded into ND may not be adoptable in other environments. They can even become a burden in TAMU's environment when it makes sense to change a business practice but ND would require significant redevelopment to support the change.

In the final analysis, every networking organization has a unique culture and a unique set of business practices. That organization must choose software that will support them in their mission. Tools that are too low-level do not have interesting enough behavior for operators and most network management applications do not interest engineers because they are too restrictive or because they are not adaptable to the organization's business practices. Many times, the only perceived alternatives to this problem are to build completely custom network management applications or to change business practices. At TAMU, ND has successfully proven that simply designing a better tool is a viable alternative with many benefits.

Future Work

We are currently looking at integrating ND with an event management package. The event management package could trigger a lookup of a network device at a specific event threshold, execute a set of

```
nd-ost-vports> show dev1-ost3-1 4 12          The engineer checks the status of the port before disabling.
Slot Port Status VLAN         MAC       Prot   Encap Mode  Timer
------------------------------------------------------------
   4  12    on     1  00:20:aa:bb:23:12 TRN  default auto   60
nd-ost-vports> disable dev1-ost3-1 4 10       The engineer disables the port and is then
                                              required to document this action.
Enter a comment to associate with this action: incident number 2000.143
nd-ost-vports> show dev1-ost3-1 4 10          Verify the status.  Now a '*' reflects a comment is
                                              present for this port, and the status has changed to 'off'.
Slot Port Status VLAN        MAC       Prot  Encap  Mode Timer
-----------------------------------------------------------
  4  10  * off    1  00:20:aa:bb:23:12 TRN default  auto    60
nd-ost-vports> show dev1-ost3-1 4 10 comment  If the student telephones the Operations Center before anyone
                                              can reach and discuss the problem with him, the Operations
                                              Center can check the status and inform the student of the situation.
 Id              Host           Unit Port   Comment
------------------------------------------------------------
46294 dev1-ost3-1.net.tamu.edu  4    10  incident number 607.689
nd-ost-vports> enable dev1-ost3-1 4 10        When the problem has been resolved, the port is re-activated.
Enter a comment to associate with this action: incident 607.689 resolved
nd-ost-vports> show dev1-ost3-1 4 10
Slot Port  Status VLAN        MAC       Prot   Encap  Mode Timer
------------------------------------------------------------
  4  10   * on     1  00:20:aa:bb:23:12 TRN  default  auto    60
nd-ost-vports> show dev1-ost3-1 4 10 comment  The comments are still assigned and can be retained, or
                                              can be removed with the 'comment delete' command.
 Id           Host             Unit Port  Comment
---------------------------------------------------------------
46294 dev1-ost3-1.net.tamu.edu  4    10  incident number 607.689
46295 dev1-ost3-1.net.tamu.edu  4    10  incident 607.689 resolved
                                              Timestamp and audit information can be viewed using the
                                              'comment info' command:
ID: 1               IP Address:  192.168.1.1   User: Operator #1 at server
Device: dev1-fms-1  Unit:        1             Port: 6
Date:   1999-10-19  Comment: disabled port for usage violations
nd-ost-vports> comment delete 46294-46295     Delete the comments by comment ID number.
```

Figure 9: Turning off a student's internet access.

predetermined ND commands, and generate a report for an operator to review and make a diagnosis. Additionally, while the Fiber Circuits and Twisted Pair Ports databases have a good command structure for querying them, it can be cumbersome populating the databases with bulk data. Additional work needs to be done on simplifying and bulletproofing this process.

Acknowledgments

We would like to thank the employees of the Texas A&M Network Group (Installation, Engineering and Systems teams) and to the Operations Center for being guinea pigs during the development phase and for patiently providing feedback to us on ND.

Availability

It has always been our intention to release the source code for ND. Extenuating circumstances prevented us from having the source available at the time of this writing; however, we expect to make it available at some time in the future.

Author Information

Ellen Mitchell obtained her Master's of Computer Science from Texas A&M University in 1994. While at Texas A&M, she has been the Systems Manager of the Computer Science Department and is presently a Security Analyst in the campus Network Group. She is involved in educating the campus user community concerning computer security. She may be reached by postal mail at Computing and Information Services, Teague Building, College Station, TX, 77843-3142 or by e-mail at ellenm@net.tamu.edu.

Eric Nelson has just recently left Texas A&M University where he was Systems Group manager of the campus Network Group. He has been the ND maintainer for the last several years and has developed many of the network analyst tools currently in use at TAMU. He received his Master's of Computer Science from Texas A&M University in 1987 and his Bachelors of Science from Texas A&M University in 1985. He may be reached by email at eric@net.tamu.edu.

David Hess was formerly the campus Network Manager at Texas A&M University. He is the original author of ND. David has extensive experience with network engineering and network management. He received his Master's of Computer Science from Texas A&M University in 1991. He recently left TAMU to join a startup company. He may be reached by e-mail at daveh@net.tamu.edu.

References

[1] Lutz, Mark ,"Programming Python," O'Reilly & Assoc., http://python.org/ .

[2] Yarger, Randy Jay, George Reese, and Tim King, "MySQL and mSQL," O'Reilly & Assoc., http://www.mysql.com/ .

[3] MacGuire, Sean, and Robert-Andre' Croteau, "Big Brother is Still Watching," SANS conference, Baltimore, Maryland, http://bb4.com/ , May 1999.

[4] Micromuse, Inc., "Netcool Suite Functionality and Benefits," http://www.micromuse.com/products/ overview.html .

[5] InfoVista, "The Challenge of Service Level Management," InfoVista Marketing, http://www.infovista.com/products/frproducts.html , 2000,

[6] *MRTG*, http://ee-staff.ethz.ch/˜oetiker/webtools/mrtg/mrtg.html .

[7] *HP OpenView*, http://www.openview.hp.com/ .

[8] Schönwälder, J., and H. Langendörfer, "How to Keep Track of Your Network Configuration," *Proceedings, 7th Conference on Large Installation System Administration (LISA VII)*, Monterey, California, http://wwwhome.cs.utwente.nl/˜schoenw/scotty/ , November 1993.

[9] Davison, Jeff, "Network Management Automation Using ACE Automated Console Expert," Diversified Data Resources, Inc., http://www.ddri.com/Products/ace-snmx.html/ .

[10] Decker, E., P. Langille, A. Rijsinghani, and K. McCloghrie, "RFC1286: Definitions of Managed Objects for Bridges."

[11] Brown, C., F. Baker, and C. Carvalho, "RFC1315: Management Information Base for Frame Relay DTEs."

[12] McMaster, D., and K. McCloghrie, "RFC1516: Definitions of Managed Objects for IEEE 802.3 Repeater Devices."

[13] Waldbusser, S., "RFC1271: Remote Network Monitoring Management Information Base."

[14] McCloghrie, K., and M. Rose, "RFC1213: Management Information Base for for Network Management of TCP/IP-based internets: MIB-II."

[15] Stallings, William, "SNMP SNMPv2 and RMON," Addison Wesley.

[16] *UCD SNMP*, http://ucd-snmp.ucdavis.edu/ .

[17] Mellquist, Peter Erik, "SNMP++ Specification," Hewlett-Packard Company, http://rosegarden.external.hp.com/snmp++/index.html .

[18] Galvin, J. M., K. McCloghrie, and J. R. Davin, "Secure Management of SNMP Networks," *Proceedings of the IFIP TC6/WG 6.6 Second International Symposium on Integrated Network Management*, pp. 703-714, North-Holland, 1991.