Desempenho e Dimensionamento de Redes

Network Situational Awareness

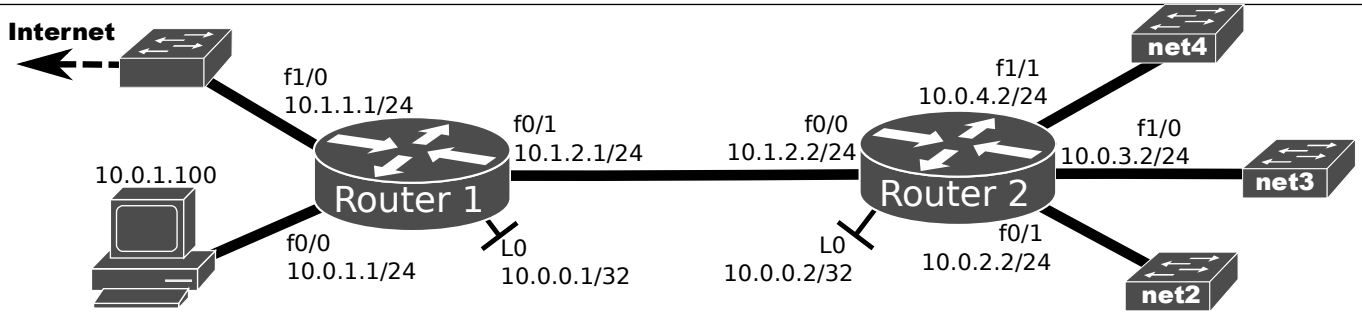
(Data Acquisition Methodologies)

Objectives

- Nodes, links and terminals data acquisition with SNMP
- Traffic flows data acquisition with Netflow/IPFIX
- Raw traffic acquisition with libpcap

Data acquisition with SNMP

1. Configure a network (in GNS3) according to the following figure. The PC can be a VM or the host PC, with Linux (Debian) with Python, SNMP tools, network MIBs and CISCO MIBs.



Objective: Using the Python package *Snimpy* and SNMP, develop a data acquisition script able to identify the active interfaces, periodically report the output and input traffic (bytes and packets) since last report and output buffer size (assuming FIFO queues), in each one.

MIB references: SNMP Object Navigator, http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en
MIBS: IF-MIB, IP-MIB, and CISCO-QUEUE-MIB

Python references: *Snimpy* – API reference, https://snimpy.readthedocs.org/en/latest/api.html
argparse - Parser for command-line options, https://docs.python.org/3/library/argparse.html
matplotlib.pyplot - http://matplotlib.org/api/pyplot_api.html

2. In both routers, configure a SNMP version 3 community (using the name "private") with Read-Only permissions, and access with authentication (MD5, password authpass) and encryption (AES128, password: privpass), for user uDDR from group gDDR:

```
Router(config)# snmp-server user uDDR gDDR v3 auth md5 authpass priv aes 128 privpass
Router(config)# snmp-server group gDDR v3 priv
Router(config)# snmp-server community private RO
```

3. Download and test the baseSNMP.py script, and understand how different MIB objects can be accessed.

```
python baseSNMP.py -r 10.0.0.2
```

baseSNMP.py:

```
mib.path(mib.path()+":/usr/share/mibs/cisco")
load("SNMPv2-MIB")
load("IF-MIB")
load("IP-MIB")
load("RFC1213-MIB")
load("CISCO-QUEUE-MIB")
...
#Creates SNMP manager for router with address args.router
m=M(args.router,'private',3,secname='uDDR',authprotocol="MD5",authpassword="authpass",privprotocol="AES",\
privpassword="privpass")
print(m.sysDescr)           #Gets sysDescr from SNMPv2-MIB
...
print(m.ifDescr.items())          #Lists (order, name) interfaces in ifDescr from IF-MIB
...
for i, name in m.ifDescr.items():
    print("Interface order %d: %s" % (i, name))
...
print(m.ipAddressIfIndex.items())     #Lists ((adr.type,adr),order) interfaces in ipAddressIfIndex from  IP-MIB
...
ifWithAddr={};    #Stores (order, first adr.) of all interfaces with address
for adr, i in m.ipAddressIfIndex.items():
    if not i in ifWithAddr:
        ifWithAddr.update({i:IPfromOctetString(adr[0],adr[1])})
    print('%s, Interface order: %d, %s'%(IPfromOctetString(adr[0],adr[1]),i,m.ifDescr[i]))
print(ifWithAddr)
```

4. Use the following MIB objects to access relevant interface traffic statistics:
- *ifHCOutUcastPkts*, *ifHCInUcastPkts*, *ifHCOutOctets*, *ifHCInOctets* from IF-MIB,
- *cQStatsDepth* from CISCO-QUEUE-MIB (assume FIFO queues, type 2).

5. Create a time loop, with periodicity given by argument, and retrieve interface statistics. Display byte and packet increments (in both directions) and current queue size.

6. Periodically save data in text or JSON files.

Note: Close all files and present final report upon script termination (e.g., catch CTRL+C):

```
try:
...
except KeyboardInterrupt:
…
```

7 (Extra, requires X). Plot interface usage (one per interface). You may use python package *matplotlib.pyplot*, *see: ion(), plot(), draw(), show().*

Data acquisition with NetFlow/IPFIX

Objective: Using Python develop a simple NetFlow collector and infer the traffic matrix between the Internet access (Router1) and the different client networks (net2, net3 and net4). The traffic matrix must be in terms of number of flows, packets and bytes.

NetFlow references: NetFlow Overview

NetFlow Export Datagram Format (version 1 and 5 formats)

Python references: *socket* - Low-level networking interface, https://docs.python.org/2/library/socket.html

struct - Interpret strings as packed binary data, https://docs.python.org/2/library/struct.html

netaddr IPAddress and IPNetwork - https://netaddr.readthedocs.org/en/latest/tutorial_01.html

netaddr IPSet - https://netaddr.readthedocs.org/en/latest/tutorial_03.html

8. Configure Router2 to export (to PC VM) the flow statistics using NetFlow version 1 for all traffic egressing interface f0/1.

```
Router2(config)# interface FastEthernet0/1
Router2(config-if)# ip flow egress
Router2(config)# ip flow-export destination 10.0.1.100 9996
Router2(config)# ip flow-export source Loopback 0
Router2(config)# ip flow-export version 1
```

9. Download and test the baseNetFlow.py script, generating traffic to and from terminals (VPCS) in networks net2, net3, and net4. Understand how the NetFlow packet is received and how data fields can be accessed.

```
python baseNetFlow.py -r 10.0.0.2 -n 10.0.2.0/24 10.0.3.0/24 10.0.4.0/24
```

baseNetFlow.py:

```
def getNetFlowData(data):
    sdata=struct.unpack("!H", data[:2])
    version = sdata[0]
    if version==1:
        print("NetFlow version %d:"%version)
        hformat="!HHIII"        # ! - network (= big-endian), H - C unsigned short (2 bytes), I - C unsigned int (4 bytes)
        hlen = struct.calcsize(hformat)
        ...
        sheader= struct.unpack(hformat, data[:hlen])
        version = sheader[0]
        num_flows = sheader[1]
        sys_uptime = sheader[2]
        time_secs = sheader[3]
        time_nsecs = sheader[4]
```

```
        fformat="!IIIHHIIIIHHHBBBBBBI"         # B - C unsigned char (1 byte)
        flen=struct.calcsize(fformat)
        ...
        flows={}
        for n in range(num_flows):
            flow={}
            offset = hlen + flen*n
            fdata = data[offset:offset + flen]
            sflow=struct.unpack(fformat, fdata)
            flow.update({'src_addr':int_to_ipv4(sflow[0])})
    else:
        print("NetFlow version %d not supported!"%version)

    out=version,flows
    return out

def main():
    parser=argparse.ArgumentParser()
    parser.add_argument('-p', '--port', nargs='?',type=int,help='listening UDP port',default=9996)
    parser.add_argument('-n', '--net', nargs='+',required=True, help='networks')
    parser.add_argument('-r', '--router', nargs='+',required=True, help='IP address(es) of NetFlow router(s)')
    args=parser.parse_args()
    …
    udp_port=args.port
    sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # UDP
    sock.bind(('0.0.0.0', udp_port))
    print("listening on '0.0.0.0':%d"%udp_port)
    try:
        while 1:
            data, addr = sock.recvfrom(8192) # buffer size is 1024 bytes
            version,flows=getNetFlowData(data)     #version 0 reports an error!
            print('Version: %d'%version)
            print(flows)
    except KeyboardInterrupt:
        sock.close()
        print("\nDone!")
```

10. Complete the code to retrieve the relevant NetFlow version 1 data and construct the traffic matrix. Configure the required additional NetFlow export commands in Router1 and Router2.

Note: consider using Python library *netaddr* classes IPAddress, IPNetwork, and IPSet.

11. Include support to NetFlow version 5.

Change routers' configurations to export flow data using NetFlow version 5:

```
Router(config)# ip flow-export version 5
```

Data acquisition with pcap

Objective: Using Python, develop a pcap/tshark based probe able to capture packets, perform DPI and output flow statistics per interval (packets and bytes, download and upload) or packet-inter-arrival times.

The client side network, and server side networks are passed as arguments. Traffic from server side to client side is considered download, and traffic from client side to server side is considered upload.

Python references: pyshark - Python packet parser using wireshark's tshark, http://kiminewt.github.io/pyshark/

- https://github.com/KimiNewt/pyshark

12. Download and test the basePCap.py script, by capturing the TCP/IP packets flows between your PC and YouTube servers during the visualization of videos. Discover your machine IP address (*pc_ipaddr*) and the range of IPv4 addresses for your location (*yt_net*). From UA network, *yt_net* should be (extending the network, and by approximation) 194.210.238.0/24.

```
python basePCap.py -i eth0 -c <pc_ipaddr> -s <yt_net>
```

basePCap.py:

```
def pkt_callback(pkt):
    global scnets
    global ssnets
    global npkts

    if IPAddress(pkt.ip.src) in scnets|ssnets and IPAddress(pkt.ip.dst) in scnets|ssnets:
        npkts=npkts+1
        if pkt.ip.proto=='17':
            print('%s: IP packet from %s to %s (UDP:%s) %s'%\
                (pkt.sniff_timestamp,pkt.ip.src,pkt.ip.dst,pkt.udp.dstport,pkt.ip.len))
        elif pkt.ip.proto=='6':
            print('%s: IP packet from %s to %s (TCP:%s) %s'%\
                (pkt.sniff_timestamp,pkt.ip.src,pkt.ip.dst,pkt.tcp.dstport,pkt.ip.len))
        else:
            print('%s: IP packet from %s to %s (other) %s'%(pkt.sniff_timestamp, pkt.ip.src,pkt.ip.dst,pkt.ip.len))


def main():
    parser=argparse.ArgumentParser()
    ...
    if args.udpport is not None:
        cfilter='udp portrange '+args.udpport
    elif args.tcpport is not None:
        cfilter='tcp portrange '+args.tcpport
    else:
        cfilter='ip'

    cint=args.interface
    print('Filter: %s on %s'%(cfilter,cint))
    try:
        capture = pyshark.LiveCapture(interface=cint,bpf_filter=cfilter)
        capture.apply_on_packets(pkt_callback)
    except KeyboardInterrupt:
        global npkts
        print('\n%d packets captured! Done!\n'%npkts)
```

13. Complete the code to store on a text file the number of packets and bytes (download and upload) per sampling interval (passed as argument, default 0.1 seconds).

14. Complete the code to optionally store the inter-arrival packet times in upload and download directions.