SNMP SMI
Structure of Management Information
Network Mgmt/Sec.
Outline

◆ ASN.1 short intro
  – BER
  – grammar/types

◆ SMI
  – types and application types
  – MACROs
  – tables/examples
jrb comment:

- this will seem like “much ado about nothing”
  - painful, but useful taken in medicine-sized doses
- formal definition of syntax
- form before content ...
- Master Kung said: “the superior person defines his/her terminology first”
Abstract Syntax Notation Dot One

- a formal grammar used for defining
  - packet encodings
    - ISO/OSI packet types (network layer and up)
      - CLNP - ISO IP equivalent
    - IETF SNMP Packet Data Units (app layer)
  - data definition language
    - X.500 data
    - RSA Public-Key Crypto Standards

Jim Binkley SNMP data (variable binding part)
for more information see:

- RSA “A Layman’s Guide to a Subset of ASN.1, BER, and DER
  - Kaliski Jr., 1993
- Stallings, SNMP, etc.
  - Appendix B
ASN consists of two parts

- a formal grammar that consists of productions
  - $A ::= B$ (definition of types and instances)
  - syntax sugar
    » e.g., comments -- this is a comment
- and rules for encoding the constructs into binary data
  - Basic Encoding Rules (BER)
- much like how a compiler takes a programming language and produces object (binary) data...

(duh)

Jim Binkley
syntax sugar

- comments
  -- BLAH BLAH
  -- BLECH FOO!
- ::= for assignment (e.g., derived types)
  - OctetStringType ::= OCTETSTRING
- identifiers begin with a lowercase letter
- type/module references begin with uppercase
- built-in types all upper case
- identifiers/type names can have digits/hypens

Jim Binkley
Basic Encoding Rules

- ISO 8825
- DER, in X.509, Distinguished Encoding, gives one way to define BER values only

how to encode/decode values of ASN.1 types into/from binary

basic idea: tag, length, value

roughly 1 byte tag (what is it), ASN.1 type

1 byte length (how long is it)

value: the data itself as a string of bytes
Great Scott!

- SNMP is all TLVs ...
- keep in mind: mostly shipping MIB variable names (OIDs) and values back and forth
- MIB values have an amazing tendency to be:
  - integers of various sizes
  - strings “my name is Joe Bob Cisco Router”
  - and a few constructs like IP addresses, etc.
BER isn’t that simple though

- 3 methods for encoding an ASN.1 value
  - length of data and/or number of tags in tag set
- 1. primitive, definite-length
  - simple, non-string types
  - ID is tag (class and tag #) of ASN.1 type
    » 02 for INTEGER, 04 OCTET STRING (bytes)
  - length, if less than 128 can fit in one byte
  - value/contents, the ASN.1 value as byte string
    » depends on the TYPE ...
BER 2/3

2. constructed, definite-length encoding
   - can be used for strings, structured types
   - length must be known in advance via length field (hence definite-length)

3. constructed, indefinite-length encoding
   - strings, structured types, again
   - difference is length field NOT used
   - must look thru contents to find End-Of-Contents, two bytes with value 0x0000
basic simple form, and bigger tag fields

<table>
<thead>
<tr>
<th>1 byte</th>
<th>1 byte</th>
<th>1..127 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag/id</td>
<td>length</td>
<td>value</td>
</tr>
</tbody>
</table>

tag field decomposed: as one byte

<table>
<thead>
<tr>
<th>class (2 bits)</th>
<th>Prim/Con (1)</th>
<th>tag # (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>octet 1</td>
<td>octet 2</td>
<td>octet 3</td>
</tr>
</tbody>
</table>

Jim Binkley
length can be long too OR ignored (indefinite length)

<table>
<thead>
<tr>
<th>One byte length (definite)</th>
<th>Multi-byte (definite)</th>
<th>Indefinite form (length not included)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>length ( \leq 127 )</td>
<td>7 bits, length in bytes more bytes</td>
</tr>
</tbody>
</table>

Jim Binkley
ASN tag classes

- basic idea is that there are universal tags and possible application-derived (non-universal, local interest) tags
- 00 - universal
- 01 - application
- 10 - context specific (more limited context than app)
- 11 - private (no standards)
some universal class/tags

- 1 - BOOLEAN
- 2 - INTEGER (2’s complement)
- 3 - BIT STRING
- 4 - OCTET STRING (aka bytes)
- 5 - NULL
- 6 - OBJECT IDENTIFIER
- 7 - Object descriptor (human string - explain object)
- 9 - REAL
- 16 - SEQUENCE and /SEQUENCE-OF
- 17 - SET and SET-OF
- 27 - GeneralString
types may be

- **simple** - defined in terms of values
  - INTEGER (say 1..127 or whatever)
- **structured** - defined in terms of other types
  - like a C structure, PERL associative array
  - or set in other programming languages
  - in ASN, structures may have structures (but not in SNMP...)
  - structures made up of **component** types
some explanation

- **OBJECTIDENTIFIER**
  - tree-based name scheme for all ASN objects
  - value is sequence of small integers

- **SEQUENCE** - like a C structure
  - ordered list of types from simpler types

- **SEQUENCE OF** - like an associative array
  - index scheme may be “interesting”
  - all component types the same

- **SET** - basically like sequence but not ordered
some BER examples (from Stallings)

- 02, 02, FF 7F (INTEGER, -129)
- 04,04, 01 02 03 04 (OCTET STRING, value is 01020304)
- 05 00 (NULL)
- 1A 05 4A 6F T3 65 73 (CharacterString, 5 bytes of “Jones”)
- 30 06, 02 01 03, 02 01 08 (SEQUENCE of two INTEGRERs)
ASN module structure

- must start with module definition
- module-name DEFINITIONS ::= BEGIN
  IMPORTS section
  EXPORTS section
  Assignments (productions) section
  End
- IMPORTS - from other modules
- EXPORTS - definitions that can be used by other modules
rfc1213.txt (aka MIB-II)

p. 12 starts with this:
RFC1213-MIB DEFINITIONS ::= BEGIN
IMPORTS
  mgmt, ...IpAddress, Counter, Gauge,
  TimeTicks FROM RFC1155-SMI

  OBJECT-TYPE FROM RFC-1212;
then some assignments ...
  (some :->)
the term “tag” may be over-used in ASN.1
new types may be defined from old types
types may be called tagged types to create sub-name conventions
implicit - replace old tag with new class/tag number (derivation)
explicit - add new tag to create one component STRUCTURE type (encapsulation)
type creation - example in ASN.1

- TelephoneNumber ::= [APPLICATION 3] IMPLICIT INTEGER (-range..+range)
- meaning a new tag/type (implicit) has the application class, and is an integer
**CHOICE, ANY**

- data types without any tagging (no BER)
- **CHOICE** when defined must include list of alternative types
  - only one will actually be used at runtime
  - e.g., SNMP PDU types include **CHOICE** of get-request, get-next-request, set-request, etc.
- ANY is used when can’t know type in advance

Jim Binkley
ASN MACRO facility exists

- allows designer to arbitrarily extend ASN syntax to define new types/values
- very limited use in SNMP (we’ll see it RSN)
- form: `<macroname>` MACRO ::= BEGIN
  TYPE NOTATION/s ::= new types
  VALUE NOTATION/s ::= new value type productions ...
END

Jim Binkley
SMI - Structure of Management Information

- ASN.1 is vast untamed grammar mechanism
- SNMP seeks to simplify to smaller set of types/constructs/and a macro or two
- need simplicity in order to have a shot at interoperability between managers/agents
overview

- MIB tree structure
- SNMP types
  - universal and application-wide
  - object types/OBJECT-TYPE macro
- tables
- a few examples
MIB tree structure (again)

- MIB variables named thru rooted tree
- 1.3.6.1.2.2.1(system) etc...
- iso(1).org(3).dod(6).internet(1) gets us to:
  - directory - reserved for X.500
  - mgmt - IAB approved objects (MIB-2)
  - experimental - used to id objects used in Inet experiments
- private - used to id private-enterprise objects
top part of OID tree

- iso(1)
- org(3)
- dod(6)
- internet(1)

- directory(1) X.500
  - mgmt(2)
    - mib-2(1)
    - experimental(3)
    - private(4)
      - enterprises(1)
types in SNMP

- basically integers/strings/null/object id, some application types, and tables (reflected in sequence/sequence-of)
- tables are simple objects (barring their index/walking mechanisms)
  - cannot have tables nested in tables
- full ASN syntax definition is cut back quite a bit

Jim Binkley
another way of looking at it:

- ASN basic types **NOT** used include:
  - BOOLEAN
  - BIT STRING
  - ObjectDescriptor
  - EXTERNAL
  - REAL
  - ENUMERATED
  - SET and SET OF
fundamentally includes:

- INTEGER
- OCTET STRING (aka bytes ...)
- OBJECT IDENTIFIER
- SEQUENCE (one tuple)
- SEQUENCE OF (ordered set of tuples)
Application types

- NetworkAddress - CHOICE of addr, but only IpAddress at this point
- IpAddress - 4 bytes OCTET STRING
- Counter (Counter32) - non-neg int, 2*32-1
- Gauge - non-neg int (can go down)
- TimeTicks - # ticks in 1/100 second since boot
- Opaque - OCTET STRING, no attributes
application types, cont

- **Counter** - a counter may be incremented but not decremented. Rolls over to zero at max
  - example: interface bytes in
- **Gauge** - may increase or decrease. If max, gets stuck (latches)
  - example: temperature
- **timetick** - note that it is relative, no notion like NTP/universal time
from rfc1155

- IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (size 4)
- Counter ::= [APPLICATION 1] IMPLICIT INTEGER (0..4294967295)
- Gauge ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)
- note: snmpv2 defines Counter32/Counter64, Gauge32/Gauge64
OBJECT-TYPES

- A MIB is a set of OBJECT-TYPES
- Each defines a kind of managed object - via a syntax description
- An object instance is a particular instance bound to a specific value
- The OBJECT-TYPE macro is used to define all MIB values
ASN syntax:

- **OBJECT-TYPE MACRO** ::= 
  BEGIN
  TYPE NOTATION ::= “SYNTAX” type (TYPE ObjectSyntax)
  “ACCESS” Access
  “STATUS” Status
  VALUE NOTATION ::= value (VALUE ObjectName)
  ...
  END

- some variable of some type with some value and a couple of attributes (access/status)
continued

- **Access includes:**
  - read-only
  - read-write
  - write-only
  - not-accessible (can’t read or write)

- **Status includes:**
  - mandatory
  - optional
  - obsolete (don’t have to do it)
  - deprecated (implemented but doomed)

Jim Binkley
continued

- note definition of derived type
- DisplayString ::= OCTET STRING (0..255)
- Indices (used with table rows) may include CHOICE
  - number INTEGER
  - string OCTET STRING
  - object OBJECT IDENTIFIER
  - address NetworkAddress
  - IpAddress IpAddress
1.3.6.1.2.1.1.1 (an example)

- `mib-2(1).system(1).sysDescr(1)`:
  - `sysDescr` OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION "A textual description of the entity. This value should include the full name and version identification of the systems’ hardware type ... yadda yadda”.

::= { system 1 }

Jim Binkley
constructed types give us TABLE

- **row**: type with form:
  
  `<row> ::=`  
  
  `SEQUENCE {  
    <type>, <type>, type }`

- `<table> ::=`  
  
  `SEQUENCE OF <row>`

- we get simple non-nestable 2-d table
- IndexPart defines index mechanism for row

Jim Binkley
example (logic not syntax garp):

- mib-2.interfaces has ifTable (table) made up of ifEntry (row)
- each ifEntry defines an interface with 22 component types
- e.g.,
  
  ifTable
  ifEntry
  ifIndex INTEGER -- unique per i/f
  ifDesc DisplayString
  ifType  INTEGER (e.g., enet)
  ifMtu INTEGER
  etc ...

Jim Binkley