Management of TNCs by means of the Simple Network Management Protocol *

H. Hmida (VA2HLH) and M. Barbeau (VE2BPM)
Département de mathématiques et d'informatique
Université de Sherbrooke
Sherbrooke (Québec), CANADA, J1K 2R1
{hmida,barbeau}@dmi.usherb.ca

Abstract

This article deals with the application of a network management framework, called *Simple Network Management Protocol* (SNMP), to manage a particular type of network devices named *Terminal Node Controller* (TNC). TNCs are widely used in the amateur packet radio community. We present new tools based on SNMP for remote management of TNCs. A *Management Information Base* (MIB) has been created for the TNCs parameters we manage in KISS mode. The MIB is implemented under the *Linux* operating system and uses the CMU-SNMP package. We implemented also a new command to manage simultaneously and remotely several TNC parameters.

Keywords: Network management, terminal node controller, management information base, Linux.

1 Introduction

The number of users accessing amateur packet radio is growing from day to day. Managers of telecommunications equipment are called to pay more attention to these pieces of equipment in order to exploit them in a better way. So far, managers were obliged to be close to the packet radio devices they wish to manage and to control. The *Simple Network Management Protocol* (SNMP) [2] is a framework that permits remote management and control of network devices. With a simple design, SNMP has the capability to manage various types of network technologies. For example, SNMP is used for the management of the Internet. The concerns of this paper is the use of SNMP to manage and control wireless networks. In particular, we are interested in the application of SNMP for the control and management of a device type called TNC [3]. A TNC is an important element

*The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR).

of networks of the amateur packet radio community. It is a device placed between a transceiver and a computer implementing a medium access protocol and a data link control protocol.

Classically, management of a TNC can be done using two types of tools, namely, terminal emulation software and utility programs [5]. These tools are helpful as long as the computer on which they run is directly connected to the TNC with a short cable. In other words, these tools are inadequate for remote management of TNCs through networks. The goal of our work is to develop and provide new tools for remote management of TNCs. These types of tools are interesting since the manager that uses them is not obliged to be close to the TNC he wants to manage.

The tools that we propose are based on SNMP. In SNMP, the collection of manageable param eters of a device are logically seen as a data base called a management information base (MIB) [4]. Data elements in a MIB are called objects which are abstractions of real parameters of a device. A MIB offers different services in order to read and modify the values of those objects. SNMP is a protocol that allows use of these services remotely through a network. In order to define the structure of objects in a MIB, the SNMP framework defines a notation called structure of management information (SMI) [1] [2].

In our work, we have identified the TNC parameters that can be managed in KISS mode and defined a MIB for them using the SMI syntax. This MIB is implemented under the Linux operating system and uses the CMU-SNMP package [6]. In addition to the commands coming with this package, we implemented a new command allowing to manage simultaneously and remotely different parameters of a TNC. Those commands access the TNC via SNMP and the MIB we created.

The rest of this paper is structured as follows. Section 2 deals with the basic aspects of SNMP. Section 3 deals with the solution we developed. First, it presents the parameters we manage and the corresponding objects that form our MIB. Second, it explains how to access, read, and set the MIB objects. Finally, it describes the relation between the TNC, the MIB, and how SNMP physically sets the TNC parameters. We conclude with Section 4.

2 Review of SNMP

SNMP was designed as an application-level protocol that is part of the TCP/IP protocol suite (see Figure 1). Four main concepts come with SNMP: management station, agent, management information base, and management protocol. A management station has a set of management applications and an interface by which the network manager may monitor and control a network. An agent responds to requests for information and requests for actions from the management station and may asynchronously provide the management station with important but unsolicited information. It provides the information by accessing its local MIB and retrieving the requested object values. The MIB is a database containing a collection of objects. Each object refers to a network-resource to be managed and is accessed via a data variable. The management station performs the monitoring function by retrieving values of MIB objects. It can change also values of specific variables. Finally, a network management protocol links the management station and agent. The one used for the management of TCP/IP networks is SNMP.

SNMP is implemented on top of the user datagram protocol (UDP), internet protocol (IP), and the relevant network-dependent protocol (e.g., Ethernet, AX.25). SNMP provides a message exchange mechanism to retrieve and set object values at the agent and notify the management station of significant events. From a management station, three types of SNMP messages are

issued on behalf of a management application: *GetRequest*, *GetNextRequest*, and *SetRequest*. A GetRequest is issued if the management application knows precisely the object it wants to read. A GetNextRequest is issued if the management application wants to read the value of an object that is successor of known object. The GetRequest and the GetNextRequest can only read objects. SetRequest, however, may create or modify objects. These messages are acknowledged by the agent as a *GetResponse* messages, which are passed up to the management application. In addition, an agent may issue *trap* messages in response to events that affect the MIB and underlying managed resources.

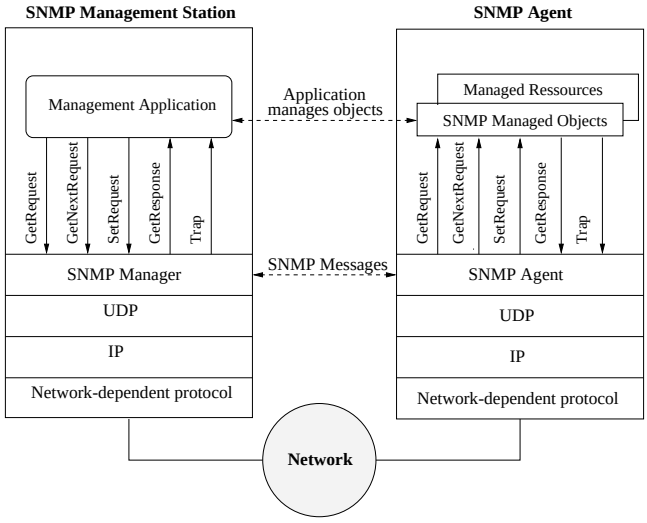

Figure 1: SNMP architecture [1].

3 Development of a solution

In this section, we present a solution we wish propose to solve remote management and control of wireless network devices. The tools we propose are developed under the Linux operating system [8]. Linux is chosen because it has a certain degree of security (like Unix) concerning the access to the system, it supports communication over the AX.25 protocol, and it supports network management. One of the most free popular SNMP packages is CMU-SNMP. It was first designed by the Carnegie Mellon University (http://www.cmu.edu) and then has been ported to Linux by Juergen Schoenwaelder (schoenw@gaertner.de) and Erik Schoenfelder (schoenfr@gaertner.de). This package is fully compliant with the SNMPv1 standard and includes, in addition, some of the new features of SNMPv2. The distribution contains some manager tools that permit, in a command line style, to send requests to devices running SNMP agents. It also contains a SNMP agent program, designed to run under Linux, that provides the manager running on the network (or the same system) information about the status of the interfaces, routing table, uptime, etc. [7]

3.1 Managed parameters and their corresponding objects

In a MIB, objects are organized into groups. Since we are in an experimental stage, we include our new objects under a group called *experimental group*. Let's now have a look at the objects of a TNC in KISS mode that can be managed and included into a MIB. The KISS mode has a limited number of manageable parameters. They are the following:

- port: the port that is being configured
- fullduplex: sets the TNC into either full duplex or half duplex
- hardware: hardware specific parameters
- txtail: the TX Tail time in milliseconds
- persist: the persist value in milliseconds
- slottime: the slottime in milliseconds
- txdelay: the TX Delay in milliseconds

These TNC parameters, in KISS mode, can be set but their value cannot be retrieved from the TNC. The MIB that we developed allows the management application to get the values given to the parameters. We associated to each TNC parameter a MIB object that reflects its value. For example, the definition of the object associated to the parameter *txtail* is illustrated in Figure 2. The notation is SMI.

```
tncTxTail  OBJECT-TYPE
SYNTAX          INTEGER (1..127)
MAX-ACCESS   read-write
STATUS           mandatory
DESCRIPTION
        "Sets the Tx Tail time in milliseconds. This value must be
        a multiple of 10."
::= { tncControl 4}
```

Figure 2: Definition of the tncTxTail object.

Figure 2 defines an object called *tncTxTail* situated under the group *tncControl*. The value 4 means that our object is the number four in the *tncControl* group (Figure 3). Also, the definition tells that the value of the object is of type *INTEGER* and can vary between 0 and 127. The label *MAX-ACCESS* defines the way an instance of the object can be accessed (via SNMP or other protocols). In our case, the object is readable and writable (setable). Generally, it can be one of the following: *read-only*, *read-write*, *write-only*, or *not-accessible*.

The *STATUS* clause indicates the implementation support required for this object. Support can be one of the following: *mandatory* (the object definition is valid and implementation is required for conformance), *current* (the object definition is valid), *optional* (the object definition is valid,

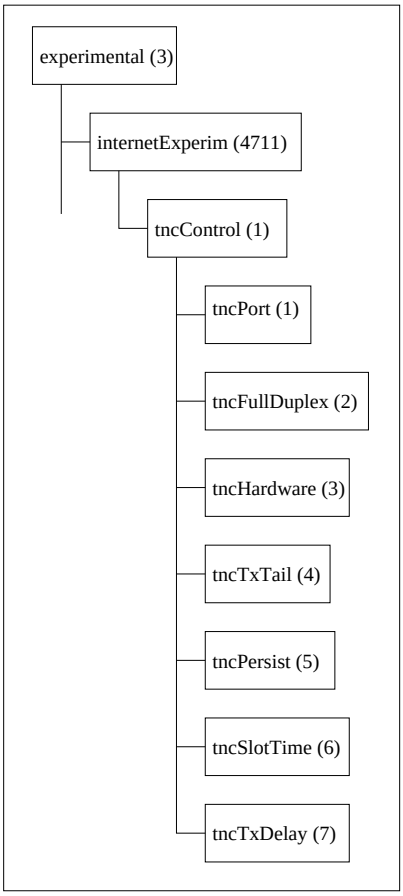

Figure 3: TNC sub-tree.

however implementation is not required for conformance), *deprecated* (the object definition is valid in limited circumstances, but has been replaced by another), or *obsolete* (managed nodes need no longer to implement this object). A valid definition has the following properties: it is well conceived (precise, unambiguous, complete, and applicable across a wide scope) and relevant (useful and has been used for implementation). With SMIv1, the value *current* is not used. With SMIv2, value *optional* is eliminated and the value *mandatory* is replaced by *current*. The complete SMI definition of all the objects corresponding to the parameters that we manage is given in Appendix 4.

3.2 Object access

CMU-SNMP provides three commands for retrieving the values of objects. The first is the *snmpwalk* command. It has three parameters:

- hostname: corresponds to the host name containing the MIB. It can be either *localhost*, if we are addressing our local machine MIB, or the explicit name of a distant host in a textual form (e.g., lolly2.dmi.usherb.ca) or in a numerical form (e.g., 132.210.48.188).
- community name: is one of the community names known by the addressed host. In our case

we use *public*.

- object-ID: is the identifier of the group of objects to be walked. The identifier can be either in a textual form (e.g., system) or in a numerical one (e.g., .1.3.6.1.2.1.1) or a combination of the two (e.g., .1.3.6.1.2.1.system).

To discover, for instance, all the objects under the *tncControl* group, we have just to type:

```
snmpwalk localhost public .1.3.6.1.3.4711
```

a normal response is:

```
internetExperim.tncControl.tncPort.0 = 1
internetExperim.tncControl.tncFullDuplex.0 = 1
internetExperim.tncControl.tncHardware.0 = 3
internetExperim.tncControl.tncTxTail.0 = 20
internetExperim.tncControl.tncPersist.0 = 10
internetExperim.tncControl.tncSlotTime.0 = 10
internetExperim.tncControl.tncTxDelay.0 = 10
```

Another way to obtain the values of these objects is to use the *snmpget* command. To do so, we have to know the exact name of an object. For example, to get the value of the *tncPort* object we can use the following command:

```
snmpget localhost public
.1.3.6.1.3.4711.tncControl.tncPort.0
```

a normal returned result is:

```
internetExperim.tncControl.tncPort.0 = 1
```

To know the value of the object next to *tncPort*, we can apply the *snmpgetnext* command in the following way:

```
snmpgetnext localhost public
.1.3.6.1.3.4711.tncControl.tncPort.0
```

a normal response is:

```
internetExperim.tncControl.tncFullDuplex.0 = 1
```

This way, we can discover the value of any object within the MIB. To change the value of any writable object, we have just to apply the *snmpset* command. In addition to the hostname, community name, and object-ID, this command needs the type of the object and the new value to be assigned to it. The type can be: *i (INTEGER), s (STRING), x (HEX STRING), d (DECIMAL STRING), n (NULLOBJ), o (OBJID), t (TIMETICKS)* or *a (IPADDRESS)*. Here is an example:

```
snmpset localhost public
internetExperim.tncControl.tncSlotTime.0 i 20
```

The *snmpset* command can set only one parameter at a time. To present to the user a better service, we developed a new command, called *snmpax25set*, that permits to set several parameters in a single command line. The syntax of this new command is the following:

```
snmpax25set hostname community object [-p port][-f 0|1][-l txtail][-r persist][-s slot][-t txt]
```

Once the parameters passed to the *snmpax25set* are verified, *snmpset* is called for each one of them.

3.3 Application implementation

Hereafter, we discuss how this application is implemented and how someone can write applications similar to the one we are presenting. We explain in the following lines the C code of the main functions we wrote for this application and the key files necessary for its implementation.

Installation of the CMU-SNMP package for Linux is required. The application uses basically the following three files:

1. *mib.txt*: contains the SMI definition of the MIB objects.

2. *snmp_vars.h*: contains function, constant, and data structure declarations.

3. *snmp_vars.c*: contains the code of the functions necessary to access, read, and set MIB objects.

First of all, the new objects to manage must be added in the appropriate object-group of the MIB. In our case, they are added to the group *tncControl* (a subgroup of *internetExperim*, which is itself a subgroup of *experimental*) (see Figure 3). The part of the file *mib.txt* defining the group *tncControl* is given in Appendix A. The file *mib.txt* also contains predefined standard managed objects. Next, the *tncControl* group need to be registered in order to be known by the agent. The function *snmp_vars_init* in the file *snmp_vars.c* does registration of managed objects. Here are the lines we added to the *snmp_vars_init* function:

```
#ifdef linux
mib_register (
   tnc_id_base,
   sizeof(tnc_id_base) / sizeof(oid), /*num. of obj. id in tnc_id_base*/
   experimental_variables,
   sizeof(experimental_variables)/sizeof(*experimental_variables), /*num. of vars. in the group*/
   sizeof(*experimental_variables)); /*size of a variable description*/
```

tnc_id_base is a vector that defines the identifier of the group *internetExperim* (see Appendix A):

```
#define INTERNET_EXPERIMENTAL 1, 3, 6, 1, 3
static oid tnc_id_base[] =
{
 INTERNET_EXPERIMENTAL, 4711
};
```

experimental_variables is an array that defines all the *tncControl* group objects to manage, their type, access mode, management function, and index under the *internetExperim* group. It is globally defined as follows:

```
struct variable2 experimental_variables[]={
{TNCPORT,       INTEGER, RWRITE, var_experimental, 2, {1, 1}},
{TNCFULLDUPLEX, INTEGER, RWRITE, var_experimental, 2, {1, 2}},
{TNCHARDWARE,   INTEGER, RWRITE, var_experimental, 2, {1, 3}},
{TNCTXTAIL,     INTEGER, RWRITE, var_experimental, 2, {1, 4}},
{TNCPERSIST,    INTEGER, RWRITE, var_experimental, 2, {1, 5}},
{TNCSLOTTIME,   INTEGER, RWRITE, var_experimental, 2, {1, 6}},
{TNCTXDELAY,    INTEGER, RWRITE, var_experimental, 2, {1, 7}}
};
```

TNCPORT, TNCFULLDUPLEX, TNCHARDWARE, TNCTXTAIL, TNCPERSIST, TNCSLOTTIME, and TNCTXDELAY are defined, in the *snmp_vars.h* file, as follows:

```
#define TNCPORT            1
#define TNCFULLDUPLEX      2
#define TNCHARDWARE        3
#define TNCTXTAIL          4
#define TNCPERSIST         5
#define TNCSLOTTIME        6
#define TNCTXDELAY         7
```

Now, we define new functions that access, read, and set the object values. The main function is *var_experimental*. It is called for each object and is partially defined as:

```
u_char *var_experimental(...)
{
    /* Declaration of local variables */
    ...

    /* Memory allocation for an object */
    ...

    /* switch to object currently handled */
    switch (...){
      case TNCPORT:
            *write_method = write_snmp_tnc_Port; (1)
            long_return = snmp_tnc_port; (2)
            break;
        /* other cases */
        ...
    }
    return ...;
}
```

There is a case for each object we are registering. For example, let us consider the case TNCPORT. There are two actions. First, the function responsible for access to the object is registered, that is *write_snmp_tnc_Port* (1). Second, the initial value of the object is registered, that is *snmp_tnc_port* (2). *snmp_tnc_port* is a memory variable that we define (in file *snmp_vars.c*) and that stores the current value of the managed object.

Function *write_snmp_tnc_Port* does the following:

```
static int write_snmp_tnc_Port (...)
{
   /* Declaration of local variables */
   ...

   /* Assert that the parameter is of type INTEGER */
   ...

   /* Assert that the parameter value is zero */
   ...


      if (action == COMMIT) {
         snmp_tnc_port = intval;
         execv("/usr/sbin/kissparms", argv);
```

```
    }
    return SNMP_ERR_NOERROR;
}
```

If the action performed on the object is a set operation (condition *action == COMMIT* is true), then the global variable *snmp_tnc_port* receives the new value of the object (*intval*) and the *kissparms* program is called to physically set the TNC parameter.

If the action performed on the object is a get operation, then the agent retrieves and returns the value in the memory variable *snmp_tnc_port*.

4 Conclusion

In this paper, we have presented a method to manage a particular type of network devices, named TNC, by means of a network management framework called SNMP. Precisely, we have implemented a MIB containing the parameters we intend to manage (Section 3.1). Those parameters are seen as objects that can be read and set using simple commands (Section 3.2). To facilitate the TNC management, we have implemented a new command that sets remotely and simultaneously several parameters. In Section 3.3, we have explained how we used the CMU-SNMP package to implement our application. [1]

Two problems were uncovered by this work. First, we noted that in Kiss mode, there are no ways to read and verify the actual values of TNC parameters. Consequently, the strength of the solution we propose depends on the reliability of the *kissparms* program we are applying to physically set TNC parameters. Second, there may be an inconsistency between MIB memory variables and actual TNC parameters. In fact, if the agent computer shuts down then the content of memory variables and updates of those variables are lost. When the computer reboots, these memory variables take default values which do not necessarily correspond to the real values in the TNC.

Actually, there are two possible solutions to this problem. First, values of memory variables are maintained in files. Updates are both reflected in the memory variables and in the file. When the system reboots, initial values are taken from that file. Second, when get requests are received, instead of reading values from memory variables, a procedure directly accesses the values in the TNC. The second solution is actually not feasible with the actual definition of the Kiss protocol.

[1] A copy of the C code of this project can be found at http://www.dmi.usherb.ca/~hmida/project.html.

References

[1] William Stallings, "*SNMP, SNMPv2, and CMIP. The Practical Guide to Network-Management Standards*", Addison Wesley, 1993.

[2] Marshall T. Rose, "*The Simple Book. An Introduction to Internet Management*", Prentice Hall, Second Edition, 1994.

[3] Philip R. Karn, Harold E. Price, and Robert J. Diersing, "*Packet Radio in the Amateur Service*", IEEE journal on selected areas in communications, Vol. SAC-3, NO 3, May 1985, pp. 431-439.

[4] David Perkins and Evan McGinnis, "*Understanding SNMP MIBs*", Prentice Hall, 1997.

[5] Terry Dawson, "*Linux AX25-HOWTO, Amateur Radio*", http://www.sunsite.unc.edu/mdw/HOWTO/AX25-HOWTO.html

[6] Erik Schönfelder, "*Linux CMU SNMP Project*", http://www.gaertner.de/snmp/

[7] David Guerrero, "*Network Management & Monitoring with Linux*", Linux Journal, June 1997, pp. 36-44.

[8] "*Linux Home Page*", http://www.linux.org

Appendix A: Definition of the tncControl group and its objects

```
experimental
   OBJECT IDENTIFIER ::= { internet 3 }
internetExperim
   OBJECT IDENTIFIER ::= { experimental 4711 }
tncControl
   OBJECT IDENTIFIER ::= { internetExperim 1 }

tncPort OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "The port name that is being configured."
   ::= { tncControl 1 }

tncFullDuplex OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "The TNC can be set into either full duplex
```

```
     or half duplex."
   ::= { tncControl 2 }

tncHardware OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is used to set the hardware specific
     parameters."
   ::= { tncControl 3 }

tncTxTail OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is used to set the Tx Tail time in
     milliseconds.
     Only values like 10, 20, etc are used."
   ::= { tncControl 4 }

tncPersist OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is used to set the persist value.
     This parameter is scaled to the range
     0 to 255."
   ::= { tncControl 5 }

tncSlotTime OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is used to set the slottime time
     in milliseconds. Only values like 10,
     20, etc are used."
   ::= { tncControl 6 }

tncTxDelay OBJECT-TYPE
   SYNTAX      INTEGER (1..127)
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is used to set the TX Delay time
     in milliseconds. Only values like 10,
     20, etc are used."
   ::= { tncControl 7 }
```