EEM.NSM Lab Coursework

Introduction

Outline

- Communicate with a specified host via SNMP
- Purpose: learn how to use SNMP to perform simple network management tasks
- Part One
  - Retrieve the TCP Connection Table from a specified host's MIB
  - Reproduce the table on the screen in a readable, aligned format
- Part Two
  - Retrieve the values of a pair of counters from a specified host's MIB
  - Calculate the counters' Uniformly-Weighted Moving Average values
- Write a brief report describing your work and results

Guidelines

- **Get familiar with programming in Java**

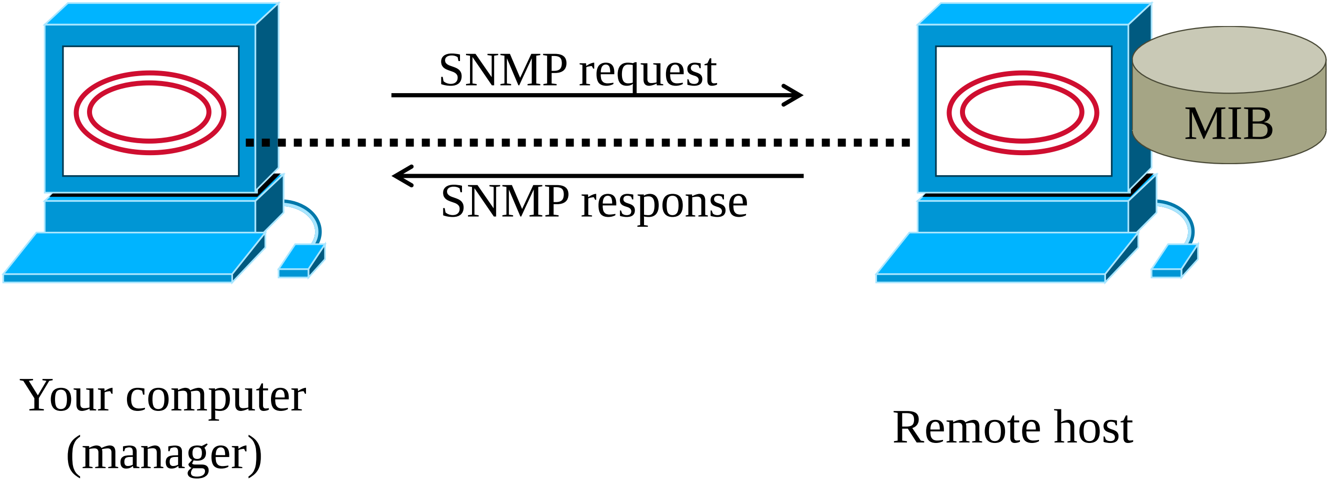
- **Use of AdventNetSnmp package**

  AdventNetSnmp.jar

- **Two sample Java programs are provided**
  - **Send GET or GETNEXT requests**
  - **SnmpGet.java**
  - **SnmpGetNext.java**

Basic Knowledge

- ❖ **How does SNMP work when retrieving information?**
  - **Connect => Request => Receive => Process => Disconnect**
  - **The remote host must have an SNMP agent running on it**



Basic Knowledge (cont'd)

- **How does a MIB store its information?**
  - **Tree-like hierarchical structure**
  - **Object IDentifier (OID)**
  - **Each MIB entry is identified by:**
    - A dot, each indicating a tree level in the MIB
    - A number, indicating its position at the current tree level
- **A real example in RFC-1213 MIB**

.1 (iso) --- Level 1

.1 (std) --- Level 2

.2 (member-body)

.3 (org)

.1 --- Level 3

…

.6 (dod) ---- the 6th entry at level 3, hence OID is .1.3.6

1. Establishing Connection

- ❖ **Establish (open) an SNMP session with a host**
  - **Host is specified with the variable “remoteHost”**

```
SnmpAPI api = new SnmpAPI();  // Create a new SNMP API
api.start();
api.setDebug( false );
// Create a new SNMP session in the API
SnmpSession session = new SnmpSession(api);

try {session.open(); }    // Open the SNMP session
catch (SnmpException e ) {
System.err.println("Error opening socket: "+e);
}       // In case the session cannot be opened

session.setPeername(remoteHost);  // Specify the remote host
```

2. Send an SNMP Request

- Each SNMP request is encapsulated in a PDU (Protocol Data Unit)
- Two things need to be set before a PDU is sent
  - Command type (Get? GetNext? or others?)
  - OIDs: one or more OIDs are bound to a PDU

```
SnmpPDU pdu = new SnmpPDU();  // Create a new PDU
pdu.setCommand( api.GETNEXT_REQ_MSG );  // Set its command

for (int i=1; i < args.length; i++)
{
       SnmpOID oid=new SnmpOID(OID); // Create an OID
       pdu.addNull(oid);      // Bind the OID to the PDU
}
try {pdu = session.syncSend(pdu);}    // Send the PDU
catch (SnmpException e) {System.err.println("Error sending
SNMP request: "+e);}   // In case the PDU cannot be sent
```

3. SNMP Response

- **What will the remote host send back to us?**
- **Requested OID appended with row identifier**
  - **Used to identify a unique SNMP response**
  - ***i.e.*, variable bindings**
- **We will come back later, after we finish explaining the TCP connection table.**

TCP Connection Table

- Every host has a table, whose entries contains all TCP connections it has made with other hosts.
- Five columns
  - Connection state (go to here for details)
  - Local address
  - Local port
  - Remote address
  - Remote port
- An example

ConnState	LocAddr	LocPort	RemoteAddr	RemotePort
3	132.168.23.34	23	145.21.56.153	78
2	124.112.45.78	80	134.244.21.45	90
2	132.254.1.2	21	131.24.45.160	65

NOTE: the five columns' OIDs are listed on the guideline webpage.

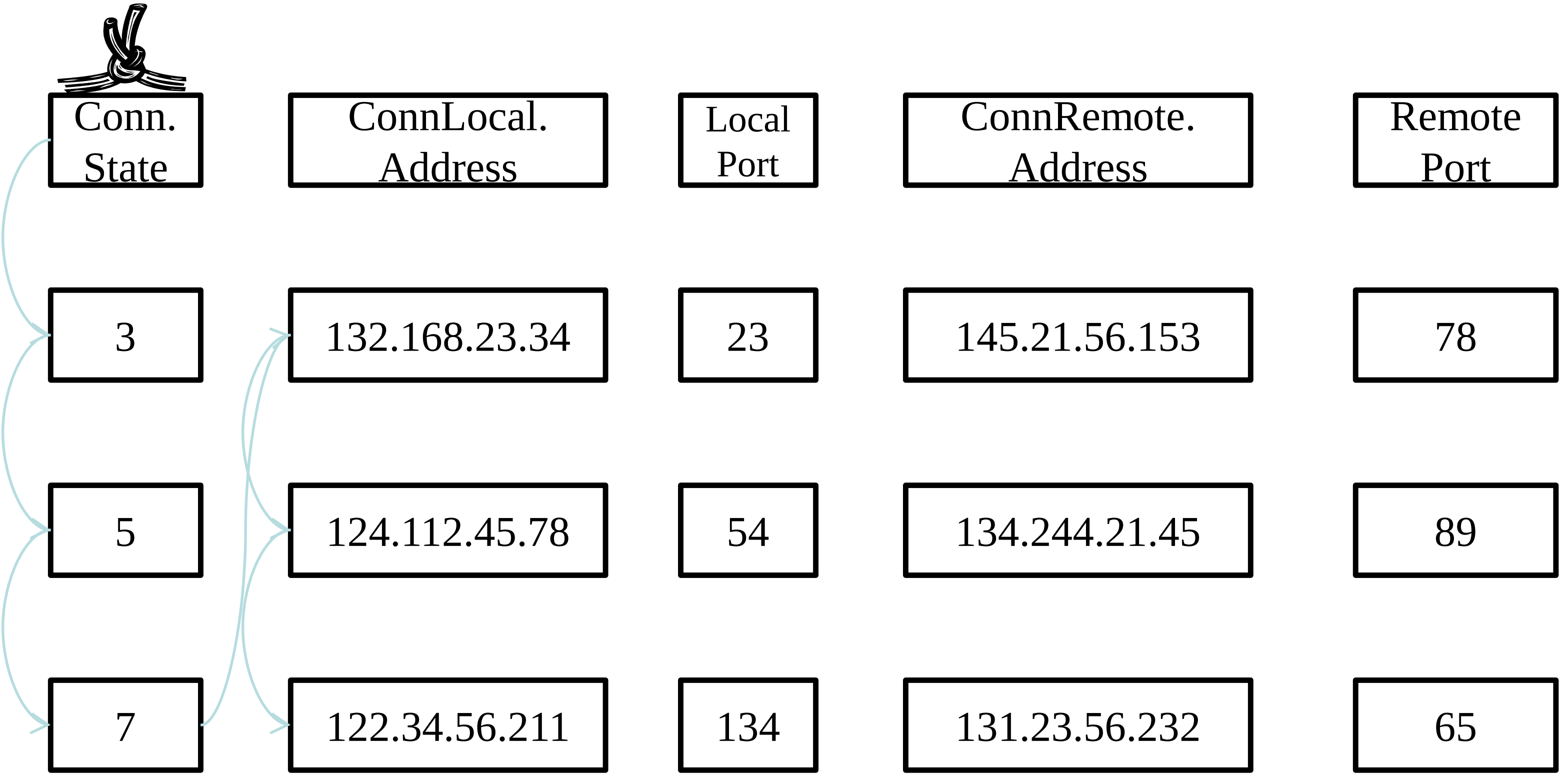
TCP Connection Table

- How should we traverse the table efficiently?
- First of all: use GetNext request
- How many OIDs should we bind to the PDU?

- Option 1: bind one OID at a time
  - Inefficient: will go row by row, then column by column
- Option 2: bind all five OIDs altogether
  - Efficient: row by row, then done!

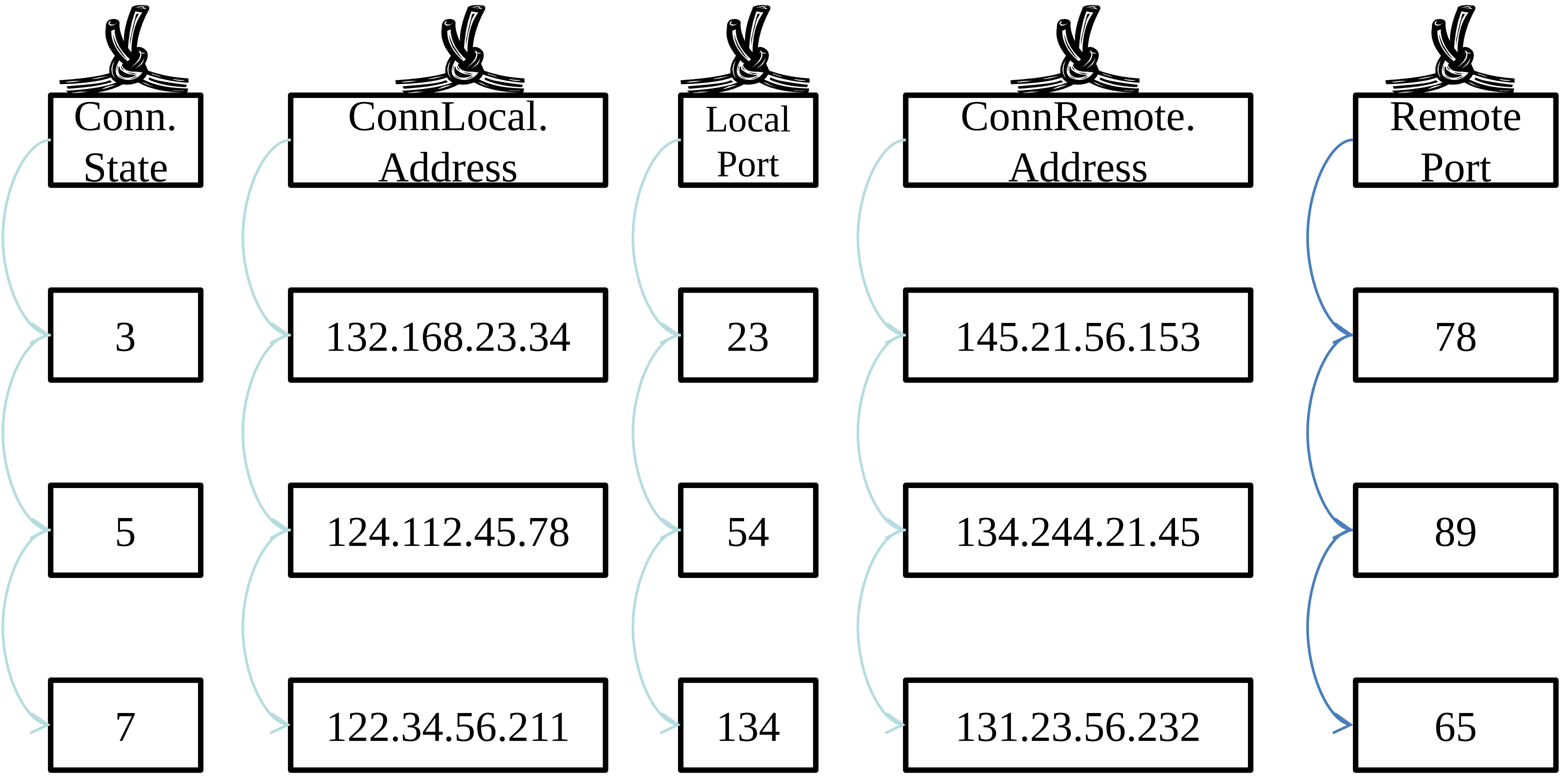
ConnState	LocAddr	LocPort	RemoteAddr	RemotePort
3	132.168.23.34	23	145.21.56.153	78
2	124.112.45.78	80	134.244.21.45	90
2	132.254.1.2	21	131.24.45.160	65

Option 1 - bind one OID at a time



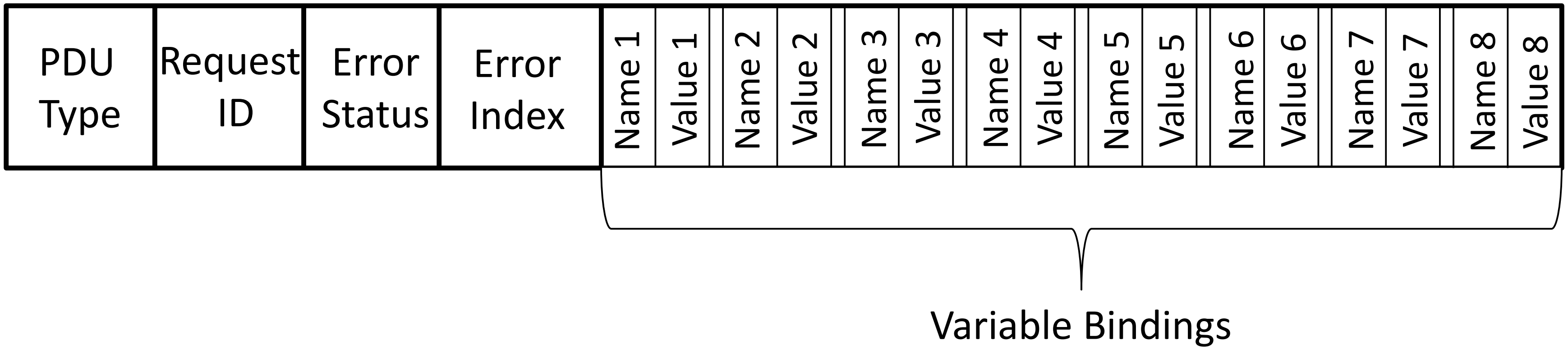
Conn. State	ConnLocal. Address	Local Port	ConnRemote. Address	Remote Port
3	132.168.23.34	23	145.21.56.153	78
5	124.112.45.78	54	134.244.21.45	89
7	122.34.56.211	134	131.23.56.232	65

Option 2: bind all five OIDs together



Conn. State	ConnLocal. Address	Local Port	ConnRemote. Address	Remote Port
3	132.168.23.34	23	145.21.56.153	78
5	124.112.45.78	54	134.244.21.45	89
7	122.34.56.211	134	131.23.56.232	65

PDU Packet



PDU Type

Request ID

Error Status

Error Index

Name 1

Value 1

Name 2

Value 2

Name 3

Value 3

Name 4

Value 4

Name 5

Value 5

Name 6

Value 6

Name 7

Value 7

Name 8

Value 8

Variable Bindings

3. SNMP Response (cont'd)

- The SNMP response contains requested OIDs appended with row identifiers.
- Consider a request PDU with the first two columns' OIDs is sent to retrieve the first row of the TCP connection table
- Request:

```
OID: OID1
OID: OID2
```

- Response (variable bindings):

```
OID: OID1.132.168.23.34.23.145.21.56.153.78
Value: 3
OID: OID2.132.168.23.34.23.145.21.56.153.78
Value: 132.168.23.34
```

- Coloured parts: row identifier (identify the first row in the TCP connection table)

4. Process the SNMP Response

- ❖ **Print out the variable bindings**
  - – **Result will look like the previous slide**

```
System.out.println(pdu.printVarBinds());
```

- ❖ **There are more Java methods that can process an SNMP response**
  - – **Required to finish part one of the coursework: reproduce the TCP connection table in a readable, aligned format**
  - – **You have to investigate them by yourself**

5. Disconnect the SNMP Session

- **Do not forget to terminate the SNMP session at the end of your program**

```
session.close();
api.close();
```

Requirements: Part One

- Objective: reproduce the TCP connection table of a specified host
  - List of hosts can be found on the guideline webpage
- All columns must be aligned (in any way you like)
- ALL rows of the table must be displayed, not one more, not one less.
- Follow the efficient OID-binding technique we explained earlier.
- You should not reproduce the table by manipulating results from printVarBinds() *i.e.*, the sample program.
- Open only one SNMP session

Requirements: Part Two

- **Objective: calculate Uniformly-Weighted Moving Average (UWMA) of a pair of specified counters in a specified host's MIB**
  - Details of the counters can be found on the guideline webpage
- **Parameters**
  - OIDs whose values are calculated
  - Polling period (should be > 6 seconds)
  - Moving average's window size
- **Your program should display:**
  - Parameter values
  - Clearly show how the UWMA results are calculated (sample values it used etc., you do not need to show the arithmetic operations)
  - UWMA results of each window
- **The program should keep running until manually terminated**

Possible Poll Implementations

- **Using threads (recommended for C users) – minimal example**

```
public class Example {
public static void main(String args[]) throws InterruptedException {
    int count = 0;
    while (count < 10){
        System.out.println("Hello World " + count);
        count++;
        Thread.sleep(1000);
}}}
```

- **Using timers**

- **Using AdventNet SnmpPoller**

General Requirements

- **ALL parameters MUST NOT be hard-coded, and should be passed as command-line arguments.**
  - **Part one: remote host, OIDs in the table**
  - **Part two: remote host, OIDs, polling period, window size**
  - **You can find how to pass arguments in the guideline webpage.**

- **Provide the best flexibility you can offer in your program.**
  - **A few examples**
    - In part one, what if we only want to see four columns of the table?
    - In part two, what data structure do you use to store the UWMA samples?

Report Structure

- ❖ **Introduction (half page)**
- ❖ **A brief summary of relevant key points of SNMP (short notes are suitable)**
- ❖ **An outline of your program design, in the form of pseudo-code or a flowchart; together with brief notes on any key features of the software; and on any issues encountered during your work together with their resolution**
- ❖ **Relevant screenshots of your program's output**
- ❖ **Concluding comments (half page)**

- ✓ **The report should be brief, of the order of 6-8 pages**
- ✓ **It is not necessary to include a copy of your final software**

Marking Criteria

- **Software:**
  - ✓ **Functions implemented correctly**
  - ✓ **Quality of output display (correct output of TCP connection table; correct display of UWMA parameters and calculation of UWMA)**
  - ✓ **Code clarity and appropriate commenting**
  - ✓ **Your ability to discuss your design and implementation of your code**
- **Report:**
  - ✓ **Quality of SNMP discussion**
  - ✓ **Quality of software description**
  - ✓ **Results / screenshots**
  - ✓ **Quality of English**

Appendix

- For the first part, the following five OIDs will be used:
  .1.3.6.1.2.1.6.13.1.1
  .1.3.6.1.2.1.6.13.1.2
  .1.3.6.1.2.1.6.13.1.3
  .1.3.6.1.2.1.6.13.1.4
  .1.3.6.1.2.1.6.13.1.5

- For the second part, the following three OIDs will be used:

.1.3.6.1.2.1.6.10 and .1.3.6.1.2.1.6.11 *(tcpInSegs and tcpOutSegs)*

.1.3.6.1.2.1.2.2.1.10 and .1.3.6.1.2.1.2.2.1.16 *(ifInOctets and ifOutOctets)*
.1.3.6.1.2.1.4.9 and .1.3.6.1.2.1.4.10 *(ipInDelivers and ipOutRequests)*

Appendix

- **Hostname : feps-teach01**
- **Set Snmp Community: teachinglabs**

- **Add AdventNetSnmp.jar**

AdventNetSnmp.jar

Right click your Java project →Build Path ->select external archives -> select the AdventNetSnmp.jar ->OK