Evolution of Open Source SNMP Tools

Jürgen Schönwälder – University of Osnabrück, Germany

April 29, 2002

Abstract

This paper reviews the evolution of some major open source SNMP tools over the last decade. It focusses on open source packages that are widely used and which have shaped some of the SNMP technology. The paper identifies some lessons that can be learned.

1 Introduction

The Simple Network Management Protocol (SNMP) [1] was defined in the late 1980s to address the management needs of the evolving Internet. The traditional SNMP architecture introduces two entities: SNMP managers and SNMP agents. SNMP managers execute management applications while SNMP agents provide access to typed MIB variables which are either simple scalars or conceptually organized into tables. SNMP agents can also send asynchronous event notifications to SNMP managers in order to report events. The scalars and conceptual tables communicated with SNMP are formally defined by using a data definition language called the Structure of Management Information (SMI). Data definitions written in the SMI are called Management Information Base (MIB) modules.

The SNMP protocol has been designed to operate in lossy networks and to minimize the resource requirements of the SNMP agents. As a consequence, SNMP agents are generally stateless and the default transport for SNMP messages is the User Datagram Protocol (UDP).

The first version of SNMP (SNMPv1) went through the standards-process quickly and became an Internet Standard in 1990 [2]. The first version of the data definition language (SMIv1) was also published in 1990 [3] and a more fomalized version in 1991 [4, 5]. Both, SMIv1 and SNMPv1 were quickly adopted by the manufacturers of networking devices and vendors of management software. SNMP is supported by almost all network devices and in wide-spread use today.

This paper describes how some major open source SNMP tools evolved over time and it tries to identify some lessons learned. The paper is organized as follows: Section 2 briefly summarizes the evolution of the SNMP technology. Section 3 discusses the evolution of various open source SNMP tools. Note that the selection of the open source tools is subjective and does not try to cover all important packages available. Some lessons learned are discussed in Section 4 before the paper concludes in Section 5.

2 Evolution of the SNMP Technology

The SNMP technology had an interesting evolution during the last decade — not all proposed improvements have been a success story nor was the process within the SNMP community always a nice example of good engineering practice. But despite all the bad press SNMP had over the years, it is widely deployed and used very extensively for gathering statistics and detecting network failures.

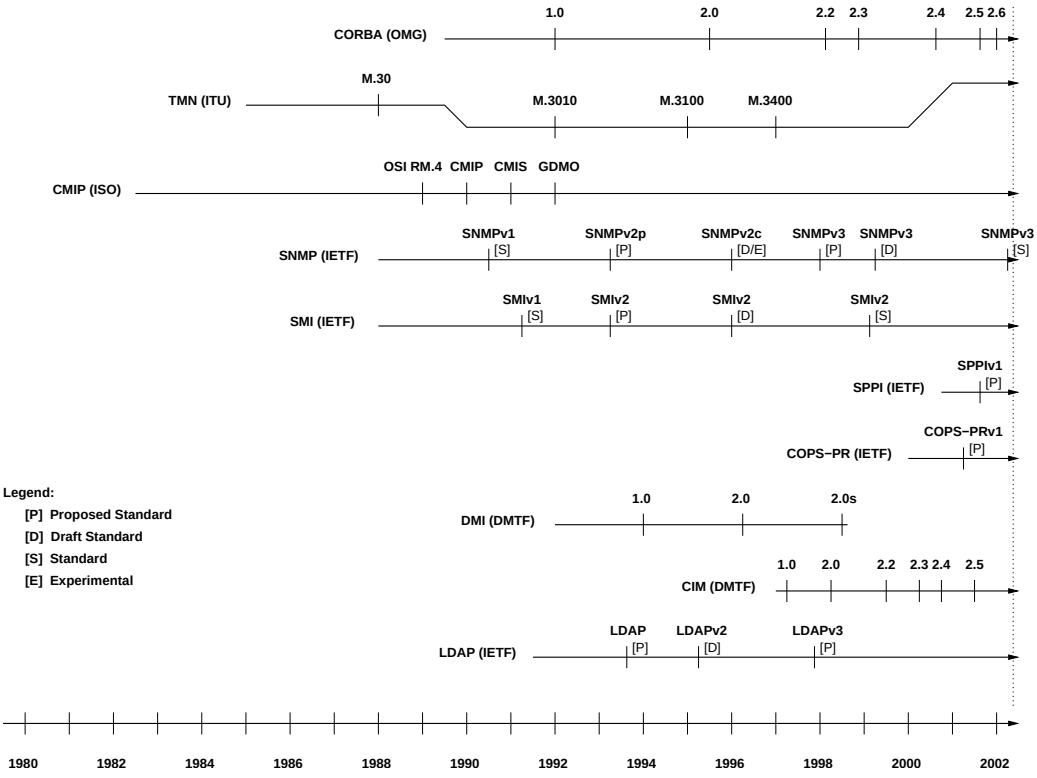

Figure 1: Historical Perspective on Network Management Technologies

Figure 1 shows the evolution of various protocols and data definition languages that have been used for managing communication network. The Internet

Engineering Task Force (IETF) finalized SNMPv1 in the early 1990s and soon started efforts to introduce SNMP version 2 together with a second version of the data definition language (SMI). This process turned out to be difficult and the result was SNMP version 2c (SNMPv2c). SNMPv2c is relatively widely deployed today, although it is not a complete IETF standard as it lacks strong security (which was one of the main goals of SNMPv2). The second version of the SMI (SMIv2) was more successful and was published as an Internet Standard in 1999 [6, 7, 8], eight years after the publication of SMIv1 as an Internet Standard.

Since SNMPv2c did not reach its goals, an attempt was started in 1997 to define SNMP version 3 (SNMPv3) which provides strong security[1], remote administration capabilities and probably most important an architectural framework [9]. SNMPv3 is gaining deployment and the revised documents are waiting for the publication as full Internet Standard which will then replace SNMPv1 and SNMPv2c. It thus took 12 years from SNMPv1 to SNMPv3 with an intermediate version of SNMPv2c which was never an official SNMP standard. More details on the documents that define the various versions of SNMP can be found in [1].

Figure 1 shows that SNMP has many competitors. The International Telecommunication Union (ITU) has defined the Telecommunications Management Network (TMN) series of standards for managing telecommunication networks. The specifications were at some point closely aligned with the Common Management Information Protocol (CMIP) defined by the International Organization for Standardization (ISO). Currently, the ITU is more moving towards CORBA as the communication middleware. The Distributed Management Task Force (DMTF, formerly known as the Desktop Management Task Force) developed the Desktop Management Interface (DMI) until 1998. Since then, the DMTF is widely know for its work on the Common Information Model (CIM). Figure 1 shows the Lightweight Directory Access Protocol (LDAP) standards because LDAP was proposed as part of the Directory Enabled Networks (DEN) initiative.

Just recently, one of the competitors started within the IETF itself: The Common Open Policy Service for Provisioning (COPS-PR) [10] and the associated Structure of Policy Provisioning Information (SPPI) data definition language [11]. There is an ongoing debate whether this competition is a good or a bad sign for people who have to run networks.

[1]The current SNMPv3 specifications provide strong authentication but only relatively weak privacy since the default encryption algorithm is single DES. Work is underway to support AES in addition to DES.

3 Evolution of Open Source SNMP Tools

Open source implementations are crucial for making new technologies or significant revisions of existing technologies successful. The IETF standardization process requires at least two independent implementations in order to move a specification forward. In the early days, these implementations were often created in research environments and freely available. This has also been true for SNMP. The classic SNMP implementations were done at the Massachusetts Institute of Technology (MIT), the Carnegie Mellon University (CMU) and the ISODE consortium. Other free implementations followed later and there are now several second generation open source implementations in widespread use.

3.1 `CMU-SNMP` → `UCD-SNMP` → `NET-SNMP`

In the early days, there were three open source SNMP implementations. The MIT implementation was pretty small and solid. However, it never got widely used since further development stopped pretty soon at the MIT. The ISODE implementation made heavy use of the `ISODE` infrastructure. Porting the whole `ISODE` distribution in order to use it often turned out to be a major effort, especially in cases where the Unix version was not one of the versions the developers used. In addition, since the implementation used lots of compiler generated code, it took quite some time to understand the code. The CMU implementation was easier to understand and extend. Many people started with the CMU version to develop their own SNMP agents and managers.

One of the branches that emerged from the CMU implementation was the Univerisity of California at Davis (UCD) SNMP distribution called `UCD-SNMP`. It was created in 1995 because there was a need for an extensible monitoring system at the UCD. Initially, `UCD-SNMP` was just a set of patches against one of the CMU releases which supported one of the attempts to support security in SNMPv2. The `UCD-SNMP` distribution became quite popular since it was easier to port and provides many new features. The `UCD-SNMP` implementation also was the first open source SNMP implemenation with full SNMPv3 support.

In 2000, a decision was made to rename `UCD-SNMP` to `NET-SNMP`[2] to reflect the fact that the development had become a community effort over the years. Furthermore, one of the lead developers left the UCD so that the UCD was no longer attachted to the project. There is currently work underway towards the 5.0 release of `NET-SNMP`, which is a major cleanup of the implementation and provides many new features.

[2] http://www.net-snmp.org/

3.2 MIB Parser and Compiler

Most SNMP implementations are able to parse MIB modules - if only to convert the binary object identifier names used by the protocol into human friendly names. Some implementations also use code generating MIB compilers.

Many parser implementations (most notably the original CMU MIB parser and parsers derived from it) however do a poor job in validating the input. The `mosy` MIB parser (originally part of the `ISODE` package) was for a long time the only freely available MIB parser which was written using compiler construction tools (lex/yacc) and which did syntax and semantic error checking. Although `mosy` was in principle a good MIB parser implementation, it became hard to maintain it since it was implemented as part of the `ISODE` package which was not trivial to port to new platforms or to integrate into other software packages.

The SMI compiler `SMIC`, later called `SMICng`, is another syntax and semantic checking parser implementation which performs much more semantic checks than `mosy`. However, `SMICng` is not freely available and thus many implementors derived code from free parsers that do a poor job in validating input. As a consequence, many MIB authors were not forced to use strict MIB parsers in order to use their favorite SNMP tools with the effect that we have many published MIB modules which contain sometimes serious syntactic and semantic errors.

Of course, it is quite a bit of programming work to create and maintain a good solid MIB parser. Since this is not cost effective for each little project where one must process information contained in MIB modules, the idea was born in 1998 to create an open source MIB parser library which can be easily embedded into applications. This parser library was also designed to allow researchers to experiment with future versions of the SMI. The result of these efforts is the `libsmi`[3] package [12]. The software package contains a program called `smilint` which is probably today the most widely used MIB validator next to `SMICng`. Many code generators and applications (open source and commercial) exist today that are based on `libsmi`. Since the `smidump` utility of the `libsmi` package is capable to produce `mosy` style output, there is no need anymore to maintain the original `mosy` implementation.

The current `libsmi` release supports several code generating backends (Perl, Python, CORBA IDL, XML, XML Schema, Java AgentX stubs, SCLI stubs) and some experimental algorithms to reverse engineer MIB modules into UML diagrams [13].

[3] http://www.ibr.cs.tu-bs.de/projects/libsmi/

3.3 SNMP and Scripting Languages

SNMP basically provides a simplistic mechanism to manipulate lists of simple typed variables that are conceptually organized in tables. It is obvious that real-world complex systems must be represented by collections of related variables and that there is a need to have control structures to automate the manipulation of related MIB variables. Since SNMP is purely data-driven and does not provide any control structures to "program" procedures on MIB variables, people started pretty early to adopt existing scripting languages for SNMP.

3.3.1 SNMP Query Language

An early and almost forgotten example was the SNMP query language `SNMPql` [14] which realizes an SQL-like query language to retrieve MIB variables. An implementation of `SNMPql` which does interesting optimizations was released to the public in 1990 and could be compiled using MIT's SNMP implementation.

```
select i.ifIndex a.ipAdEntAddr i.ifOperStatus
from ipAddrTable a ifTable i
where a.ipAdEntIfIndex = i.ifIndex.
```

Figure 2: `SNMPql` query to retrieve interface details for each IP address

The example query shown in Figure 2 shows how `SNMPql` can be used to retrieve for each IP address of a multi-homed host the associated interface number, the IP address and the operational status of the interface. Other query languages similar to `SNMPql` were later developed in various other projects. All these languages are nice for interactive use. However, they require some knowledge and understanding of the relevant MIB objects in order to formulate useful queries.

3.3.2 SNMP and AWK

Another early experiment was an SNMP extension for Gnu AWK [15] called `sgawk` which was distributed with the `ISODE` package. It allowed to access and manipulate MIB variables very easily within an AWK script.

Figure 3 shows an SNMP AWK script which displays information about the network interfaces and the associated IP addresses. It is important to note that AWK variables whose names match MIB variables or conceptual MIB tables will be retrieved from the agent automatically whenever they are accessed. This makes the script rather short — but it requires either an efficient and intelligent caching

```
BEGIN {
        printf "%-4s %-4s %-14s %-15s %-7s %-5s %-7s %-5s %-4s %-5s\n",
                "Name", "Mtu", "Net/Dest", "Address", "Ipkts", "Ierrs",
                "Opkts", "Oerrs", "Drop", "Queue";

        didone = 0;
        for (i in ifIndex) {
            didone = 1;

            dest = "";
            addr = "";
            for (j in ipAdEntAddr) {
                if (ipAdEntIfIndex == ifIndex) {
                    split(addr = ipAdEntAddr, a, ".");
                    split(ipAdEntNetMask, b, ".");
                    dest = bit_and(a[1],b[1]) "." \
                           bit_and(a[2],b[2]) "." \
                           bit_and(a[3],b[3]) "." \
                           bit_and(a[4],b[4]);
                    break;
                }
            }

            printf (length(ifDescr) <= 4 ? "%-4s " : "%s\n     "), ifDescr;
            printf "%-4d %-14s %-15s %-7d %-5d %-7d %-5d %-4d %-5d\n",
                   ifMtu, dest, addr, ifInUcastPkts+ifInNUcastPkts,
                   ifInErrors, ifOutUcastPkts+ifOutNUcastPkts,
                   ifOutErrors, ifOutDiscards, ifOutQLen;
        }
        if (!didone && DIAGNOSTIC)
            printf "ifTable: %s\n", DIAGNOSTIC;
      }
```

Figure 3: `sgawk` script to display interfaces and associated IP addresses

scheme or it requires that programmers use these magic AWK variables very carefully. Otherwise, even short scripts can turn out to be very inefficient. Another problem with this implicit approach is error handling. It should also be noted that from todays point of view, the script shown in Figure 3 is problematic. Many systems have multiple IP addresses assigned to an interface which this script fails to handle and some of the statistics may not be present for all the interface types in use today.

It turned out that AWK was not the ideal language for scripting SNMP since it was primarily designed for pattern scanning and processing. People therefore started to develop SNMP extension for other scripting languages that seemed to provide a better fit.

3.3.3 SNMP and Tcl

The Tool Command Language (Tcl) [16] became very popular in the early 1990s. Since Tcl was designed as an extensible embedable command language, it was no

```
package require Tnm 3.0

proc walkproc {s stat vbl} {
    if {$stat == "noError"} {
        set i [Tnm::mib unpack [Tnm::snmp oid $vbl 0]]
        set o [Tnm::snmp value $vbl 0]
        set a [Tnm::snmp value $vbl 1]
        puts "[$s cget -address]\t$i\t$o\t$a"
    }
}

puts "ADDRESS\t\tIFACE\tOPER\tADMIN"
foreach host $argv {
    set s [Tnm::snmp generator -address $host]
    $s walk {ifOperStatus ifAdminStatus} { walkproc %S %E "%V" }
}
Tnm::snmp wait
exit
```

Figure 4: `Tnm` asynchronous walk to retrieve the interface status information

surprise that several people started to use Tcl as a base language.

Probably the first SNMP extension for Tcl was written by Poul-Henning Kamp, originally based on the CMU SNMP implementation. It was first released in Januar 1992. In 1993, a draft Tcl API was circulated which was written by Glenn Trewitt and Poul-Henning Kamp — but there were no open implementations of this API. In 1994, the `scotty`[4] package [17, 18] was released which includes an SNMP extension for Tcl which was derived from Poul-Henning Kamp's previous work but largely re-implemented. In the same year, Marshall Rose and Keith McCloghrie announced their SNMP Tcl API together with an implementation derived from the `ISODE` package [19].

Tcl itself is a simplistic language. It's strength comes from the fact that one can easily introduce new control structures and that there was reasonable support for general purpose event-driven programming. Figure 4 shows a simple script which retrieves MIB variables by doing a so called MIB walk. After loading the `Tnm` extension (which is part of the `scotty` package), a procedure `walkproc` is defined which gets three arguments: a session name, a status code and a list of variables together with their values (varbind list).

The body of the `walkproc` procedure checks the status code and outputs the address associated with the session and the varbind list. The rest of the script iterates through the command line arguments and creates an SNMP command

[4] http://wwwsnmp.cs.utwente.nl/~schoenw/scotty/

generator session for each host. For each session, an asynchronous walk for the `ifTable` defined in the `IF-MIB` module is started. The code in the curly braces is the callback code fragment which is executed whenever a response has been received. The `Tnm::snmp wait` command forces the program to process events until there are no outstanding asynchronous SNMP requests for all SNMP sessions. Thus, the control flow in this example is quite different from what one might conclude when first looking at the code. But the power here is the simplicity and efficiency with which this can be implemented. Furthermore, the `Tnm` implementation provides mechanisms to shape the outgoing SNMP traffic in order to protect the network from bursty SNMP traffic patterns that can be easily generated by scripts like the one shown in Figure 4.

One of the motivations for starting the `scotty` project was to create an open network management platform which people and especially researchers could extend by writing new or modifying existing Tcl scripts. However, this goal was only marginally reached. Instead, the most common usage of the `scotty` package today is probably testing. Several network device vendors use the package to drive test suites. In addition, there are commercial SNMP test tools based on Tcl which in addition provide an environment to define and manage test cases and results.

3.3.4 SNMP and Perl

Perl [20] became the most popular scripting language for system administrators in the 1990s. There are several SNMP extensions for Perl. One of the Perl SNMP extensions (`SNMP_Session.pm`[5]) was created in 1995 and is written entirely in Perl which makes the code highly portable. It currently supports SNMPv1 and SNMPv2c and provides a pattern oriented parser to read MIB modules in order to map human friendly names to the numbers used by the protocol.

The second widely used Perl SNMP API is based on the `NET-SNMP` implementation and thus supports SNMPv1, SNMPv2c and SNMPv3. The Perl script shown in Figure 5 is based on the `NET-SNMP` Perl extension and retrieves information from the routing table defined in the `RFC1213-MIB` module and displays them in a human readable format.

3.4 SNMP Command Line Interface (`scli`)

Many network operators today configure at least the static aspects of their network devices through a command line interface (CLI). This either happens interactively or it is driven by scripts that automate the otherwise interactive configuration pro-

[5] http://www.switch.ch/misc/leinen/snmp/perl/

```
use SNMP;
$SNMP::use_enums = 1;

my $host = shift;
my $comm = shift;
$sess = new SNMP::Session(DestHost => $host, Community => $comm);

$vars = new SNMP::VarList( ['ipRouteIfIndex'], ['ipRouteType'],
                           ['ipRouteProto'], ['ipRouteMask'],
                           ['ipRouteNextHop'], ['ipRouteAge'],
                           ['ipRouteMetric1']);

format STDOUT_TOP =
  Destination      Next Hop          Mask       Proto    Age    Metric
--------------- --------------- -------------- ------- -------- ------
.

format STDOUT =
@<<<<<<<<<<<<<< @<<<<<<<<<<<<<< @<<<<<<<<<<<<< @|||||| @||||||| @|||||
$dest,          $nhop,          $mask,         $proto, $age,    $metric
.

for (($index,$type,$proto,$mask,$nhop,$age,$metric) = $sess->getnext($vars);
     $$vars[0]->tag eq 'ipRouteIfIndex' and not $sess->{ErrorStr};
     ($index,$type,$proto,$mask,$nhop,$age,$metric) = $sess->getnext($vars)) {
    $dest = $$vars[0]->iid;
    write;
}

print "$sess->{ErrorStr}\n";
```

Figure 5: Perl SNMP script to retrieve and display the `ipRouteTable`

cess. Network management tools are sometimes not used to configure devices as they are either too expensive, they lack important features, are hard to automate, do not scale to large numbers of devices or simply do not support the required features on the set of heterogenous devices that exist in a network. However, using command line interfaces is also problematic since they differ between vendors and sometimes even within a single product line.

In 2001, a project was started to realize a command line interface to browse, monitor and configure manageable devices via SNMP [21]. The motivations for this SNMP command line interface (`scli`[6]) were twofold: First, by running the command line interpreter locally and using SNMP and standardized MIBs, it should be possible to manage devices from different vendors in a uniform and convenient way. Second, such a command line interface should provide meaningful data to human operators in a comprehensive way — which is quite different from what most generic SNMP tools do.

Some examples might be useful to make this point clearer. Lets assume that one is interested in monitoring the disk usage across several systems. There are

[6] http://www.ibr.cs.tu-bs.de/projects/scli/

different ways to retrieve the required information. A relatively efficient and low overhead mechanism is the use of SNMP. However, in order to use SNMP for this purpose, one has to find, read and understand the relevant MIBs (the `HOST-RESOURCES-MIB` in this case) and one needs to write some code which computes the actual disk usage from some MIB objects. In order to make this code robust, the implementation should do data validation before processing it and it should be smart to deal with so called "holes" in SNMP tables and other oddities that might happen during the SNMP interactions. Of course, some knowledge of an SNMP API such as the `NET-SNMP` C API or an SNMP extension for a scripting language is needed. It turns out that only few people have the skills and the time available to actually make a good and robust implementation.

```
(xxxx) scli > show system storage
INDEX DESCRIPTION    TYPE             SIZE  USED  FREE USE%
    1 /              fixed disk        10G 2401M 8492M  22
    2 /dev/pts       fixed disk         0K    0K -----   0
    3 /win           fixed disk      7983M 2155M 5828M  26
  101 Real Memory    ram              123M   75M   47M  61
  102 Swap Space     virtual memory     0K    0K -----   0
  103 Memory Buffers other
```

Figure 6: `scli` listing of all storage areas

With `scli`, you can simply retrieve information about the current disk usage by using the `show system storage` command as shown in Figure 6. This `scli` command handles missing data for the memory buffers, provides human readable storage type names, scales values to keep the output compact and readable, and computes derived values where appropriate. Of course, someone who understands the relevant MIBs has invested time to implement this command. However, compared to other approaches, this is relatively easy and straightforward since no knowledge about low-level SNMP APIs is needed. This is possible since `scli` command implementations use compiler generated MIB stubs which do all the low-level SNMP serialization/deserialization and which also perform automated type/range/size checking. Programmers use some nice MIB-specific programmatic interfaces to retrieve and manipulate the MIB data structures in C. In general, the `scli` provides an infrastructure which hopefully increases the number of people who write additional commands for those things not yet covered by the existing commands.

In order to allow more complex management systems to be built on top of `scli`, support for a second XML-based output format was added. The XML-based format is easy to parse by programs and scripts and allows the default format

to be optimized for human readability. Furthermore, care has been taken to make the `scli` implementation efficient and to avoid for example expensive startup times. Calling `scli` in scripts frequently does not cause a performance penalty.

```
(xxxx) scli > show interface details ^Gi
Interface:   3                   Name:    Gi1/0
OperStatus:  up                  Address: 00:03:FD:32:E4:1C
AdminStatus: up                  Type:    ethernetCsmacd
Traps:       enabled             MTU:     1500 byte
Connector:   true                Speed:     1g bps
Promiscuous: false               Change:  2002-03-12 15:32:52 +01:00
IP Address:  134.169.246.1       Prefix:  /24
Description: GigabitEthernet1/0
Alias:       Link to the LAN of the Technical University Braunschweig
Port:        WISEMAN
Module:      GiagbitEthernet
Container:   Chassis Slot
Chassis:     7206VXR chassis, Hw Serial#: 21275454, Hw Revision: D
```

Figure 7: `scli` description of all Gigabit Ethernet interfaces

Figure 7 show the detailed description of an Ethernet interface. Note that `scli` lists the IP address(es) assigned to that interface and also describes the physical hardware on which the interface is implemented. Additional `scli` commands are available to browse the entire containment structure of the physical components that make up a device. Figure 7 also demonstrates that interfaces can be selected by regular expressions. The command argument `^Gi` is a regular expression that is matched against interface descriptions, which selects all Gigabit Ethernet interfaces on this particular router.

```
(xxxx) scli > monitor interface stats
Agent:    xxxx:161 up 58 days 00:57:06                                  15:42:26
Descr:    Cisco Internetwork Operating System Software   IOS (tm) 7200 Software
IPv4:     7614 pps in 7580 pps out 7558 pps fwd    0 pps rasm    0 pps frag
UDP:        29 pps in   24 pps out
TCP:         0 sps in    0 sps out    0 con est    0 con aopn    0 con popn
Command:  monitor interface stats

INTERFACE STATUS I-BPS O-BPS I-PPS O-PPS I-ERR O-ERR  DESCRIPTION
        1  UUCN     3m    2m  5010  4498     0     0  POS2/0
        2  UUCN     49     0     0     0     0     0  FastEthernet0/0
        3  UUCN     2m    3m  4551  5026     0     0  GigabitEthernet1/0
        4  UDCN      0     0     0     0     0     0  ATM4/0
        5  UDNN   ----  ----  ----  ----  ----  ----  ATM4/0-atm layer
        6  UDNN   ----  ----  ----  ----  ----  ----  ATM4/0.0-atm subif
        7  UDNN      0     0     0     0  ----  ----  ATM4/0-aal5 layer
        8  UDNN      0     0     0     0  ----  ----  ATM4/0.0-aal5 layer
        9  UUNN      0    68     0     1     0     0  Null0
       10  UDC-   ----  ----  ----  ----  ----  ----  E1 5/0
       11  UDC-   ----  ----  ----  ----  ----  ----  E1 5/1
       12  UUNN      0     0     0     0     0     0  Loopback0
```

Figure 8: `scli` monitoring interface statistics

`scli` can also be used to do ad-hoc monitoring in order to track down problems. The monitoring display has the same look and feel as the well known Unix `top` command. Figure 8 shows `scli` displaying interface statistics. Some cells in the table are dashed out since no values are available due to the interface layering and the interface types.

3.5 MRTG → RRDTool

The Multi Router Traffic Grapher (MRTG[7]) was originally written to monitor and visualize traffic statistics on a relatively slow and constantly overloaded Internet link [22]. Version 1.0 of MRTG, which was released in 1995, was entirely written in Perl and used several external programs to collect data and to create graphics. MRTG quickly became popular since the generated graphics turned out to be very useful to inform network users and the management people who decide about the money for upgrading networks. MRTG was very user friendly – every standard Web-browser was able to retrieve and display the router statistics.

However, the first version of MRTG did not scale very well and the dependencies on other tools such as the CMU `snmpget` command made it difficult to install. These two problems were solved by rewriting critical parts in C and by using an SNMP extension for Perl which was entirely written in Perl and thus very portable. The result of these improvements were released as MRTG 2.0 in

[7] http://people.ee.ethz.ch/~oetiker/webtools/mrtg/

1997. This version also included a program called `cfgmaker` to generate skeleton configuation files for those users who did not want to dive into the SNMP details.

MRTG 2.0 became one of the most widely used SNMP-based monitoring systems. And while it was significantly faster than MRTG 1.0, some people tried to run it in environments that again hit performance limits or they were trying to use MRTG for data sources it was not really designed to handle. This lead to the development of RRDTool, a system to efficiently update and consolidate round robin databases (RRDs) for time-series data. Scripting language interfaces for Perl or Tcl can be used to update RRDs efficiently from scripts.

MRTG 3.0 will be based on RRDTool. Since RRDTool is now very efficient, the data collection itself becomes the critical part. MRTG 3.0 will therefore provide improvements in the SNMP data gathering by allowing multiple simultaneous requests to reduce overall latency.

4 Lessons Learned

Looking back at the evolution of some major open source SNMP tools, it is possible to identify some lessons that could be learned.

4.1 Scripting

There is an important need for scripting capabilities since the invention of SNMP. Scripting is not only important on the manager side, but also on the agent side and sometimes on the tools side as well. This is well reflected history of SNMP tools. There are SNMP APIs available for all major scripting languages.

However, one of the question this paper raises is whether the scripting APIs should be focussed on the SNMP protocol or the information defined in MIB modules, which immediately leads to the next lesson learned.

4.2 Generic vs. Specific Tools

SNMP is a pretty generic protocol. The basic SNMP operations to `read` and `write` variables and to `notify` applications about the occurance of events are the basic primitives needed to manage devices and networks[8]. The specifics of how to use SNMP for solving certain management problems are contained in the MIB definitions. And since SNMP was designed with a small set of protocol operations in mind, it is not surprising that MIBs sometimes tend to be complex in order to map complex management interactions to the protocol operations.

[8] SNMP is therefore sometimes called the ‘turing machine for network management’.

The fact that SNMP protocol operations are rather simplistic obviously motivates people to design tools that focus more on the SNMP protocol itself than on the semantics of the data structures described in MIBs. In other words, many existing tools try to be generic instead of doing something very specific. Typical examples are so called MIB browsers which help users to browse and manipulate raw MIB data. Such tools usually only understand the machine parseable parts of a MIB module - which is just good enough to shield users from the many arcane numbers used in the protocol. Other examples are scripting language APIs which basically just provide a programmer friendly view on the SNMP protocol. However, in order to realize more useful management application, it is necessary to understand the semantics of and the relationships between MIB variables. Generic tools require that the users have this knowledge - which is however not always the case.

Specific tools focus on implementing a certain specific function and they contain the knowledge about the semantics of the MIB variables which must be manipulated to realize the function. Users of specific tools therefore do not need to understand MIBs in order to use the functions realized by a specific tool. Only the people who develop and debug specific tools need to be aware of the MIB details. Of course, it requires some efforts to obtain the MIB knowledge and to design useful management functions for a certain subject domain. Usually some experimentation is required to identify which of several alternatives works best.

The `scli` is an attempt to create the infrastructure for a specific and extensible SNMP tool. The software has been designed to allow programmers to focus on the management functions rather than the underlying SNMP operations. However, in order to make `scli` even more useful, many more specific commands should be added.

4.3 First vs. Second Generation Tools

Most open source implementations are second generation tools these days where almost all of the original code has been rewritten. This shows that software requires redesign as the technologies around it evolve. The original CMU package is a good example. Several branches were created from the original CMU release. Many of them died out over time and only those branches remained which got enough attention to actually implement necessary redesigns.

The `NET-SNMP` project is currently working on release 5.0 which will again include major changes to keep the package maintainable in the future. Another good example is the MRTG tool which started as a simple Perl script and which turned over time into an efficient tool to maintain time-series data plus a frontend for collecting statistics via SNMP.

Other open source SNMP tools were less successful. The `mosy` MIB compiler

is an example of a tool which turned out to be too difficult to maintain/evolve. It was cheaper to design and implement a new MIB compiler from scratch which incorporates among many other new things the functionality of `mosy`. Implementations which are easy to understand and which allow for redesigns might have a longer lifetime than implementations which are perhaps more powerful but difficult to adapt over time.

4.4 Software Quality

The quality of the original SNMP implementations was not very high considering today's expectations on software stability. The stability generally improved with the movement from K&R C to ANSI C and with the availability of C compilers which produce helpful warnings. However, some problems of the original implementations are still present. In February 2002, the Computer Emergency Response Team (CERT) issued an advisory because many SNMP implementations have been found to suffer from buffer overflow problems in the ASN.1/BER decoders/encoders. Of course, most of these problems can be fixed easily - but widespread deployment of these fixes will take time.

Another problem which is still present are invalid MIB modules which people have published over time. Commercial SNMP toolkit and application vendors still prefer to accept invalid MIB modules rather than doing strict syntax and semantic checks. This is motivated by the need of end users to work with the many invalid MIB modules they have collected over time. The unfortunate consequence of this is that MIB authors who do not use validating MIB parsers can go a long way without even noticing that the MIB module they have produced is invalid. Things have improved somewhat in recent years with the availability of `smilint` — but it will take many more years until the majority of MIB modules will have gotton fixed.

5 Conclusions

This paper provided an overview over the evolution of some major open source SNMP packages. There are of course many more open source SNMP package available than could be reasonably handled in this paper and the selection is certainly subjective. For example, this paper completely ignores SNMP extensions for languages such as Java or more exotic languages such as Erlang [23].

Most of the open source SNMP packages are already second generation tools where major or even all parts of the original code base have been redesigned and rewritten. The overall software quality has become more important and recent CERT advisories will force even more programmers to pay attention to detect and

handle malicious SNMP messages. This will lead to tools that do perform better data validation before data is passed to any critical devices or applications.

The paper also discussed the need for specific rather than generic open source SNMP tools which do not require that users have knowledge about the semantics and relationships between MIB variables.The SNMP command line interface `scli` is an attempt to create such a specific tool.

Acknowledgements

The author likes to thank Wes Hardaker and Tobias Oetiker for their useful comments on a draft version of this paper. Frank Straß provided feedback which improved the presentation.

References

[1] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction to Version 3 of the Internet-standard Network Management Framework. RFC 2570, SNMP Research, TIS Labs at Network Associates, Ericsson, Cisco Systems, April 1999.

[2] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol. RFC 1157, SNMP Research, PSI, MIT, May 1990.

[3] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[4] M. Rose and K. McCloghrie. Concise MIB Definitions. RFC 1212, Performance Systems International, Hughes LAN Systems, March 1991.

[5] M. Rose. A Convention for Defining Traps for use with the SNMP. RFC 1215, Performance Systems International, March 1991.

[6] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Structure of Management Information Version 2 (SMIv2). RFC 2578, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, April 1999.

[7] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Textual Conventions for SMIv2. RFC 2579, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, April 1999.

[8] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Conformance Statements for SMIv2. RFC 2580, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, SNMP Research, International Network Services, April 1999.

[9] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing SNMP Management Frameworks. RFC 2571, Cabletron Systems, BMC Software, IBM T. J. Watson Research, April 1999.

[10] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, Nortel Networks, Intel, Cisco, IPHighway, PFN, Allegro Networks, March 2001.

[11] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. Structure of Policy Provisioning Information (SPPI). RFC 3159, Cisco Systems, Nortel Networks, Intel, Allegro Networks, PFN, August 2001.

[12] J. Schönwälder and F. Strauß. Next Generation Structure of Management Information for the Internet. In *Proc. 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, pages 93–106. Springer Verlag, October 1999.

[13] J. Schönwälder and A. Müller. Reverse Engineering Internet MIBs. In *Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, May 2001.

[14] W. Yeong. SNMP Query Language. Technical Report 90-03-31-1, Performance Systems International, March 1990.

[15] A. V. Aho, B. W. Kernighan, and P. J. Weinberger. *The AWK Programming Language*. Addison Wesley, 1988.

[16] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, April 1994.

[17] J. Schönwälder and H. Langendörfer. Tcl Extensions for Network Management Applications. In *Proc. 3rd Tcl/Tk Workshop*, pages 279–288, Toronto, July 1995.

[18] J. Schönwälder. Married with Tcl. In *Proc. 1st European Tcl/Tk User Meeting*, Hamburg, June 2000.

[19] M. T. Rose and K. McCloghrie. *How to Manage Your Network Using SNMP*. Prentice Hall, 1995.

[20] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl*. O'Reilly, 3 edition, July 2000.

[21] J Schönwälder. Specific Simple Network Management Tools. In *Proc. LISA 2001*, December 2001.

[22] T. Oetiker. MRTG - Multi Router Traffic Grapher. In *Proc. 12th Conference on Large Installation System Administration (LISA XII)*, Boston, December 1998.

[23] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in Erlang*. Prentice Hall, 2 edition, 1996.