SNMPv1 agent for lwIP

Author: Christiaan Simons

This is a brief introduction how to use and configure the SNMP agent.

Note the agent uses the raw-API UDP interface so you may also want to

read rawapi.txt to gain a better understanding of the SNMP message handling.

0 Agent Capabilities

====================

SNMPv1 per RFC1157

This is an old(er) standard but is still widely supported.

For SNMPv2c and v3 have a greater complexity and need many

more lines of code. IMHO this breaks the idea of "lightweight IP".

Note the S in SNMP stands for "Simple". Note that "Simple" is

relative. SNMP is simple compared to the complex ISO network

management protocols CMIP (Common Management Information Protocol)

and CMOT (CMip Over Tcp).

MIB II per RFC1213

The standard lwIP stack management information base.

This is a required MIB, so this is always enabled.

When builing lwIP without TCP, the mib-2.tcp group is omitted.

The groups EGP, CMOT and transmission are disabled by default.

Most mib-2 objects are not writable except:

sysName, sysLocation, sysContact, snmpEnableAuthenTraps.

Writing to or changing the ARP and IP address and route

tables is not possible.

Note lwIP has a very limited notion of IP routing. It currently

doen't have a route table and doesn't have a notion of the U,G,H flags.

Instead lwIP uses the interface list with only one default interface

acting as a single gateway interface (G) for the default route.

The agent returns a "virtual table" with the default route 0.0.0.0

for the default interface and network routes (no H) for each

network interface in the netif\_list.

All routes are considered to be up (U).

Loading additional MIBs

MIBs can only be added in compile-time, not in run-time.

There is no MIB compiler thus additional MIBs must be hand coded.

Large SNMP message support

The packet decoding and encoding routines are designed

to use pbuf-chains. Larger payloads than the minimum

SNMP requirement of 484 octets are supported if the

PBUF\_POOL\_SIZE and IP\_REASS\_BUFSIZE are set to match your

local requirement.

1 Building the Agent

====================

First of all you'll need to add the following define

to your local lwipopts.h:

#define LWIP\_SNMP 1

and add the source files in lwip/src/core/snmp

and some snmp headers in lwip/src/include/lwip to your makefile.

Note you'll might need to adapt you network driver to update

the mib2 variables for your interface.

2 Running the Agent

===================

The following function calls must be made in your program to

actually get the SNMP agent running.

Before starting the agent you should supply pointers

to non-volatile memory for sysContact, sysLocation,

and snmpEnableAuthenTraps. You can do this by calling

snmp\_set\_syscontact()

snmp\_set\_syslocation()

snmp\_set\_snmpenableauthentraps()

Additionally you may want to set

snmp\_set\_sysdescr()

snmp\_set\_sysobjid() (if you have a private MIB)

snmp\_set\_sysname()

Also before starting the agent you need to setup

one or more trap destinations using these calls:

snmp\_trap\_dst\_enable();

snmp\_trap\_dst\_ip\_set();

In the lwIP initialisation sequence call snmp\_init() just after

the call to udp\_init().

Exactly every 10 msec the SNMP uptime timestamp must be updated with

snmp\_inc\_sysuptime(). You should call this from a timer interrupt

or a timer signal handler depending on your runtime environment.

An alternative way to update the SNMP uptime timestamp is to do a call like

snmp\_add\_sysuptime(100) each 1000ms (which is bigger "step", but call to

a lower frequency). Another one is to not call snmp\_inc\_sysuptime() or

snmp\_add\_sysuptime(), and to define the SNMP\_GET\_SYSUPTIME(sysuptime) macro.

This one is undefined by default in mib2.c. SNMP\_GET\_SYSUPTIME is called inside

snmp\_get\_sysuptime(u32\_t \*value), and enable to change "sysuptime" value only

when it's queried (any function which need "sysuptime" have to call

snmp\_get\_sysuptime).

3 Private MIBs

==============

If want to extend the agent with your own private MIB you'll need to

add the following define to your local lwipopts.h:

#define SNMP\_PRIVATE\_MIB 1

You must provide the private\_mib.h and associated files yourself.

Note we don't have a "MIB compiler" that generates C source from a MIB,

so you're required to do some serious coding if you enable this!

Note the lwIP enterprise ID (26381) is assigned to the lwIP project,

ALL OBJECT IDENTIFIERS LIVING UNDER THIS ID ARE ASSIGNED BY THE lwIP

MAINTAINERS!

If you need to create your own private MIB you'll need

to apply for your own enterprise ID with IANA: http://www.iana.org/numbers.html

You can set it by passing a struct snmp\_obj\_id to the agent

using snmp\_set\_sysobjid(&my\_object\_id), just before snmp\_init().

Note the object identifiers for thes MIB-2 and your private MIB

tree must be kept in sorted ascending (lexicographical) order.

This to ensure correct getnext operation.

An example for a private MIB is part of the "minimal Unix" project:

contrib/ports/unix/proj/minimal/lwip\_prvmib.c

The next chapter gives a more detailed description of the

MIB-2 tree and the optional private MIB.

4 The Gory Details

==================

4.0 Object identifiers and the MIB tree.

We have three distinct parts for all object identifiers:

The prefix

.iso.org.dod.internet

the middle part

.mgmt.mib-2.ip.ipNetToMediaTable.ipNetToMediaEntry.ipNetToMediaPhysAddress

and the index part

.1.192.168.0.1

Objects located above the .internet hierarchy aren't supported.

Currently only the .mgmt sub-tree is available and

when the SNMP\_PRIVATE\_MIB is enabled the .private tree

becomes available too.

Object identifiers from incoming requests are checked

for a matching prefix, middle part and index part

or are expanded(\*) for GetNext requests with short

or inexisting names in the request.

(\* we call this "expansion" but this also

resembles the "auto-completion" operation)

The middle part is usually located in ROM (const)

to preserve precious RAM on small microcontrollers.

However RAM location is possible for a dynamically

changing private tree.

The index part is handled by functions which in

turn use dynamically allocated index trees from RAM.

These trees are updated by e.g. the etharp code

when new entries are made or removed form the ARP cache.

/\*\* @todo more gory details \*/