# SNMP PROGRAMMING PROJECT

CS158B ASSIGNMENT 2

Sajay Shah & Brandon Feist
SJSU

# Table of Contents

# Functional Specification

This section of the report will cover what a user will observe when using the program.

The goal of this project is to design and implement a program that monitors the traffic in a device.

More specifically:

1. Discover the device interfaces
2. Discover what devices they are connected to (at the IP level)
3. Monitor the traffic on the device interfaces

## Functional Specification Requirements

Program functional requirements can be broken down into **2** categories:

1. Input
2. Output

The following must occur after program has successfully compiled and user has correctly installed and configured their NetSNMP agent.

### Program Input Requirements

User must provide:

1. Time interval (seconds) between samples.
    a. Must be a positive integer.
    b. If it is a positive decimal, it will be converted to an integer.
2. Number of samples to take.
    a. Must be a positive integer
    b. If it is a positive decimal, it will be converted to an integer.
3. Community name.
    a. This should be properly configured during NetSNMP agent setup.
4. Hostname of the agent.
    a. This will be a parameter for method calls using the NetSNMP API.

## Program Output Requirements

The program will print user friendly tables if no errors were encountered. If the error encountered was handled by the program, a descriptive message will output and the program will terminate.

1. The first table will have 2 columns and *n* rows for the number of devices discovered.
    a. Each row will have an interface number and IP address of that device.
2. The second table will have 2 and *m* rows for the number of neighbors discovered for each device.
    a. Each row will have the interface number and IP address of that neighbor.
3. The next *n* tables will display IN and OUT traffic for each device discovered.
    a. Each table will have 4 columns and *numSamples* rows.
    b. Traffic data will be rounded to whole numbers and displayed in *KB/s.*

# Design Specification

This section of the report will cover the program design.

## System Overview

The program written for this project uses the NetSNMP API and MIB-2 definitions to:
1. Discover the device interfaces and their IP addresses
2. Discover device interface neighbors and their IP addresses
3. Monitor the traffic on the device interfaces

## Design Considerations

This subsection will address certain issues that must be resolved before creating a design solution.

### Assumptions and Dependencies

In order to successfully run this program, the user must have correctly configured NetSNMP on their device. The user must know their agent's community name. An invalid community name will cause the program to have a segmentation fault.

### General Constraints

NetSNMP is known to have poor API documentation. Some veteran NetSNMP users would state that the *Simple* in ***S****imple* ***N****etwork* ***M****anagement* ***P****rotocol* is a lie.

Given how the user of this program has set up NetSNMP on their device, the program may or may not be able to obtain the desired information.

This program has been written and tested on a Ubuntu environment. The program may or may not produce the same output if run on a different environment.

### Goals

This program will ultimately accomplish two things:
1. Obtain interface, neighbor, and traffic information using MIB-II variables.
2. Calculate traffic information based off the time delta the user provides.

## Guidelines

The following MIB-II objects will be used.

| MIB-II Object Name | MIB-II Object ID | Description |
|---|---|---|
| ipAdEntAddr | 1.3.6.1.2.1.4.20.1.1 | Discover IP Address |
| ipAdEntIfIndex | 1.3.6.1.2.1.4.20.1.2 | Discover ifIndex |
| ipNetToMedia ifIndex | 1.3.6.1.2.1.4.22.1.1 | Discover neighbor ifIndex |
| ipNetToMedia Net Address | 1.3.6.1.2.1.4.22.1.3 | Discover neighbor IP Address |
| ifInOctets | 1.3.6.1.2.1.2.2.1.10.x | Retrieve inOctets of an interface. (x = ifIndex) |
| ifOutOctets | 1.3.6.1.2.1.2.2.1.16.x | Retrieve outOctets of an interface. (x = ifIndex) |

## System Architecture

This subsection will describe a high level overview of the methods used for the SNMP Agent.

### `createSNMPSession`

1. Establishes a SNMP session using the provided version, community, and hostname.
2. Returns session handle.

### `snmpGet`

1. Mainly takes advantage of `snmp_synch_response` convenience routine that will send the request, wait for the response and process it before returning.
2. Returns a `struct` containing PDU Response variables

### `snmpGetNext`

1. Similar to `snmpGet` except the PDU used for the `snmp_synch_response` is of type `SNMP_MSG_GETNEXT`.
2. Returns a `struct` containing PDU Response variables

### `snmpWalk`

1. A `while` loop that calls `snmpGetNext` until the methods return value shows that the walk is to be terminated.
2. Returns `0`.

### getTrafficData

1. Calling `snmpGet` to obtain `inOctet` and `outOctet` values returns a `struct` with a pointer to the actual data value. This method takes the `struct` and copies over the desired data value to memory so it can be successfully converted from `octets` to a data velocity, which for this program is kilobytes per second.
2. Returns `long` value of `inOctet` or `outOctet`.

### trafficDeltaToKB

1. Takes two `octet` values and time interval.
2. `Octet` value will reset to `0` and continue counting until it reaches the max value of $2^{32} - 1$, as a result, the two input values have to be compared to determine whether a reset occurred during the time interval.
3. Returns kilobytes per second value.

## Program Interface

This subsection will describe the input and output of the program.

### Input

1. The program will immediately terminate if the correct number of parameters are not provided.
2. The program will immediately terminate if both the time interval and number of samples are not positive values.

### Output

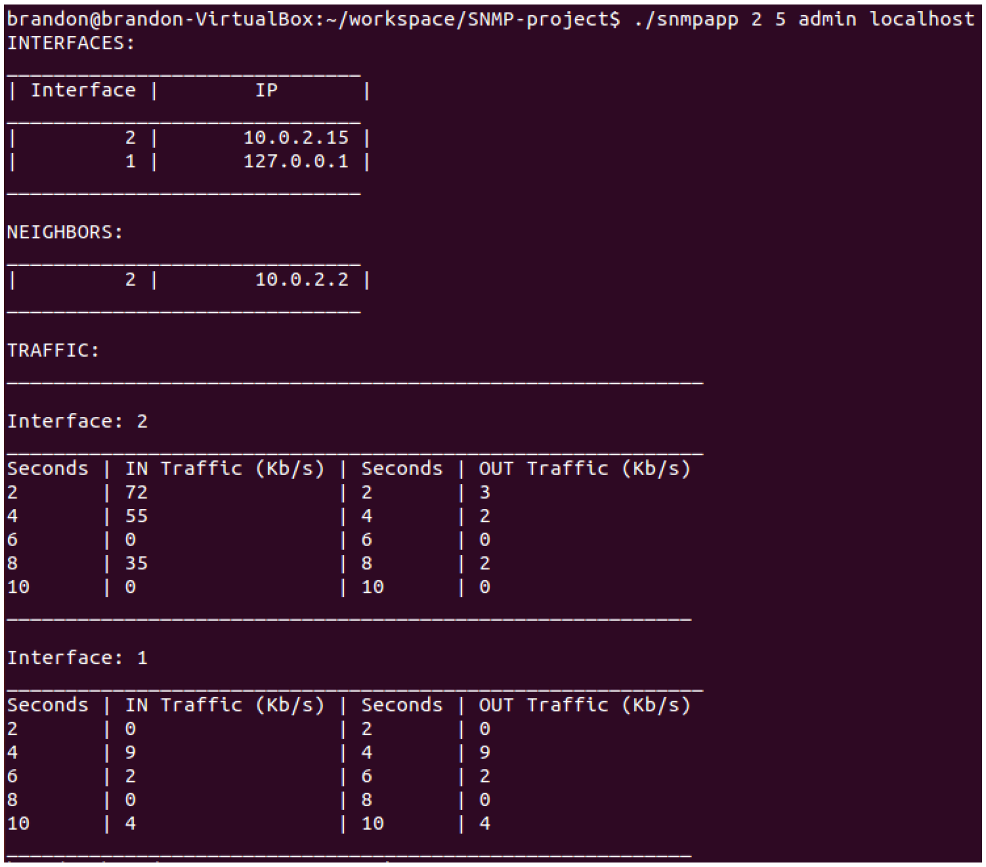
1. User may notice the program generates temporary `.`txt files during its execution.
    a. It is important that the user does not interact with these files.
        i. They are used to store `snmpwalk` results which are needed for later program operations.
    b. These files will be automatically deleted from the system during the `main` method's clean up phase.
2. The program will output data in console throughout its execution. Refer to the **Program Output Requirements** section of the **Functional Specification** portion of this report for more information.

## Program Usage

This subsection will describe how to compile and run the program.

1. Open terminal.
2. Navigate to directory where `snmpproject.tar.gz` is located
3. Run the following command: `tar xvzf snmpproject.tar.gz`
4. Run the following command: `sudo make`
    a. If the command results in an error you can compile the C code using the following command:
    `gcc -o snmpproject.c –L/path-to-net-snmp-folder/snmplib/.libs -lnetsnmp`
    b. In our case the compile command was
    `gcc -o snmpproject.c –L/home/brandon/net-snmp-5.7.3/snmplib/.libs -lnetsnmp`
5. Run the following command: `./snmpproject` *w x y z*
    a. `w` = time interval (positive value)
    b. `x` = number of samples (positive value)
    c. `y` = NetSNMP community name (valid community name)
    d. `z` = hostname (valid hostname)

## Output



```
brandon@brandon-VirtualBox:~/workspace/SNMP-project$ ./snmpapp 2 5 admin localhost
INTERFACES:
______________________________
| Interface |        IP       |
______________________________
|         2 |       10.0.2.15 |
|         1 |       127.0.0.1 |
______________________________

NEIGHBORS:
______________________________
|         2 |        10.0.2.2 |
______________________________

TRAFFIC:
____________________________________________________________

Interface: 2
____________________________________________________________
Seconds | IN Traffic (Kb/s)  | Seconds | OUT Traffic (Kb/s)
2       | 72                 | 2       | 3
4       | 55                 | 4       | 2
6       | 0                  | 6       | 0
8       | 35                 | 8       | 2
10      | 0                  | 10      | 0
___________________________________________________________

Interface: 1
____________________________________________________________
Seconds | IN Traffic (Kb/s)  | Seconds | OUT Traffic (Kb/s)
2       | 0                  | 2       | 0
4       | 9                  | 4       | 9
6       | 2                  | 6       | 2
8       | 0                  | 8       | 0
10      | 4                  | 10      | 4
___________________________________________________________
```

# Testing

This section will describe how the program is tested.

Assuming that the file has been compiled successfully:

| Test case | Output |
|---|---|
| Invalid number of arguments: Too few arguments. | The correct number of parameters were not supplied: (time interval \| number of samples \| community \| hostname) |
| Invalid number of arguments: Too many arguments | The correct number of parameters were not supplied: (time interval \| number of samples \| community \| hostname) |
| Invalid parameter type: Time interval non-positive value | Please ensure that both time interval and number of samples are positive values. |
| Invalid parameter type: Number of samples non-positive value | Please ensure that both time interval and number of samples are positive values. |
| Invalid parameter type: Time interval and number of samples non-positive values | Please ensure that both time interval and number of samples are positive values. |
| Invalid community name | Segmentation fault (core dumped). There's nothing the program can do to check if community name is valid. |
| Invalid hostname | getaddrinfo: "hostname" Temporary failure in name resolution.<br>ack: Unkown host ("hostname")<br>The session was not able to open |

Program will terminate after error message output.

# Extra Credit

The goal of the extra credit portion of this project is to analyze the accuracy of the traffic estimation this program provides.

## Design

In order to understand the accuracy of this `snmpprogram`, we will compare results for `eth0` in/out traffic with those from `bmon`, a Linux traffic monitoring tool. Data from `eth0` is used since localhost is a loop where in/out data is identical.

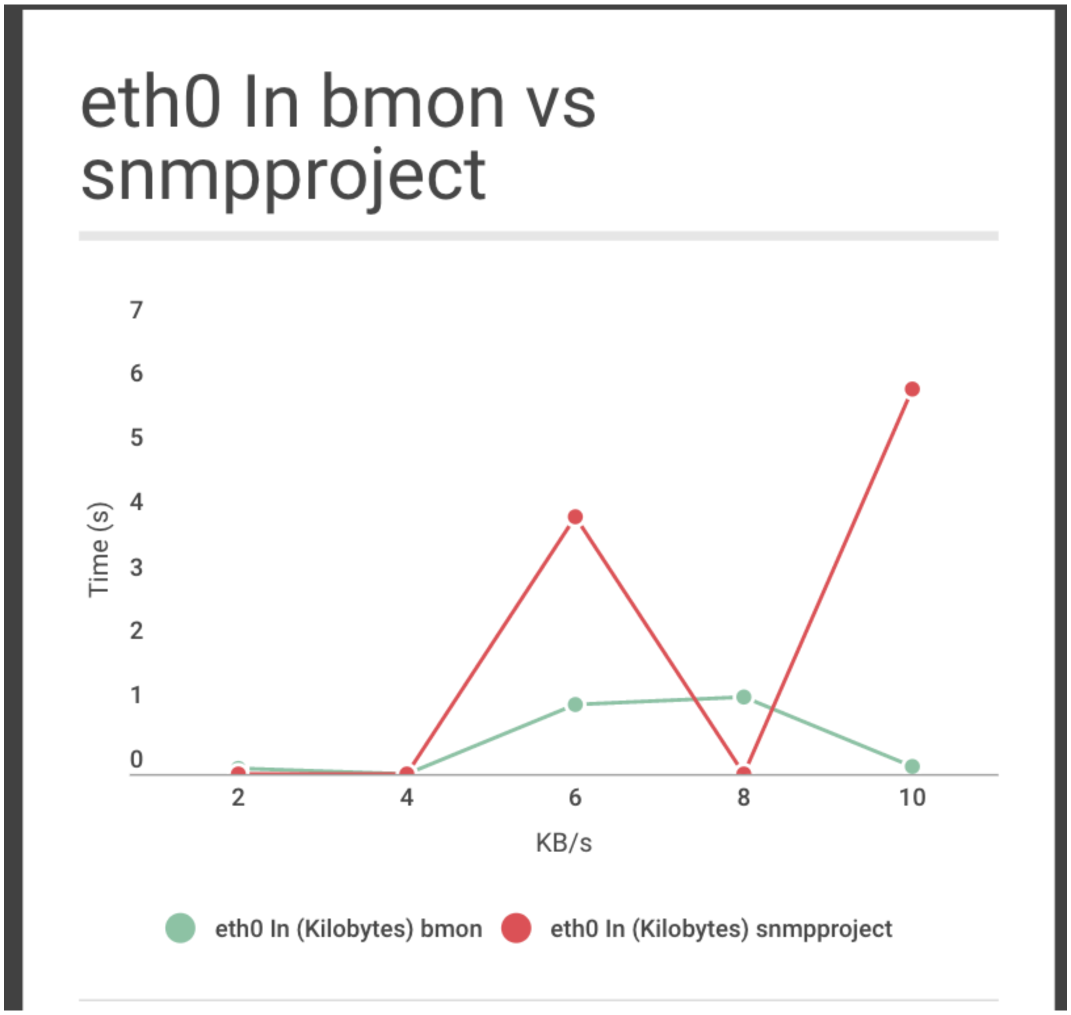
We obtain five samples from our program at two second intervals and compared that with data from `bmon`, which refreshed every second. Necessary adjustments were made in order to fairly compare data.

## Results

### Table - eth0 In `bmon` vs. `snmpproject`

| Time (s) | eth0 In (Kilobytes) bmon | eth0 In (Kilobytes) snmpproject |
|---|---|---|
| 2 | 0.093 | 0 |
| 4 | 0.001 | 0 |
| 6 | 1.083 | 4 |
| 8 | 1.2 | 0 |
| 10 | 0.126 | 6 |

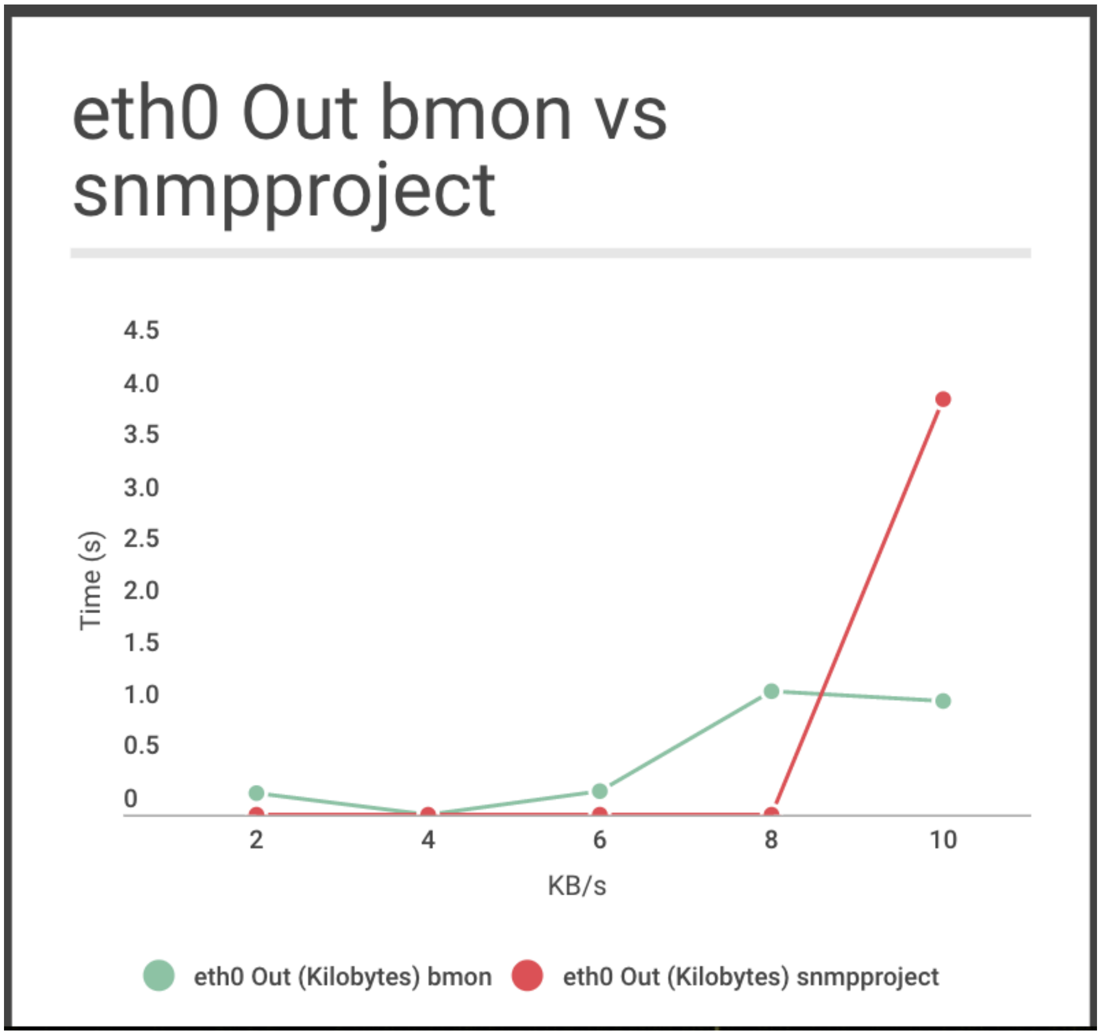
### Graphical Representation



## Table: eth0 Out `bmon` vs. `snmpproject`

| Time (s) | eth0 Out (Kilobytes) bmon | eth0 Out (Kilobytes) snmpproject |
|---|---|---|
| 2 | 0.204 | 0 |
| 4 | 0.003 | 0 |
| 6 | 0.225 | 0 |
| 8 | 1.18 | 0 |
| 10 | 1.089 | 4 |

## Graphical Representation



## Conclusion

1. Our program often outputted 0 KB/s. This was because we were returning whole number kilobytes instead of bytes and as a result a fractional value less than 1 kilobyte would be rounded off to 0.
2. Spikes and drops of traffic in `bmon` seemed to be reflected in `snmpprogram`. The reason behind why the two data sets aren't identical has to do with the time each program obtains its data.
    a. The influx of data is continuous; however, we are monitoring it by seconds. Even though both data sets were adjusted to show the total data at two second intervals, the exact time each data value was observed was not synchronous.
        i. Example: `bmon` – checks eth0 In traffic at 10:02.23
        `snmpprogram` – checks eth0 In traffic at 10:02.52
3. Traffic speeds tend to fluctuate without notice, constant streams of data are not guaranteed, this could also explain the differences in the two data sets.

We can ultimately conclude that the data our program produces is accurate, at the KB/s level.