The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS
VOLUME 9, NUMBER 1 DECEMBER, 2001

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol. In each issue, *The Simple Times* presents technical articles and featured columns, along with a standards summary and a list of Internet resources. In addition, some issues contain summaries of recent publications and upcoming events.

In this Issue:

The Simple Times is available as an online journal in HTML, PDF and PostScript. New issues are announced via an electronic mailing list. For information on subscriptions, see page 21.

Editorial

Aiko Pras, University of Twente
Jürgen Schönwälder, University of Osnabrück

The year 2001 has been one of the more silent years in the SNMP history. But sometimes silence is a good thing, especially if it is not caused by a lack of activity. Quite some work has been done in 2001 to progress core technology documents and to complete or revise several MIB modules.

Work on SNMPv3 is reaching completion as the specifications are currently progressing through the IESG approval process to become a full Internet Standard. The publication of the SNMPv3 specifications as Internet Standards is expected to happen in 2002 and it might go hand in hand with an action to classify SNMPv1 as Historic. Of course, such an IESG action does not immediately impact all the widely deployed SNMPv1 implementations. SNMPv1 implementations will stay with us for many more years. But still, classifying SNMPv1 Historic sets a clear signal that SNMPv3 has become the stable SNMP version of the future and people are safe in deploying SNMPv3 implementations in large scales.

The IETF standard for extensible agents (RFC 2741, RFC 2742) has been elevated to Draft Standard in December 2001. The AgentX specifications are some of the few documents that manage to progress in the IETF standardization process without any changes to the RFCs. This is even more impressive since the working group collected 13 implementation and interoperability reports.

The MIB module for Differentiated Services, another milestone in the set of IETF MIBs, has been approved for publication as Proposed Standard in November 2001. Work on this MIB involved the cooperation of several subject matter experts since the automated configuration of routers supporting the differentiated services architecture is frequently used to demonstrate policy-based configuration management schemes.

It is good to see that the core SNMP technology has reached stability again after a lengthy period of several competing versions in the 1990s. The SMIv2 data definition language is a full Internet Standard since 1999 and it is expected that the SNMPv3 protocol is published as a full Internet Standard in 2002. This gives the industry a stable technology to create new products and network operators a stable base for their future deployment decisions. On the other hand, every successful living technology also needs a controlled evolutionary path forward to address new requirements and to adapt to changes in the environment. It is thus not surprising that the IETF has started two new working groups:

- The *Evolution of SNMP* working group (EOS) focuses on the evolution of the protocol operations.
- The *Next Generation Structure of Management Information* working group (SMIng) focuses on the evolution of the data definition language.

The SMIng working group has produced a document which discusses the objectives of the SMIng work. This document was published as RFC 3216 in December.

SNMP Set: Can it be saved?

Andy Bierman, Cisco Systems

There are many factors preventing the SNMP command responder application interface from displacing the command line interface (CLI) as the primary configuration mechanism for network devices. Some of the issues have been raised on the SNMP WG mailing list in the past, but were largely ignored at the time. Not all of the issues are directly related to SNMP or the SMI, and not surprisingly, the most important factor is money. SNMP agent code costs too much to develop, test, and maintain, compared to the equivalent CLI code. The complex SNMP `set` state machine and the overhead of lexicographic sorting are the worst problem areas, usually much more (five to ten times?) expensive to develop than CLI code. In order for SNMP to succeed as a configuration management interface, development costs must be competitive with the CLI, while offering significantly more value to developers than the CLI. This should be possible, since the CLI is primarily a human interface and does not provide many important features of a programmatic interface, such as stability, detailed command semantics, or compliance and conformance information.

The problem starts with the API definition itself – the SMIv2 information module. CLI specifications are easier to define than MIBs because there are not nearly as many documentation requirements and modification rules. SMIv2 syntax is not straightforward, widely understood, or easy to learn. Furthermore, SNMP has no real transaction semantics, which means input and output parameters to commands are modelled as data elements (i.e. MIB objects).

So why are SNMP configuration objects so expensive to develop compared to the equivalent CLI configuration commands? Simply put, the CLI accepts input at an appropriate granularity (one command at a time) while SNMP accepts input at an inappropriate granularity (one parameter at a time). This applies to row creation as well as row modification.

SNMP `set` PDUs may contain an arbitrary number of (potentially unrelated) arbitrarily partial commands, and the agent is expected to accept individual parameters, not complete commands. The partial "commands-in-progress" have to be processed "best-effort", as if an entire set of parameters existed, and then saved as MIB objects, so these parameters can be retrieved by an NMS in subsequent read operations.

The agent cannot simply store partial commands until they are complete, because some parameters have "act now" semantics and others have "act on activation" semantics (e.g. `bufferControlMaxOctetsRequested` from the RMON-MIB (RFC 2819) is an "act-now" parameter while `bufferControlChannelIndex` is an "act-on-activation" parameter, both defined in the same table).

The requirement to accept partial input is most complicated if some parameters are inter-related, which is usually the case. The `set` PDU logic for such MIB objects can be quite complicated, as well as the selection of appropriate default values for missing parameters. For example, if an agent supporting the `DISMAN-PING-MIB` (RFC 2925) received a `set` PDU containing only one varbind for a `pingCtlTargetAddressType` object (syntax `InetAddressType`) set to `dns(16)`, then the agent has to select an arbitrary value for the `pingCtlTargetAddress` object, which is supposed to be a DNS name.

One work-around for this problem is for the MIB designer to place as many essential parameters as possible in the `INDEX` clause. This is problematic from a MIB design perspective, since `OBJECT IDENTIFIER`s are limited to 128 sub-identifiers and the `INDEX` clause implies uniqueness across all its parameters. From an agent implementation perspective, this technique simply trades `set` PDU complexity for lexicographic sorting complexity. For an management system developer, editing an existing row becomes more complex because parameters in the `INDEX` clause cannot simply be changed, but rather the entire row must be deleted and re-created with a new index value. This can sometimes cause disruptive behavior on the device as well.

The CLI allows only one command at a time to be input, instead of an arbitrary mix of commands. If essential parameters are missing, the entire input is rejected as a syntax error, rather than accepted as partial input. These two simple restrictions make the development and testing of CLI code relatively trivial, compared to SNMP.

CLI does not need the `createAndWait` state, because it uses a reliable stream-oriented transport (e.g. TCP) and it has a "line continuation" mechanism. Fragments from the same command are simply cached by the CLI parser until the final fragment is received, and then the entire command is parsed, as if it was received all at once. The CLI has the robust `createAndGo` mechanism that SNMP needs.

Even if SNMP development costs are lowered, there is an unrelated factor which hinders any potential CLI replacement. Network operators rely on the configuration file as a concise, readable representation of all settings for a device, and therefore consider it to be the "native language" representation of the device itself. These files are usually formatted in plain ASCII so they can be examined and edited with even the most basic tools. In some cases, rather complex tools are used to generate and manage device configurations automatically. In many products, the non-volatile (and network) storage of device configuration data is in "CLI representation", which implies that only settings which are supported by the CLI interface can be restored after a reboot, or archived to an external storage device.

But the SNMP `set` can be "fixed". If the agent accepted input in a manner that is consistent with the CLI, then agent development costs could be dramatically lowered, and much more instrumentation code could be shared between SNMP and CLI access mechanisms.

First, MIB designers need to keep writable objects simple and aligned with CLI access granularity by only defining objects with "act-on-activation" instead of "act-now" behavior. The practice of designing SNMP command responders to accept any arbitrarily partial input must end. It must be acceptable for an agent to reject otherwise valid SNMP `set` PDUs because too many different commands are combined in the same PDU, or not enough parameters for a particular command are contained in the same PDU. The complex `RowStatus` states (i.e. `createAndWait` and `notInService`) need to be deprecated, and no longer used in new standard MIBs.

Then SNMP can be aligned with the CLI with some relatively minor enhancements to the SMI and SNMP. New standards under development in the SMIng and EOS working groups can provide enough new features to simplify the SNMP `set` implementation requirements. The new data definition language from the SMIng WG can provide enough machine-parseable semantics to convey the expected parameter granularity for basic functions, such as class creation, modification, and deletion. The new row-based (should really be class-based) operations from the EOS WG can potentially provide a mechanism in the protocol to allow an SNMP engine to reject partial input to the command responder application in a generic manner. Input could also be generically presented to the command responder in groups instead of individual elements. This can simplify the parameter validation logic and allow for better agent code re-use and more sophisticated automatic code generation tools.

The complex and nested containment provided by SMIng classes, combined with class-based operations and "RowStatus Lite" from EOS, could allow SNMP tool and engine developers to greatly simplify the agent software development environment. Lower development costs will increase the likelihood that new devices will ship with an SNMP interface in addition to the (default) CLI interface.

Discovery of Spanning Trees in Virtual Bridged LANs

Meng Guo, Georgia Institute of Technology
Subrata Mazumdar, Avaya Labs

Virtual Local Area Network (VLAN) capabilities are nowadays an integral feature of switched LAN solutions provided by LAN equipment vendors [1, 2]. VLANs facilitate easy administration of logical groups of stations on different LAN segments as if they were on the same LAN segment. A virtual bridged LAN [3, 4] consists of one or more VLAN-aware bridges and allows the definition, creation and maintenance of the VLANs. All the bridges within a bridged LAN environment participate in a spanning tree over which multiple VLANs can coexist. One of the challenges of VLAN deployment is the degree to which VLAN configuration is automated. The first step towards automatic VLAN configuration is the discovery of the initial configuration of the bridges, the spanning tree operated over the bridges and the mapping from the spanning tree to the VLAN topology. There can be multiple instances of spanning trees over a set of bridges. The operation of the spanning tree is transparent to bridges as well as external management applications, and the spanning tree information is distributed over all the bridges in the bridged LAN. Reconstruction of the spanning tree and discovery of the mapping between the spanning tree and the VLAN is central to the management of bridged LANs.

Many bridge vendors provide VLAN management tools, but these tools usually only work for their own devices since they make use of vendor specific MIB modules. In some cases, the bridges do not even fully implement the IETF `BRIDGE-MIB` (RFC 1493). In addition, some vendor supplied tools are end-user applications that are to be used only by human network administrators. No programming APIs or information models and class libraries are supplied in these tools for new application development. Even when APIs are supplied [5], these APIs tend to be device centric and they are not based on logical entities such as VLANs or spanning trees. Furthermore, generic SNMP-based management platforms are not capable of discovering the relationship between the MIB objects because that would require customized processing of the MIB variables related to the BRIDGE-MIB.

The main goal of our spanning tree discovery project is to create a vendor-neutral tool to discover the bridges and spanning trees using only SNMP and standard MIB modules, such as RFC1213-MIB (RFC 1213), BRIDGE-MIB (RFC 1493), and Q-BRIDGE-MIB (RFC 2674). The scope of our bridge discovery is limited to bridges that support the BRIDGE-MIB. As a result, our discovery excludes other layer two devices such as hubs or bridges that do not support the BRIDGE-MIB.

Since different vendors have different implementations of MIB modules [2], we have created a vendor-neutral information model for the bridged LAN based on the standard MIB modules, and at the same time, provide a mediation layer framework and tools for mapping between the vendor specific MIB modules and the standard MIB modules. In addition to the discovery of bridges, VLANs and the associated spanning trees (STs), we wanted to build a set of reusable Java class libraries with a well defined information model and interfaces for logical entities that span multiple devices in a bridged LAN. We need a better information model than device centric MIB modules for bridges because relationships between bridges are maintained implicitly through bridge addresses used by the spanning tree protocol (STP). Our discovery tool is the first step in our implementation of Java class libraries that will establish the relationship between bridges and provide navigation capabilities from one node of the spanning tree to another. In the future, we will extend the class libraries with capabilities for adding, changing, or removing station membership.

The key contribution of this article is a description how STP bridge addresses can be associated with IP addresses and an algorithm to build the ST by querying standard MIB modules for bridges. If the standard MIB modules related to bridges are not supported, then we provide a mapping between vendor specific MIB modules and standard MIB modules for the bridges.

Virtual Bridged LANs: Concepts and Definition

A virtual bridged LAN [3, 4, 6, 7] consists of one or more VLAN-aware bridges and allows the definition, creation and maintenance of Virtual Local Area Networks (VLANs). A VLAN is a proper subset of the active topology of a bridged local area network. Each VLAN is associated with a VLAN identifier (VID). When packets from two different VLANs transit a common link, a tag is prepended to the frame that contains the VID and the priority (tagged frames). On links that have only one VLAN, tagging is optional. VLAN-aware bridges or end stations recognize and support VLAN-tagged frames. A VLAN can be implemented in any bridged LAN environment that supports IEEE 802 LAN MAC protocols [3] and over shared media LANs as well as point-to-point LANs. The bridges forward unicast, multicast, and broadcast traffic only on LAN segments that serve the VLAN to which the traffic belongs. Typically, a router is used to move traffic from one VLAN to another. All bridges within a bridged LAN environment participate in a spanning tree over which multiple VLANs can coexist [3, 4, 8]. The primary reason for running a spanning tree is to eliminate loops in a bridged infrastructure and to provide redundant paths, which can be unblocked upon failure. Although the VLAN standard [4] specifies a single spanning tree for all VLANs, there is a choice to be made as to how many spanning trees operate in a VLAN environment and how the VLANs in that environment map to those spanning trees. Vendor dependent implementations vary in whether there is one spanning tree for each VLAN or whether multiple VLANs map to a spanning tree.

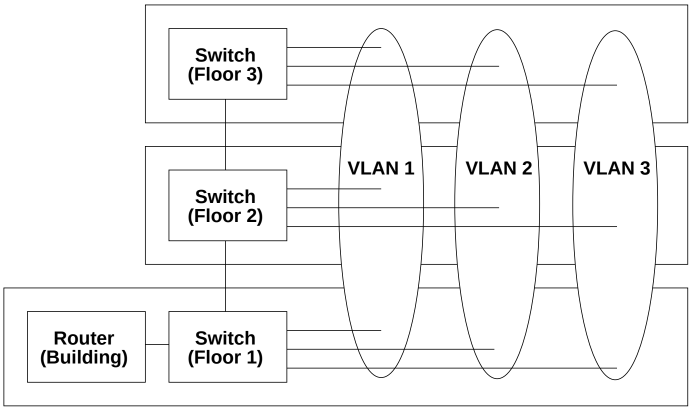

Figure 1: Typical VLAN configuration.

A virtual LAN allows a group of stations to communicate as if they were on the same physical LAN segment. The stations could be connected to different ports of the same switch or to ports on different bridges. A VLAN allows a network manager to logically segment a LAN into different broadcast domains (see Figure 1). Since this is a logical segmentation and not a physical one, workstations do not have to be physically located together. Users on different floors of the same building, or even in different buildings can now belong to the same logical LAN. VLANs can be spanning across different physical network environments, such as ATM, Ethernet, Token Ring or FDDI networks.

There are several criteria by which VLAN membership can be defined [2, 4]:

- By port: Each port on a bridge belongs to one VLAN. All traffic within the VLAN is switched, and traffic between VLANs is routed (by an external router or by a router within the switch). This type of VLAN is also known as a segment-based VLAN. Figure 1 illustrates a set of VLANs based on port-based grouping where one port from each bridge belongs to a different VLAN. The IP packets between the VLANs are exchanged through the router. This is the membership criterion supported by the IEEE/IETF MIB modules.

- By MAC address: The traffic from a specific end station is put into a specific VLAN based on the MAC address of the network interface card in the end station.

- By network address: VLANs based on network addresses can differentiate between different layer-3 protocols, allowing the definition of VLANs to be made on a per-protocol basis. Routing between VLANs comes automatically, without the need for an external router or card. Network address-based VLANs will mean that a single port on a switch can support more than one VLAN. This type of VLAN is also known as a virtual subnet VLAN.

- By IP multicast groups: All workstations that join an IP multicast group can be seen as members of the same virtual LAN. However, they are only members of a particular multicast group for a certain period of time.

In addition to the above criteria, VLANs may be extended beyond a single bridge through the use of trunking between bridges. A trunk is a point-to-point link carrying the traffic of several VLANs. To preserve VLAN information across the trunk, the Ethernet frames are prepended with 802.1Q tags.

Related Work

There has been extensive work done on the discovery of VLAN configuration. Most bridge vendors provide their own tools for VLAN configuration and path discovery between end points. These tools primarily depend on proprietary discovery mechanisms for bridges.

- Avaya's VLANMaster application in the CajunView Suite [9] for network management is a powerful, easy-to-use tool that helps simplify all areas of VLAN management - from initial assignment to moves and changes. CajunView will automatically maintain an up-to-date view of VLANs. VLANMaster only works with Avayas Cajun switches.

- Aprisma's SPECTRUM VLAN Manager [10] allows users to configure and manage their 802.1Q switches, ports, and VLANs from a single workstation on the network using a graphical user interface. It uses the Cabletron Discovery Protocol (CDP) to find all CDP compatible 802.1Q switches in a domain by simply providing the SPECTRUM VLAN Manager with the IP address of one of the switches in the domain. It also allows a network administrator to create, configure, and delete 802.1Q VLANs in a domain. Once a VLAN has been created, the administrator can easily assign it to a switch or switch port.

- Cisco's VlanDirector [11], which is coupled with CiscoView, provides VLAN discovery as well as functions for creation and management of VLANs. VlanDirector uses the Cisco Discovery Protocol (CDP) to discover the physical connectivity of the devices in the network. VlanDirector cannot manage any devices that do not run CDP.

- 3COM's Enterprise VLAN Manager [12] is used to monitor, control and automate the network by discovering and displaying physical and logical switched topologies and presenting views through graphical user interfaces.

- Granite [5] is an open source C API/SDK to provision VLAN configurations using SNMP. The API supports standard MIB modules for bridges. In addition, the API can be used to manage layer two filters and layer three access control lists on Riverstone products.

Our work is closely aligned with Aprisma's SPECTRUM VLAN Manager and the Granite API, which are based on IETF standard MIB modules for bridges. In addition, we provide a vendor neutral model based on standard MIB modules. We also provide a framework for mediating vendor specific MIB modules to standard MIB modules for bridges.

Discovery of Spanning Trees and VLANs

In this section, we propose our method of spanning tree and VLAN discovery in the virtual bridged LAN based on the IETF standard MIB modules for bridges. MIB modules that are used in this article are the `RFC1213-MIB` (RFC 1213), the `BRIDGE-MIB` (RFC 1493) and the `Q-BRIDGE-MIB` (RFC 2674). The discovery method has the following steps:

- Deduce the target Virtual Bridged LAN from an arbitrary IP address in the Virtual Bridged LAN;
- Automatically discover all the bridges in the target Virtual Bridged LAN;
- Collect spanning tree and VLAN related MIB variables from the discovered bridges using SNMP;
- Construct the spanning trees in the Virtual Bridged LAN and associate the spanning trees with VLANs.

According to the specification of Virtual Bridged LANs [4], there is only one spanning tree for all the VLANs and the object models for the `BRIDGE-MIB` and `Q-BRIDGE-MIB` represent that recommendation. In reality, many switch vendors support more than one spanning tree. Our object model for spanning tree information represents the most general case, one spanning tree per VLAN.

Our discovery method makes certain assumption about the target bridged LAN and the MIB modules supported in the bridges. We have used the SNMPv1 protocol to access MIB variables because the bridges in our test network only support SNMPv1. We have used a single community string for accessing all the bridges in the target LAN. We have used the MIB variables from `RFC1213-MIB` because of our familiarity with this MIB module. A concise description of mapping between `RFC1213-MIB` variables and the SMIv2 counterparts (RFC 1907, RFC 2011, RFC 2096, RFC 2863) can be found in [13].

Deduction of Target Virtual Bridged LAN

In order to constrain the domain of the bridges in a LAN environment, we select a bridged LAN that is bordered by routers as the target bridged LAN. The starting point of the whole VLAN discovery process is an IP address of one of the hosts, routers or switches of the target bridged LAN. We need to determine the IP address ranges (possibly a collection of subnets) of the target bridged LANs before the bridge discovery step. If the initial IP address does not represent a router, we first need to obtain the next-hop router of the given IP address by retrieving the `ipRouteNextHop` variable of the `ipRouteTable`. If there is more than one router address, then we pick the address that is in the same subnet as the given IP address.

Once we have an IP address of a router, we can find the associated interface index by examining `ipAdEntIfIndex` and we obtain all the subnets associated with the router. For each entry in the `ipRouteTable`, we obtain the `ipRouteDest` and `ipRouteMask` objects that match the interface index and compute the IP subnet address range using the combined value of `ipRouteDest` and `ipRouteMask`. For example, if the starting IP address is 135.8.29.11 and the next-hop router is 135.8.28.1, we check the `ipRouteTable` in the next-hop router. There, we will find an entry with `ipRouteDest = 135.8.12.0` and a corresponding `ipRouteMask = 255.255.252.0` and the IP address range will be 135.8.12.1-135.8.16.255. Thus, the span of our bridge discovery is the union of all the subnet ranges computed for the interface index associated with router's IP address.

Auto Discovery of Bridges

The aim of automatic discovery of all the bridges is to create a one-to-one mapping between the IP address and MAC layer bridge address of a bridge in the target LAN environment. The bridge address is the lowest MAC address of the bridge used by the bridge for running the spanning tree protocol. The IP address of the bridge is needed because SNMP messages are sent to the IP address of the bridge in order to get MIB information from the bridge. On the other hand, the spanning tree protocol is a MAC layer protocol and the information about the parent nodes of the spanning tree is kept in the form of bridge addresses. The only way the information in a specific bridge can be correlated with the information in its parent bridge (in the spanning tree) is by mapping the bridge MAC address to an IP address. The discovery of the bridges is accomplished in two steps. First, we query the `sysDescr` object to determine if the device on an IP address is running an SNMP agent. Second, we query the `dot1dBridgeAddress` object to see if the device is a bridge. The detailed operations are described as below:

1. Given the computed IP address range from the seed IP address, we send an SNMP `get` message to retrieve the `sysDescr` variable for each IP address in the address range. A timeout or an error response indicates that the IP address is either not reachable or that SNMP access is not enabled.
2. If the device on an IP address is running an SNMP agent, then we send a second SNMP `get` message to retrieve the scalar `dot1dBaseBridgeAddress`. If the SNMP agent supports the `BRIDGE-MIB`, then the returned value is the address of the bridge. A `noSuchName` error or `noSuchObject` exception indicates that this device is not a bridge.

If the above two steps are successful, we conclude that the device with this IP address is a bridge. To discover all the bridges which support the BRIDGE-MIB on the specified IP range, we run the above two steps for every IP address on the computed IP address ranges and then compile a mapping between the bridge IP addresses and corresponding bridge MAC addresses.

Collection of Spanning Tree Information

Automatic discovery of bridges generates a list of bridges that support the STP. The STP is a distributed protocol and each switch only maintains local STP information. The STP related information is stored in the dot1dStpPortTable table of the BRIDGE-MIB as shown below. For constructing the spanning tree, we are only interested in a subset of the MIB variables of the dot1dStpPortTable (marked below).

```
dot1dStpPortEntry OBJECT-TYPE
  SYNTAX Dot1dStpPortEntry
  ACCESS not-accessible
  STATUS mandatory
  DESCRIPTION
      "A list of information maintained by every
       port about the Spanning Tree Protocol
       state for that port."
  INDEX  { dot1dStpPort }
  ::= { dot1dStpPortTable 1 }

Dot1dStpPortEntry ::= SEQUENCE {
  dot1dStpPort                    INTEGER, --
  dot1dStpPortPriority            INTEGER,
  dot1dStpPortState               INTEGER, --
  dot1dStpPortEnable              INTEGER,
  dot1dStpPortPathCost            INTEGER,
  dot1dStpPortDesignatedRoot      BridgeId, --
  dot1dStpPortDesignatedCost      INTEGER,
  dot1dStpPortDesignatedBridge    BridgeId, --
  dot1dStpPortDesignatedPort      OCTET STRING, --
  dot1dStpPortForwardTransitions Counter
}
```

For each discovered bridge, we send one SNMP getnext message for each row of the dot1dStpPortTable to get the values of the marked variables. As we walk through the table row by row, we check the dot1dStpPortState variable of each row and discard entries whose port state is broken. We also extract the VLANs associated with the selected ports by retrieving dot1qPvid from the associated entry of the dot1qPortVlanTable of the Q-BRIDGE-MIB. Given all this information, we create one row of our internal table for each combination of VLAN identifier and port identifier. The columns of the internal table are shown in Table 3. The knowledge of the bridge address, the port identifier, the VLAN identifier, the designated root, the designated bridge, and the designated port are necessary and sufficient for constructing the per-VLAN spanning tree.

Per-VLAN Spanning Tree Construction

Once the internal spanning tree information table is built, we sort it by the VLAN identifier and the bridge address columns so that spanning tree information for each VLAN is in contiguous rows and ordered by the bridge address. Then for each group of entries for each VLAN identifier, we perform the following steps to construct the per-VLAN spanning tree: Given a VLAN identifier, if the VLAN identifier of an entry in the table equals to the given value, compare the bridge address and designated bridge. If both the bridges addresses are the same, then skip this entry; if they are different, perform the following operations:

1. Select the first entry from this VLANs group of entries as the current entry.
2. If the bridge address and the designated bridge address of the current entry are the same, then go to step 3; otherwise perform the following steps:
   - Set this bridge address as the child of the designated bridge address of the spanning tree.
   - Mark the port id of this entry as the egress port on this bridge, and the designated bridge port id as the ingress port of the designated bridge.
3. Select the next unvisited entry as the current entry and go to step 2; The spanning tree is constructed when all the entries of this VLAN identifier have been visited.

Complexity Analysis

The analysis of the complexity of our solution is composed of two aspects: traffic load and spanning tree computation. In our four-phase solution, the first three phases send SNMP packets to collect necessary information. The fourth phase is a pure local processing step which injects no traffic in the network.

In the automatic bridge discovery phase, assume there are n IP addresses, and there are m bridges that support per-VLAN spanning tree in the target LAN environment. We send at most (2n) SNMP packets to the network in order to find m bridges among n IP addresses. Note that the two get messages can be combined into a single message which reduces the number of required SNMP packets to n.

In the STP information collection phase, we assume that there are at most l entries in the dot1dStpPortTable for each bridge, and there are at most p entries in the dot1qPortVlanTable. Since we need to collect only 5

BRIDGE-MIB/Q-BRIDGE-MIB	PROMINET-MIB
dot1dStpPort	promBridgePortIndex
dot1qPvid	promBridgePortIndex, promVlanID
dot1dStpPortState	promBridgePortState
dot1dStpPortDesignatedRoot	promBridgePortDesignatedRoot
dot1dStpPortDesignatedBridge	promBridgePortDesignatedBridge
dot1dStpPortDesignatedPort	promBridgePortDesignatedPort

Table 1: Mapping between proprietary and standard MIB variables.

columns of the `dot1dStpPortTable`, we can send one SNMP message for each port to get the values of the variables. We have to get only one variable from the `dot1qPortVlanTable`. Thus, we have to send out at most (l + p) SNMP packets in order to collect spanning tree information from each bridge.

Putting things together, the VLAN and spanning tree discovery process needs to send out 2n+m(l+p) SNMP packets to the network or n+m(l+p) if the first two messages are combined into a single message. During these initial steps, the local processing cost of storing the data into the table is trivial compared to the sending and receiving SNMP messages. For example, in our test environment the values of n, m, l and p are as follows: n=512, m=11, l=48, p=7. Thus, we have to send about 1629 SNMP messages or 1117 messages if the first two messages are combined. The above analysis is based on SNMPv1. If we use SNMPv2c or SNMPv3, which support `getbulk` operations, we may be able to retrieve the tables with fewer messages by packing more than one row in SNMP responses.

As to the complexity of constructing the spanning tree, our method is actually a scan of the table for each VLAN identifier and then it builds the tree from bottom up, i.e., from leaf to the root. This algorithm can be finished in linear time.

Experimentation and Analysis of Results

In order to test our spanning tree discovery method, we ran a set of tests in our internal corporate network. Our corporate network consists of a large number of Avaya P580 Cajun bridges [14] and routers. We have restricted the tests to a small subnet of the corporate network.

Avaya's Cajun bridges store the STP related information in their proprietary MIB, called `PROMINET-MIB`. The Cajun switches maintain one spanning tree for each VLAN. The `PROMINET-MIB` is an extension of the `BRIDGE-MIB` for representing MIB objects for multiple VLANs. The `PROMINET-MIB` stores spanning tree information in its `promBridgePortTable`. This table has all the objects defined in `dot1dStpPortTable` of the `BRDIGE-MIB`. The `PROMINET-MIB` maintains one spanning tree for each VLAN. The `promBridgePortTable` is indexed by a combination of the VLAN identifier and the port identifier. In the `BRIDGE-MIB` only the port identifiers are used to maintain the spanning tree information.

Table 1 describes the mapping of the variables of `promBridgePortTable` to the standard `BRIDGE-MIB`. The VLAN identifier of a port is extracted based on the VLAN index encoded in the `promBridgePortIndex`. The port identifier and the VLAN identifier are combined to form the `promBridgePortIndex`. The VLAN identifier is deduced in the following way:

- First, we retrieve the `promBridgePortIndex`. The first half of this object is the VLAN bridge index, the second half of this object is the bridge port identifier. We extract the first half of `promBridgePortIndex` to get the VLAN bridge index (`promVlanBridgeIndex`).
- Second, we retrieve all instances of the `promVlanId` and the `promVlanBridgeIndex` from the `vlanTable` in the `PROMINET-MIB` and use this information to map the VLAN bridge index obtained in the first step to the VLAN identifier (`promVlanID`) for the corresponding bridge port.

We have implemented our algorithms using a set of Java based applications. The initial seed IP address we selected was 135.8.12.1 and the IP address is configured on a router. The IP address range of the selected subnet is 135.8.12.0-135.8.15.255.

Bridge Discovery

We have written an application called `BridgePing` that, given an IP subnet, automatically discovers all the bridges in the target address range. Running the `BridgePing` program on all IP subnets computes the mapping shown in Table 2 between IP addresses and MAC layer addresses of the discovered bridges.

Collecting Spanning Tree Related Information

Once we have identified the bridges, we send SNMPv1 message to the bridges to build our internal table (as shown in Table 3) in our Java application for computing the spanning tree:

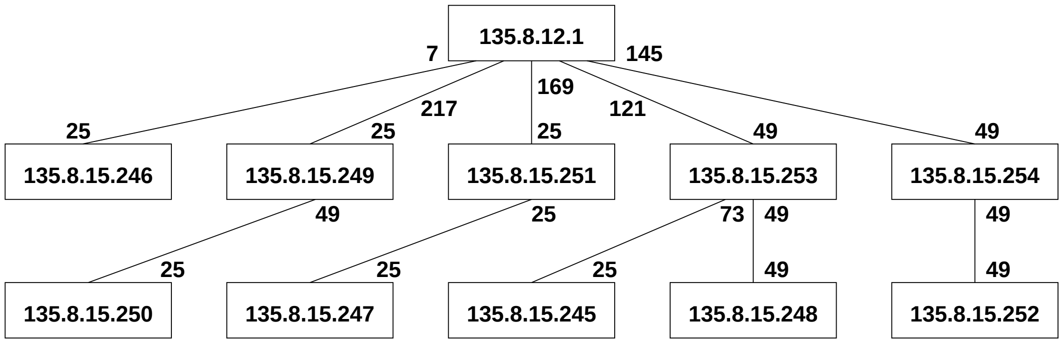

Figure 2: Spanning Tree for VLAN 12 based on Table 3.

- First, we collect all the entries in the `vlanTable` of the `PROMINET-MIB` in order to create a mapping between VLAN identifiers and Cajun switch specific VLAN indexes for the bridge;
- Second, we collect the STP related information from the `promBridgePortTable`. All the variables of a specific row of the table are retrieved in one message. The VLAN index is computed from the `promBridgePortIndex` and mapped to the VLAN identifier using the information collected in the first step. The port identifier is also computed from `promBridgePortIndex`. When all the entries from the `promBridgePortTable` have been collected, we filter out those entries whose state is `broken` or `invalid` and then build the internal table (as shown in Table 3). The bridge address in the table is obtained by mapping the IP addresses to corresponding bridge MAC addresses (using the information in Table 2).

IP Address	Bridge Address
135.8.12.1	00306d3b8800
135.8.15.245	00306d67dc00
135.8.15.246	02e03b0529b3
135.8.15.247	00306d62c000
135.8.15.248	02e03b005e33
135.8.15.249	02e03bfa7800
135.8.15.250	00306d633800
135.8.15.251	02e03bdbf800
135.8.15.252	02e03bfa8000
135.8.15.253	00306d838400
135.8.15.254	00306d63b400

Table 2: Mapping IP to Bridge MAC Addresses.

Construction of Per-VLAN Spanning Tree

In Table 3, each row describes a node of the spanning tree for a specific VLAN. The root of the spanning tree for a VLAN is easily deduced from the Designated Root column. The Designated Root is the same for all nodes of a spanning tree. For a given node (bridge), the parent of the node is obtained from the Designated Bride column. The port identifier of this bridge is marked as egress port and the port identifier of the parent (Designated Bridge Port) is marked as ingress port for the parent node.

Figure 2 shows the result of a spanning tree that is constructed based on the information in Table 3 for the VLAN with identifier 12. The numbers below each bridge node are ingress port identifiers and the numbers above each bridge are egress port identifiers.

Conclusion

In this article, we propose an approach to discover spanning trees in Virtual Bridged LANs using SNMP MIB modules. Our work is composed of three phases. In the first phase, we discover the bridges within a target range of IP addresses and create a one to one mapping between IP and bridge MAC addresses. In the second phase, we collect the STP information from the discovered bridges using SNMP. In the third phase, we compute the spanning tree for each VLAN in a bottom-up manner. We have tested our approach in our internal networks and we verified the result.

The main advantages of our approach are:

- It adds very limited amount of traffic. Our solution obtains the VLAN configuration information using SNMP. The number of packets we send has a quadratic bound.
- Spanning tree construction is a pure local processing step and adds no burden on the designated network. The spanning tree construction algorithm can be finished in linear time.
- The whole process is highly automated. Given a starting IP address, there is no need for human

Bridge Address	Vlan	Port	Port State	Designated Root	Designated Bridge	Designated Port
00306d62c000	10	25	forwarding	00306d3b8800	02e03bdbf800	25
00306d62c000	12	25	forwarding	00306d3b8800	02e03bdbf800	25
00306d62c000	16	25	forwarding	00306d3b8800	02e03bdbf800	25
02e03b005e33	10	49	forwarding	00306d3b8800	00306d838400	49
02e03bfa7800	12	25	forwarding	00306d3b8800	00306d3b8800	217
00306d633800	2	25	forwarding	00306d3b8800	02e03bfa7800	49
02e03b0529b3	12	25	forwarding	00306d3b8800	00306d3b8800	73
00306d67dc00	12	25	forwarding	00306d3b8800	00306d838400	79
02e03b005e33	12	49	forwarding	00306d3b8800	00306d838400	49
00306d838400	12	49	forwarding	00306d3b8800	00306d3b8800	121
00306d633800	12	25	forwarding	00306d3b8800	02e03bfa7800	49
02e03bdbf800	12	49	forwarding	00306d3b8800	00306d3b8800	169
02e03bfa8000	12	49	forwarding	00306d3b8800	00306d63b400	49
00306d63b400	12	25	forwarding	00306d3b8800	00306d3b8800	145
...	...	...	...	...	...	...

Table 3: Internal Table for VLANs and Spanning Trees.

interference for bridge and spanning tree discovery. This makes the approach very easy to use.

We have completed the first step of building a vendor neutral tool for VLAN configuration. Our next step is to extend the discovery tool to implement add/change/remove capabilities for VLAN membership and the configuration of VLANs.

Acknowledgement

We would like to thank Mike MacFaden of Riverstone Networks for his comments that helped us to improve the article significantly.

References

[1] Buerger, D.J., *Virtual LAN cost savings will stay virtual until networking's next era*, Network World, March 1995.

[2] Passmore, D., Freeman, J., The Virtual LAN Technology Report, March, 1997.

[3] IEEE 802.1D, *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Common Specifications–Media access control (MAC) bridges*, 1998.

[4] IEEE 802.1Q, *IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridge Local Area Networks*, 1998.

[5] NMOPS, Granite: A C API/SDK to provision L2 Filter and L3 ACLs and VLANs on Riverstone Routers

[6] Hein, M., Griffiths, D., Berry, O., *Switching Technology in the Local Network: From LAN to Switched LAN to Virtual LAN*, Thompson Computer Press, February 1997.

[7] Subramanian, M., *Network Management, Principles and Practice*, Addison-Wesley, 1995.

[8] Cisco Documentation, Understanding Spanning Tree Protocol

[9] Avaya, Avaya CajunView Enterprise Network Management Systems

[10] Aprisma, SPECTRUM VLAN Manager

[11] Cisco Documentation, VlanDirector - Getting Started Guide

[12] 3COM Documentation, Transcend Management Software Enterprise VLAN Manager User Guide

[13] Riverstone, RS Platform SNMP Management

[14] Avaya, P580 MultiService Switch

Westhawk's SNMP Stack in Java

Tim Panton, Westhawk
Birgit Arkesteijn, Westhawk

In 1995, the authors were working for West Consulting BV on a SNMP agent and manager for a UPS (Uninterruptable Power Supply) maker (Victron, now part of GE Industrial Systems). It was written in 'C' and was ported to 13 different UNIX platforms. In a discussion with the customer it was mentioned that the new Java platform might make the porting/installing of the manager code simpler in the future. The customer remarked that he doubted that Java would be able to do the low level UDP networking or the detailed graphics that were needed.

This was taken as somewhat of a challenge, and so began the SNMP stack in Java described in this article. At the same time, West were offered a chance to present at one of Sun's promotional JavaDays so the first version of the stack was used as a demo at that event. It used almost all the Java features available at the time.

The first version ran on a 16Mb 40Mhz 486 in Netscape's JVM so it needed to be lightweight and simple. West were kind enough to make the stack freeware and it was released in 1996.

The authors later moved to Westhawk Ltd in the UK. Since that time, the SNMP stack has been extended to encompass JavaBeans, SNMPv2c and SNMPv3, and the performance has been improved quite a bit.

Design Decisions

The fundamental design goal was that it should be possible to use the stack in an applet and, other things being equal, leave it running all day. Taking that basic goal a step or two further led to the following core design decisions:

- *No MIB knowledge.* Bandwidth and memory constraints prevented us from reading a full MIB into the applet. Therefore the stack could not contain a MIB browser/parser. Any OIDs would be calculated when creating the code, and then compiled in. Most management applications start off by reading in a whole MIB, parsing it and storing it in memory, and then allow the user to type in descriptors like `sysUpTime`. The applet environment was too restricted for that to work.

- *Abstract PDU base classes.* In order to prevent application code becoming littered with 'magic' OIDs, abstract classes for each of the basic PDU types were created. Application coders are expected to create MIB/domain specific sub-classes which encapsulate some real-world concept (like interface throughput) and request the OIDs needed to calculate that value. This was based on the observation that very few of the displayed values on a manager application are single MIB values - most often it takes a few values combined to create a useful data-point.

- *No byte offsets.* The C code mentioned above was full of byte pointers which were incremented by state machines, which are hard to follow. Java does not support pointers, so a different approach was needed. In essence the ASN.1/BER in a received packet is parsed into a tree of ASN.1 objects and only then are the SNMP rules applied to the resulting tree. Each ASN.1 object class knows how to decode and encode itself, thus keeping the bit-bashing localized to a few classes.

- *Multithreaded.* Since Java supports threads, they were used to try and clarify the code and improve performance. All requests are asynchronous, and use the Observer/Observable mechanism from the `java.util` package.

- *Java Bean.* Later on a JavaBeans layer was added to the package, so that visual programming environments could use the stack. These are also asynchronous, but use the Event/Listener model.

The overall architecture is shown in Figure 1.

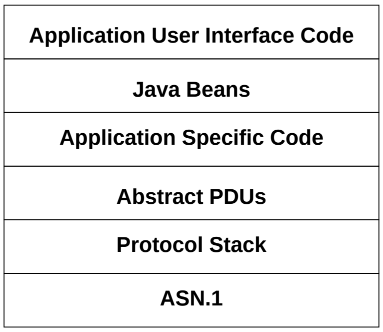

Figure 3: Westhawk's SNMP Architecture.

Threading

To implement a SNMP manager, a timeout/retry mechanism to re-send packets is required. In the UNIX/C version, this was quite untidy, since basically all the code funneled down to a single `select()` statement which monitored all the datagrams and allowed a single timeout. Much of the complexity of the stack involved trying to keep state across invocations of `select()` and in trying to workout which PDU would need to retry next, meaning that different requests needed to cooperate.

Java's multi-threading allow the handling of PDUs (requests) to be isolated from each other. A thread is allocated to each (outgoing) request. This thread is responsible for the timeouts/retries on that PDU and no other. There is also a thread that listens for incoming packets (replies) and hands them out. This results in a loop running in each PDU thread that looks like this:

```
answered = false;
for (int n = 0; n < retryIntervals.length; n++) {
  if (! answered){
    sendme(community);
  }
  try{
    sleep(retryIntervals[n]);
  } catch (java.lang.InterruptedException e) {}
}
```

The `answered` flag is set once a matching reply PDU is received by the listening thread. Whilst this is simple and clear, it is inefficient and caused problems on Windows. In particular, it creates a new thread for every request and holds onto it longer than is needed - until the sum of all the retry times has elapsed. This was changed to use a thread pool, where threads are reused once idle. Also a get-out clause was added, allowing the loop to finish early once a reply arrived.

The performance of the stack has never been a problem. It can saturate an Ethernet on a 600Mhz Pentium system and there are users who poll hundreds of hosts. In fact, the only complaint in that area was about a beta version, which due to a typo, dropped exactly half the packets it received. It is a tribute to the design of SNMP that this showed up as a performance problem, not as a total failure.

SNMPv3 Support

In 2000, Westhawk was commissioned to extend the stack to support SNMPv3. As the authors had worked on a proof of principle implementation of SNMPv2usec in 1996, as an extension of the 'C' stack, we were delighted to accept. SNMPv3 implements many of the same mechanisms as were in the USEC draft.

As it turned out, it was not quite as simple as was hoped. First a suitable open source set of lightweight classes had to be found that implement the cryptographic algorithms used by SNMPv3 for privacy and authentication (DES, MD5 and SHA1). A couple of sources were evaluated, and the code from BouncyCastle was selected in the end.

The authors of the SNMPv3 RFCs had assumed a certain style of ASN.1 encoder and decoder, which worked in such a way as to make it possible to replace byte ranges in the message. As mentioned above, this SNMP stack represents the PDU as a tree, and not as a linear array of bytes. An attempt was made to implement SNMPv3 in that style, by replacing parts of the tree, and then encoding it. However, this caused other problems since the RFC requires the replacement of the contents of an ASNSequence, not the sequence itself. The ASN object classes had to be extended, so that they take note of their byte offsets when encoding and decoding themselves. These offsets are then used as markers in the (say) outgoing byte array, and it is rewritten just before it is sent. This goes to show that no matter how hard you try to be independent of implementation details when you write a specification, you still run the risk of favoring a specific style by assuming how things will be done.

At the same time as implementing SNMPv3, support for receiving and sending traps was also added. Both of these required quite careful thought as to how to create a tidy API. In both cases, the problem boiled down to the fact that the stack is middle-ware, not a finished product.

In the case of receiving traps, the issue is that the manager program may not have created a context to receive the trap. If there is a context for the trap source, a user interface can simply add a trap listener to the existing context. An interactive management station might only create contexts for the devices that were currently on the screen, while wanting to accept traps from all the devices it can monitor. This presents a specific problem for SNMPv3 with privacy, since the context is used to store the authentication and privacy keys, as well as the time-line info (of which more later...). It was decided to create a `DefaultTrapListener`, as a last resort. Applications can ask to be notified when an 'unclaimed' trap message arrives. The application then has to create a suitable context, which can be used to decode the trap.

Sending a trap represents a different problem. It is the only element of agent functionality supported so far. SNMPv3 places additional requirements on agents in terms of keeping three bits of data:

1. `snmpEngineId` - a unique identification for the agent
2. `snmpEngineBoots` - the total number of reboots of the agent
3. `snmpEngineTime` - how long the agent has been running since the last reboot

SNMPv3 uses these to protect itself against replay attacks. An initial implementation provided reasonable values for these variables. However this code makes assumptions about the environment in which it runs. So a mechanism was added which allows programmers to supply their own class to provide these values and which can ensure the values are appropriate to the device.

MIBs and Beans

Over the years the user base has used the stack to talk to a wide variety of devices and their corresponding MIBs. Probably the most fun (and useful) was a manager application which monitored the state of an ISDN card (from Dialogic) which was running a telephone IVR (Interactive Voice Response) system. The level of detail in the Dialogic MIB was such that it could actually give a sense of what was happening on each call. It tracks the state of the DSP hardware - listening for DTMF tones, playing a recorded message, idle etc. Icons for each of these states were created and a simple dedicated manager application was produced for the operations staff to use so they could keep an eye on the system.

Others have used the stack to provide a gateway to CORBA, to scan the Internet facing mis-configured devices as part of a routing protocol testbench, as well as the bread and butter tasks of monitoring the status of routers and connections. Sadly, at least to the author's knowledge, no-one uses it to control a UPS!

Where possible the user base is encouraged to write Java beans to encapsulate the key aspects of the devices to be monitored. In the IVR example above, a `DialogicChannelStatusBean` was created. This provides a programmer with access to the data she needs to provide a visual representation of the state of the device with the minimum of exposure to SNMP. In fact the only things she needs to know is the IP address, the port number and the community name of the target SNMP agent. All of the MIB specific details, OIDs, tables, etc. are managed by the bean and not exposed to the user interface programmer.

In order to permit the user interface programmer to use multiple beans in her program, common resources need to be shared. A context pool was created, so that two beans that are talking to the same agent with the same community name will share the socket and communications threads. Java's interface mechanism was used to make the change invisible to most of the pre-existing user base.

The downside of this method is that the person who writes the beans has to understand the MIB, with little or no assistance from us. Personally I tend to use an interactive, MIB-aware, tool like Scotty to browse a new MIB, get some sample values from a real device and then capture the OIDs. Only then I start to code up a new bean. This has proved hard for some beginners, since there is quite a lot to learn all at once. Indeed, one group uses the stack as the basis of a course in SNMP, perhaps exactly because, at this level, it does not hide the detail of SNMP.

Open source

From the outset the stack has been a collaborative project, indeed the first code came from Jordan Hargrave, not from the authors. It has been interesting to lead such a project, but somewhat disheartening at times. Since the stack is middleware, the view was taken that neither the Gnu Public License, nor the Lesser Gnu Public License were appropriate for code which might be embedded in devices, or put on web servers in the form of applets. Instead a 'BSD' style license with very few conditions is used. In retrospect it should have required that users notify the authors of their use of the stack. As it is, the stack is found in various places (not always acknowledged) and the project gets no feedback from those users about how it could be improved.

On the other hand, the team gets quite a few emails from people with thanks and bug reports – both of which are equally welcome, if the bug report comes with a fix! It is always interesting to hear how people are using the stack since this info can be used to tune new versions. Some of the uses are unexpected and as such could be better supported if small changes were made in the stack. For example, there are users who poll hundreds of agents per minute and the stack creates and destroys many threads as a result. New I/O features in Java 1.4 may be able to reduce this churn and hence boost performance, but if the team aren't told, we can't address it.

In common with most projects of this (smallish) size, most of the code has been written by two people with many smaller contributions from others. In the 3 months since 6th of September 2001, the stack has been downloaded some 1190 times, and there are some 93 email addresses on the list of individuals who are notified whenever a new version comes out.

Future

As mentioned above we expect to use Java 1.4 features to improve support polling of large numbers of agents. Requests have been made to add support for agent functionality. This has been investigated, but is unlikely to happen unless someone provides some encouragement, either in the form of code or money!

Summary

What started as a proof of concept not only proved its point, but has turned into more than we expected! The fact that it is lightweight, freeware and open source means that it can be used by everyone for every purpose. The way the stack is designed makes it straightforward to experiment with different protocols or to extend it in any other ways to suit your needs.

With its features, Java has proved an excellent language for the stack. With its current development, Java holds lots in store for the future of the stack.

Reality Check: IETF meets Network Operators

Steve Feldman, Vivace Networks

Earlier this year, the Operations and Management area directors in the IETF started a dialog between network management protocol designers and network operators to help in determining future directions for work in the IETF. As part of that effort the IETF held several interim meetings in conjunction with meetings of various network operator groups.

In May 2001, a meeting was held in conjunction with NANOG, the North American Network Operators Group, in Scottsdale, Arizona. Similar meetings were held in October, at the RIPE/NCC meeting in Prague, and in November at the USENIX LISA conference in San Diego.

It quickly became clear that although SNMP is popular for monitoring, most network operators are not using it for device configuration. Reasons for this include lack of vendor support, difficulty in scripting and debugging, and the desire to use a common configuration method regardless of the communication mechanism (e.g. console port or network access.)

During the meeting with NANOG, some network operators volunteered to write a document describing their requirements for methods to configure devices in service provider networks. This document has been posted as an internet-draft[1], with the intention to make it an informational RFC once it is complete and consensus has been reached on the contents in the traditional IETF style. The goal is to have this done before for the next IETF meeting, in March 2002.

Some of the requirements proposed for inclusion in the document include:

- All devices must accept an ASCII command line interface (CLI) via the network and a console port.
- All access to a device must be possible via a commonly available security and authentication mechanism. The document will not mandate a specific protocol, but suggest some possibilities, such as SSH, Kerberos, and SSL.
- A common configuration language must be developed so that similar operations performed on different vendors' platforms will use the same command syntax.
- It must be possible to write the entire configuration (including default values) of a device to a text file, which can then be read by another similar router to produce an identical configuration.
- The command interface must provide for automation of management tasks. In particular, command output must be machine parseable and include a numeric result code. It is permissible to have separate modes for human and machine-readable output, but both modes must contain the same information.

There are quite a few more potential requirements under active discussion. Network operators are encouraged to read the draft document and join in the mailing list discussion at ops-nm@ietf.org. (Send subscription requests to ops-nm-request@ops.ietf.org.)

References

[1] Woolf S., Woodcock, B., Operator Requirements of Infrastructure Management Methods, work in progress, 2001.

Four Engineers and a Troublemaker

Tom Cikoski

A recent poster on the `comp.protocols.snmp` Usenet news group asked about the availability of software to convert "CMIP GDMO to SNMP ASN.1." Esteemed SNMP author Dave Perkins replied that, were such a task possible, it would require a team of "4 engineers and a trouble maker." He then went one to list the roles in his proposed team.

1. SNMP MIB Expert (*SNMP*)
2. CMIP MIB Expert (*CMIP*)
3. Subject Matter Expert (*Subj*)
4. Application Developer (*Devel*)

5. Network Operator (*User*)

Since Dave did not explicitly identify the "trouble-maker" on the team, speculation ran rife, and several news group regulars wrote me, identifying the "obvious choice." The answers all differed.

So, by special permission of the parties involved, I am providing a transcript of the first meeting of the above team with their venture capital provider, Mr. Muncie ("Mun") E. Baggs. You can decide for yourself who the trouble maker is.

- *Baggs:* First I'd like to thank all of you for coming today.
- *User:* Ok, but where's the rest room?
- *Devel:* And where's the phone?
- *CMIP:* And where's the food?
- *SNMP:* And where's the beer?
- *Baggs:* Ladies and gentlemen, all of these details are in your registration bags, along with your free luggage tag and "I Like Founder's Options" tee shirt. Now, would one of you please comment on the feasibility of the business plan?
- *Devel:* Well, as you know from all the other software startups you've funded, we can create any program you can imagine given enough staff, time, equipment and gourmet coffee.
- *Baggs:* So?
- *Devel:* Put another hundred coders and two more years in the plan. I'll have the beta done a year after that, and then we'll all change jobs for a new start up.
- *CMIP:* This whole effort is pointless. CMIP is so far superior to SNMP that the product is unnecessary. SNMP is only a passing phase anyway. Say, would anyone like a new shelf load of requirements documents?
- *Subj:* A MIB is a horse designed by a committee. I've seen both these so-called MIBs and neither one adequately describes the underlying reality. The real world is not a discrete set of disconnected values with N choices of operational response. It's all over the map!
- *SNMP:* CMIP is so pointlessly bizarre that it could not be boiled down to the elegant simplicity of the SMI. What would be the point. Besides, SMI isn't ASN.1, so you need to retitle the plan anyway. Hey, just do a new SMIv2 MIB by hand. Is there any Pauli Girl in here?
- *User:* While you all sit here playing with yourselves, I still have a network to run. I have to provide high reliability at low cost with limited staff. I have to react to every new technology that comes along whether I want to or not, and learn every new acronym as soon as it surfaces. There are still no management products that live up to the expectations the sales guys put into my boss's head. And you think I'm going to care about this product? Feh!
- *Baggs:* Thank you all for your incisive comments. I see we have the makings of a whole new industry leviathan here. I'll announce the product at the next Interop!

Standards Summary

Please consult the latest version of *Internet Official Protocol Standards*. As of this writing, the latest version is RFC 3000.

SMIv1 Data Definition Language

Full Standards:

- RFC 1155 - Structure of Management Information
- RFC 1212 - Concise MIB Definitions

Informational:

- RFC 1215 - A Convention for Defining Traps

SMIv2 Data Definition Language

Full Standards:

- RFC 2578 - Structure of Management Information
- RFC 2579 - Textual Conventions
- RFC 2580 - Conformance Statements

SNMPv1 Protocol

Full Standards:

- RFC 1157 - Simple Network Management Protocol

Proposed Standards:

- RFC 1418 - SNMP over OSI
- RFC 1419 - SNMP over AppleTalk
- RFC 1420 - SNMP over IPX

SNMPv2 Protocol

Draft Standards:

- RFC 1905 - Protocol Operations for SNMPv2
- RFC 1906 - Transport Mappings for SNMPv2
- RFC 1907 - MIB for SNMPv2

Experimental:

- RFC 1901 - Community-based SNMPv2
- RFC 1909 - Administrative Infrastructure
- RFC 1910 - User-based Security Model

SNMPv3 Protocol

Draft Standards:

- RFC 2571 - Architecture for SNMP Frameworks
- RFC 2572 - Message Processing and Dispatching
- RFC 2573 - SNMP Applications
- RFC 2574 - User-based Security Model
- RFC 2575 - View-based Access Control Model
- RFC 1905 - Protocol Operations for SNMPv2
- RFC 1906 - Transport Mappings for SNMPv2
- RFC 1907 - MIB for SNMPv2

Proposed Standards:

- RFC 2576 - Coexistence between SNMP Versions

Informational:

- RFC 2570 - Introduction to SNMPv3

Experimental:

- RFC 2786 - Diffie-Helman USM Key Management

SNMP Agent Extensibility

Draft Standards:

- RFC 2741 - AgentX Protocol Version 1
- RFC 2742 - AgentX MIB

SMIv1 MIB Modules

Full Standards:

- RFC 1213 - Management Information Base II
- RFC 1643 - Ethernet-Like Interface Types MIB

Draft Standards:

- RFC 1493 - Bridge MIB
- RFC 1559 - DECnet phase IV MIB

Proposed Standards:

- RFC 1285 - FDDI Interface Type (SMT 6.2) MIB
- RFC 1381 - X.25 LAPB MIB
- RFC 1382 - X.25 Packet Layer MIB
- RFC 1414 - Identification MIB
- RFC 1461 - X.25 Multiprotocol Interconnect MIB
- RFC 1471 - PPP Link Control Protocol MIB
- RFC 1472 - PPP Security Protocols MIB
- RFC 1473 - PPP IP NCP MIB
- RFC 1474 - PPP Bridge NCP MIB
- RFC 1512 - FDDI Interface Type (SMT 7.3) MIB
- RFC 1513 - RMON Token Ring Extensions MIB
- RFC 1525 - Source Routing Bridge MIB
- RFC 1742 - AppleTalk MIB

SMIv2 MIB Modules

Full Standards:

- RFC 2819 - Remote Network Monitoring MIB

Draft Standards:

- RFC 1657 - BGP version 4 MIB
- RFC 1658 - Character Device MIB
- RFC 1659 - RS-232 Interface Type MIB
- RFC 1660 - Parallel Printer Interface Type MIB
- RFC 1694 - SMDS Interface Type MIB
- RFC 1724 - RIP version 2 MIB
- RFC 1748 - IEEE 802.5 Interface Type MIB

- RFC 1850 - OSPF version 2 MIB
- RFC 1907 - SNMPv2 MIB
- RFC 2115 - Frame Relay DTE Interface Type MIB
- RFC 2571 - SNMP Framework MIB
- RFC 2572 - SNMPv3 MPD MIB
- RFC 2573 - SNMP Applications MIBs
- RFC 2574 - SNMPv3 USM MIB
- RFC 2575 - SNMP VACM MIB
- RFC 2790 - Host Resources MIB
- RFC 2863 - Interfaces Group MIB

Proposed Standards:

- RFC 1666 - SNA NAU MIB
- RFC 1696 - Modem MIB
- RFC 1697 - RDBMS MIB
- RFC 1747 - SNA Data Link Control MIB
- RFC 1749 - 802.5 Station Source Routing MIB
- RFC 1759 - Printer MIB
- RFC 2006 - Internet Protocol Mobility MIB
- RFC 2011 - Internet Protocol MIB
- RFC 2012 - Transmission Control Protocol MIB
- RFC 2013 - User Datagram Protocol MIB
- RFC 2020 - IEEE 802.12 Interfaces MIB
- RFC 2021 - RMON Version 2 MIB
- RFC 2024 - Data Link Switching MIB
- RFC 2051 - APPC MIB
- RFC 2096 - IP Forwarding Table MIB
- RFC 2108 - IEEE 802.3 Repeater MIB
- RFC 2127 - ISDN MIB
- RFC 2128 - Dial Control MIB
- RFC 2206 - Resource Reservation Protocol MIB
- RFC 2213 - Integrated Services MIB
- RFC 2214 - Guaranteed Service MIB
- RFC 2232 - Dependent LU Requester MIB
- RFC 2238 - High Performance Routing MIB
- RFC 2266 - IEEE 802.12 Repeater MIB
- RFC 2287 - System-Level Application Mgmt MIB
- RFC 2320 - Classical IP and ARP over ATM MIB
- RFC 2417 - Multicast over UNI 3.0/3.1 / ATM MIB
- RFC 2452 - IPv6 UDP MIB
- RFC 2454 - IPv6 TCP MIB
- RFC 2455 - APPN MIB
- RFC 2456 - APPN Trap MIB
- RFC 2457 - APPN Extended Border Node MIB
- RFC 2465 - IPv6 Textual Conventions and MIB
- RFC 2466 - ICMPv6 MIB
- RFC 2493 - 15 Minute Performance History TCs
- RFC 2494 - DS0, DS0 Bundle Interface Type MIB
- RFC 2495 - DS1, E1, DS2, E2 Interface Type MIB
- RFC 2496 - DS3/E3 Interface Type MIB
- RFC 2512 - Accounting MIB for ATM Networks
- RFC 2513 - Accounting Control MIB
- RFC 2514 - ATM Textual Conventions and OIDs
- RFC 2515 - ATM MIB
- RFC 2558 - SONET/SDH Interface Type MIB
- RFC 2561 - TN3270E MIB
- RFC 2562 - TN3270E Response Time MIB
- RFC 2564 - Application Management MIB
- RFC 2576 - SNMP Community MIB
- RFC 2584 - APPN/HPR in IP Networks
- RFC 2591 - Scheduling MIB
- RFC 2594 - WWW Services MIB
- RFC 2605 - Directory Server MIB
- RFC 2613 - RMON for Switched Networks MIB
- RFC 2618 - RADIUS Authentication Client MIB
- RFC 2619 - RADIUS Authentication Server MIB

- RFC 2667 - IP Tunnel MIB
- RFC 2662 - ADSL Line MIB
- RFC 2665 - Ethernet-Like Interface Types MIB
- RFC 2668 - IEEE 802.3 MAU MIB
- RFC 2669 - DOCSIS Cable Device MIB
- RFC 2670 - DOCSIS RF Interface MIB
- RFC 2677 - Next Hop Resolution Protocol MIB
- RFC 2720 - Traffic Flow Measurement Meter MIB
- RFC 2737 - Entity MIB
- RFC 2742 - AgentX MIB
- RFC 2787 - Virtual Router Redundancy Proto. MIB
- RFC 2788 - Network Services Monitoring MIB
- RFC 2789 - Mail Monitoring MIB
- RFC 2873 - Fibre Channel Fabric Element MIB
- RFC 2851 - Internet Network Address TCs
- RFC 2856 - High Capacity Data Type TCs
- RFC 2864 - Interfaces Group Inverted Stack MIB
- RFC 2895 - RMON Protocol Identifier Reference
- RFC 2925 - Ping, Traceroute, Lookup MIBs
- RFC 2932 - IPv4 Multicast Routing MIB
- RFC 2933 - IGMP MIB
- RFC 2940 - COPS Client MIB
- RFC 2954 - Frame Relay Service MIB
- RFC 2955 - Frame Relay / ATM PVC MIB
- RFC 2959 - Real-Time Transport Protocol MIB
- RFC 2981 - Event MIB
- RFC 2982 - Expression MIB
- RFC 3014 - Notification Log MIB
- RFC 3019 - Multicast Listener Discovery MIB
- RFC 3020 - Frame Relay UNI/NNI Multilink MIB
- RFC 3055 - PSTN/Internet Interworking MIB
- RFC 3083 - DOCSIS Baseline Privacy Interface MIB
- RFC 3144 - RMON Interface Monitoring MIB
- RFC 3165 - Scripting MIB

Informational:

- RFC 1628 - Uninterruptible Power Supply MIB
- RFC 2620 - RADIUS Accounting Client MIB
- RFC 2621 - RADIUS Accounting Server MIB
- RFC 2666 - Ethernet Chip Set Identifiers
- RFC 2707 - Print Job Monitoring MIB
- RFC 2896 - RMON Protocol Identifier Macros
- RFC 2922 - Physical Topology MIB

Experimental:

- RFC 2758 - SLA Performance Monitoring MIB
- RFC 2786 - Diffie-Helman USM Key MIB
- RFC 2934 - IPv4 PIM MIB

IANA Maintained MIB Modules

The Internet Assigned Numbers Authority (IANA) maintains several MIB modules. The IANA MIB repository is located at ftp://ftp.iana.org/mib/iana.mib/.

- Interface Type Textual Convention (ianaiftype.mib)
- Address Family Numbers Textual Convention (ianaaddressfamilynumbers.mib)
- TN3270E Textual Conventions (ianatn3270etc.mib)
- Language Identifiers (ianalanguage.mib)
- IP Routing Protocol Textual Conventions (ianaiprouteprotocol.mib)

Related Documents

Informational:

- RFC 1270 - SNMP Communication Services
- RFC 1321 - MD5 Message-Digest Algorithm
- RFC 1470 - Network Management Tool Catalog
- RFC 2039 - Applicability of Standard MIBs to WWW Server Management

- RFC 2962 - SNMP Application Level Gateway for Payload Address Translation
- RFC 2975 - Introduction to Accounting Management
- RFC 3052 - Service Management Architectures Issues and Review
- RFC 3216 - SMIng Objectives

Experimental:

- RFC 1187 - Bulk Table Retrieval with the SNMP
- RFC 1224 - Techniques for Managing Asynchronously Generated Alerts
- RFC 1238 - CLNS MIB
- RFC 1592 - SNMP Distributed Program Interface
- RFC 1792 - TCP/IPX Connection MIB Specification
- RFC 3139 - Requirements for Configuration Management of IP-based Networks
- RFC 3179 - Script MIB Extensibility Protocol 1.1
- RFC 3198 - Terminology for Policy-Based Management

Open Source News

NET-SNMP 4.2.3

`http://net-snmp.sourceforge.net/`

It is now more than one year back that the `NET-SNMP` sourceforge project started from the `UCD-SNMP` sources. The latest release of `NET-SNMP`, version 4.2.3, appeared at the end of November 2001. This version removes a large number of minor errors, and improves support for the MIB-II and the host resources MIB. Recently, work on the upcoming release 5.0 has started.

scli 0.2.6

`http://www.ibr.cs.tu-bs.de/projects/scli/`

The SNMP command line interface (`scli`) is a new open source package which provides a command line interface to display, modify and monitor data retrieved from SNMP agents. The scli provides command line editing, completion and history capabilities. The `scli` commands are MIB aware and organized in a logical command tree and hide the details of the SNMP interactions and the underlying MIB data structures.

libsmi 0.3.0

`http://www.ibr.cs.tu-bs.de/projects/libsmi/`

In November 2001, version 0.3.0 of `libsmi` appeared. The `libsmi` is a C library that allows applications to access SMI MIB module information through a well defined API. The distribution contains the `smilint` and `smidump` tools for MIB module validation and conversion. The most significant change since the previous 0.2.x releases is the addition of the `smidiff` tool which can be used to compare two revisions of the same MIB module.

ntop 2.0

`http://www.ntop.org/`

Ntop version 2.0 was released in December 2001. It provides improved stability and performance enhancements plus some new features. The most interesting new feature is probably the ability to export traffic flows in Cisco's NetFlow format. Work has already started on ntop version 2.0.1 which will include a mirror mode where flow information is collected by ntop for traffic analysis.

ethereal 0.9

`http://www.ethereal.com/`

Ethereal is a free graphical protocol analyzer which allows operators to browse the captured data. Ethereal 0.9.0 has been released in December 2001 and many new protocol dissectors.

Recent Publications

Managing Business and Service Networks

- Author: Lundy Lewis <lewis@aprisma.com>
- Publisher: Kluver Academic / Plenum Publishers `http://www.kluver.com/`
- ISBN: 0-306-46559-0
- Available: March, 2001

This book explains the challenges involved in managing todays communication networks. The first part of the book introduces general network management concepts and the architecture of Aprisma's Spectrum management system. The second and probably the most interesting part of the book describes several case studies where management technologies have been applied to solve real-world management problems. The third part tries to draw some conclusions for future directions of network management research and development.

The text is very specific to Aprisma's Spectrum management system and sometimes has a touch of a marketing document. This style comes as no surprise once you realize that the author has been heavily involved in the development of Spectrum and owns several patents in this area.

The book provides useful information for readers who want to better understand the overall picture of managing networks and who like to learn how management systems can be integrated and applied in real-world networks. The third part also contains an interesting description of the gap between real-world network operations and network management research and why research results are only very slowly adopted by the industry.

Essential SNMP

- Authors: Douglas R. Mauro, Kevin J. Schmidt
- Publisher: O'Reilly & Associates `http://www.oreilly.com/`
- ISBN: 0-596-00020-0
- Available: July, 2001

This O'Reilly book on SNMP does not spend lots of pages to explain all the little details of the SNMP technology. The authors instead cover SNMP-based management software which is widely deployed, such as HP's OpenView, Castle Rock's SNMPc, NET-SNMP, and MRTG. They explain in practical examples how to configure agents and to setup polling and monitoring strategies.

Calendar and Announcements

IETF Meetings:

- 52nd Meeting of the IETF
  December 9-14, 2001, Salt Lake City, UT, USA
- 53rd Meeting of the IETF
  March 17-22, 2002, Minneapolis, MN, USA
- 54th Meeting of the IETF
  July 14-19, 2002, Yokohama, Japan

Conferences and Workshops:

- Large Installation System Administration Conference 2001 (LISA 2001)
  December 2-7, 2001, San Diego, CA, USA
- Network Operations and Management Symposium 2002 (NOMS 2002)
  April 15-19, 2002, Florence, Italy
- Policy Workshop (Policy 2002)
  June 5-7, 2002, Monterey, CA, USA
- Workshop on Distributed Systems Operations and Management 2002 (DSOM 2002)
  October 21-23, 2002, Montreal, Canada
- Integrated Network Management (IM 2003)
  March 24-28, 2003, Colorado Springs, CO, USA

Exhibitions and Trade Shows:

- NetWorld + Interop Sydney
  March 5-7, 2002, Syndey, Australia
- NetWorld + Interop Las Vegas
  May 5-10, 2002, Las Vegas, USA
- NetWorld + Interop Tokyo
  July 1-5, 2002, Tokyo, Japan
- NetWorld + Interop Toronto
  July 10-12, 2002, Toronto, Canada
- NetWorld + Interop Sao Paulo
  August 20-23, 2002, Sao Paulo, Brazil
- NetWorld + Interop Melbourne
  September 4-6, 2002, Melbourne, Australia
- NetWorld + Interop Atlanta
  September 8-13, 2002, Atlanta, USA
- NetWorld + Interop Paris
  November 6-8, 2002, Paris, France

Publication Information

Contact Information

E-mail	`st-editorial@simple-times.org`
ISSN	1060–6068

Submissions

The Simple Times solicits high-quality articles of technology and comment. Technical articles are refereed to ensure that the content is marketing-free. By definition, commentaries reflect opinion and, as such, are reviewed only to the extent required to ensure commonly-accepted publication norms.

The Simple Times also solicits terse announcements of products and services, publications, and events. These contributions are reviewed only to the extent required to ensure commonly-accepted publication norms.

Submissions are accepted only via electronic mail, and must be formatted in HTML version 1.0. Each submission must include the author's full name, title, affiliation, postal and electronic mail addresses, telephone, and fax numbers. Note that by initiating this process, the submitting party agrees to place the contribution into the public domain.

Subscriptions

The Simple Times is available in HTML, PDF and PostScript. New issues are announced via an electronic mailing list. Send electronic mail to

```
st-request@simple-times.org
```

with

```
subscribe simple-times
```

in the body if you want to subscribe to this list. Back issues are available via *The Simple Times* Web server:

```
http://www.simple-times.org/
```