



UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E
INFORMÁTICA

1- DESEMPENHO E DIMENSIONAMENTO DE REDES

Network Situational Awareness

(Data Acquisition Methodologies)

8240 - MESTRADO INTEGRADO EM ENGENHARIA DE
COMPUTADORES E TELEMÁTICA

António Rafael da
Costa Ferreira
NMec: 67405

Rodrigo Lopes
da Cunha
NMec: 67800

Docentes: Paulo Salvador,
Susana Sargento

Março de 2016
2015-2016

Conteúdos

1	Aquisição de dados com SNMP	2
1.1	Configuração	2
1.2	Implementação com Python	3
1.3	Apresentação de gráficos	6
2	Aquisição de dados com NetFlow/IPFix	9
3	PCap	10
3.1	Mostrar os dados obtidos num gráfico	11
3.2	Gravar num ficheiro	11

1 Aquisição de dados com SNMP

1.1 Configuração

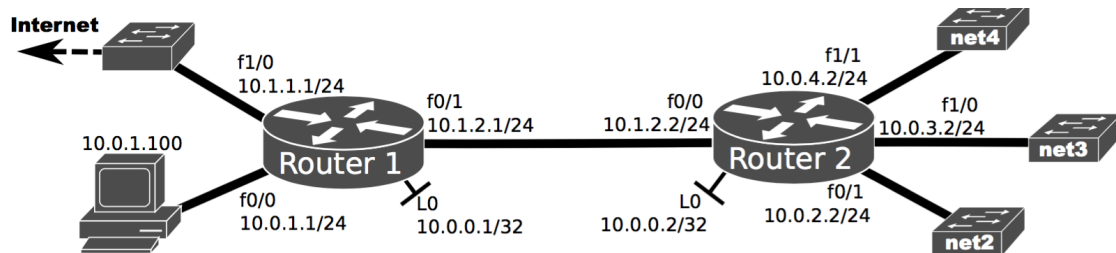


Figura 1: Mapa de IP's

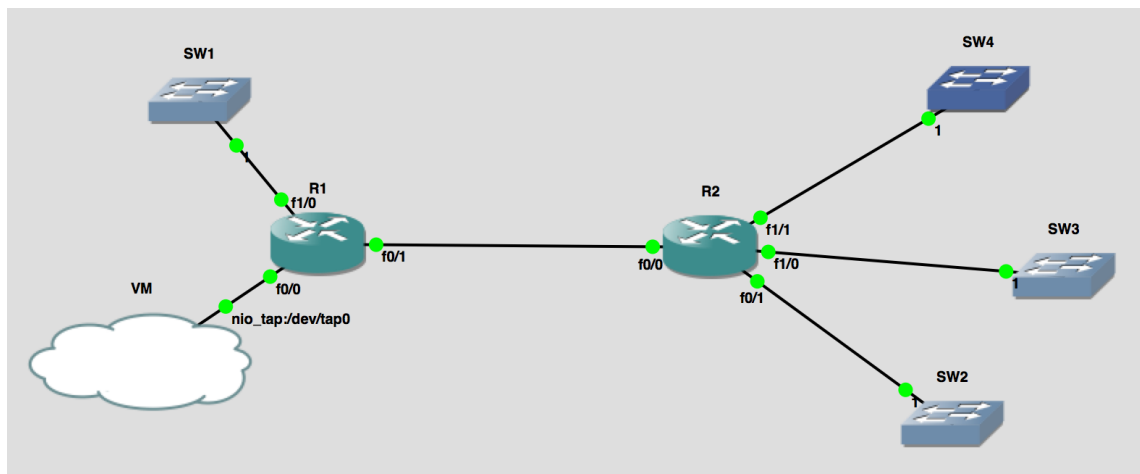


Figura 2: Mapa de IP's

Para ser possível usar o sistema operativo OS X para esta aula, foi usada uma configuração usando TunTap. Usando a imagem disponibilizada no elearning (Ubuntu Server) iniciou-se uma máquina virtual em que tem duas interfaces uma NAT com o computador para ser feito SSH (com port forwarding) e outra interface bridge na interface tap que estiver configurada no GNS. Após isso garantiu-se que para as redes 10.x.x.x era usado o Router 1 na máquina virtual.

Para fazer o desenvolvimento do código Python, usou-se o PyCharm com um Remote Interpreter via SSH.

1.2 Implementação com Python

ifHCOUcastPkts

Através do objeto MIB ifHCOUcastPkts é possível obter o número total de pacotes, em cada interface, que protocolos de nível superior solicitaram ser transmitidos e que não foram endereçados para um endereço multicast ou broadcast. São incluídos, ainda, aqueles que foram descartados ou não foram enviados.

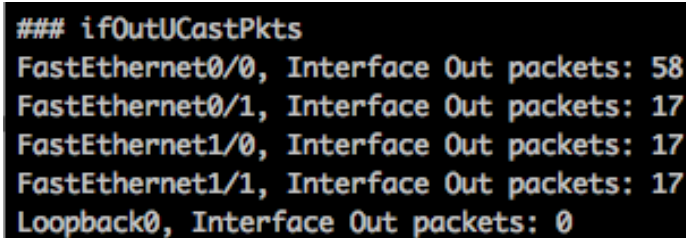
Para obter esta informação utilizou-se o seguinte código Python:

```
ifOutUCastPkts[t] = {}

for i, pkts in m.ifHCOUcastPkts.items():
    if i in ifWithAddr.keys():
        if i not in ifOutUCastPkts[t]:
            ifOutUCastPkts[t].update({i: pkts})
            print('%s, Interface Out packets: %d' % (m.ifDescr[i], pkts))
```

A variável t representa o instante temporal em segundos. Na variável *ifWithAddr* foi guardada a informação de cada interface assim como o seu IP.

Obteve-se o seguinte resultado:



```
### ifOutUCastPkts
FastEthernet0/0, Interface Out packets: 58
FastEthernet0/1, Interface Out packets: 17
FastEthernet1/0, Interface Out packets: 17
FastEthernet1/1, Interface Out packets: 17
Loopback0, Interface Out packets: 0
```

Figura 3: Pacotes por interface

ifHCInUcastPkts

Com o objeto MIB ifHCInUcastPkts é possível obter-se o número de pacotes entregues por uma interface para um camada superior, que não foram endereçados a um endereço multicast ou broadcast.

O código Python, para a obtenção desta informação foi o seguinte:

```
ifInUCastPkts[t] = {}

for i, pkts in m.ifHCInUcastPkts.items():
    if i in ifWithAddr.keys():
        if i not in ifInUCastPkts[t]:
            ifInUCastPkts[t].update({i: pkts})
            print('%s, Interface In packets: %d' % (m.ifDescr[i], pkts))
```

Foi possível então obter os seguintes resultados por interface:

```

### ifInUcastPkts
FastEthernet0/0, Interface In packets: 51
FastEthernet0/1, Interface In packets: 0
FastEthernet1/0, Interface In packets: 0
FastEthernet1/1, Interface In packets: 0
Loopback0, Interface In packets: 0

```

Figura 4: Pacotes por interface

ifHCOutOctets

O objeto MIB ifHCOutOctets devolve o número total de octetos transmitidos pela interface, incluindo os *framing characters*.

Utilizou-se o seguinte código Python:

```

ifOutOctets[t] = {}

for i, pkts in m.ifHCOutOctets.items():
    if i in ifWithAddr.keys():
        if i not in ifOutOctets[t]:
            ifOutOctets[t].update({i: pkts})
            print('%s, Interface Out octets: %d' % (m.ifDescr[i], pkts))

```

Por cada interface, obtiveram-se os seguintes resultados:

```

### ifOutOctets
FastEthernet0/0, Interface Out octets: 13404
FastEthernet0/1, Interface Out octets: 4991
FastEthernet1/0, Interface Out octets: 4991
FastEthernet1/1, Interface Out octets: 4991
Loopback0, Interface Out octets: 0

```

Figura 5: Octetos por interface

ifHCInOctets

O número total de octetos recebidos por cada interface, incluindo os *framing characters*, é dado pelo objecto MIB ifHCInOctets.

Utilizou-se então, o seguinte código Python:

```

ifInOctets[t] = {}

for i, pkts in m.ifHCInOctets.items():
    if i in ifWithAddr.keys():
        if i not in ifInOctets[t]:
            ifInOctets[t].update({i: pkts})
            print('%s, Interface In octets: %d' % (m.ifDescr[i], pkts))

```

Por cada interface, obtiveram-se os resultados seguintes:

```
### ifInOctets
FastEthernet0/0, Interface In octets: 9831
FastEthernet0/1, Interface In octets: 0
FastEthernet1/0, Interface In octets: 0
FastEthernet1/1, Interface In octets: 0
Loopback0, Interface In octets: 0
```

Figura 6: Octetos por interface

ifQstats

O objeto MIB ifQstats devolve o número de mensagens que se encontram na fila de mensagens.

Utilizou-se o seguinte código Python para obter o número de mensagens por interface:

```
ifQstats[t] = {}

for (i, u), pkts in m.cQStatsDepth.items():
    if i in ifWithAddr.keys():
        if i not in ifQstats[t]:
            ifQstats[t].update({i: pkts})
            print(' %s, _Interface_Queue_Size:_%d' % (m.ifDescr[i], pkts))
```

Por cada interface, obtiveram-se os seguintes números de mensagens por interface:

```
### ifQstats
FastEthernet0/0, Interface Queue Size: 0
FastEthernet0/1, Interface Queue Size: 0
FastEthernet1/0, Interface Queue Size: 0
FastEthernet1/1, Interface Queue Size: 0
Loopback0, Interface Queue Size: 0
```

Figura 7: Mensagens na fila por interface

Verifica-se então que não existem mensagens em nenhuma Fila de cada interface, o que mostra que o processamento das mensagens tem sido feito de forma eficiente, não ficando mensagens em espera.

1.3 Apresentação de gráficos

Utilizando o package de Python *matplotlib.pyplot*, foi possível a construção de gráficos com a informação que se foi disponibilizando, enquanto o programa estava a correr, podendo ser feita então a análise com recurso a imagens gráficas da maneira como os pacotes, octetos foram recebidos ou transmitidos.

Num exemplo, onde a captura foi parada ao fim de 20 segundos e onde foram efetuados algumas vezes o comando *ping*, obtiveram-se os seguintes gráficos para cada interface:

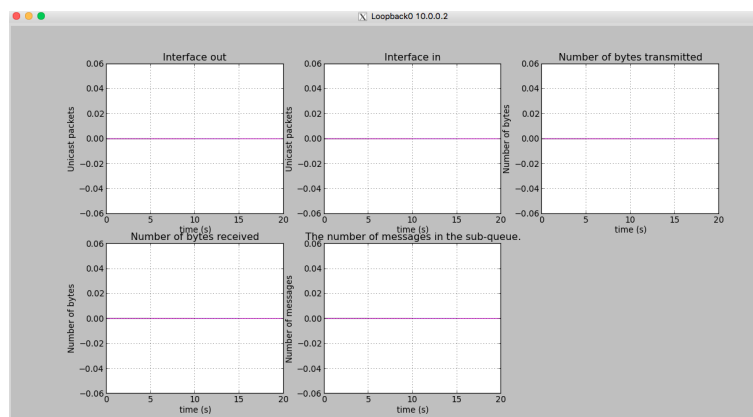


Figura 8: Inteface Loopback0

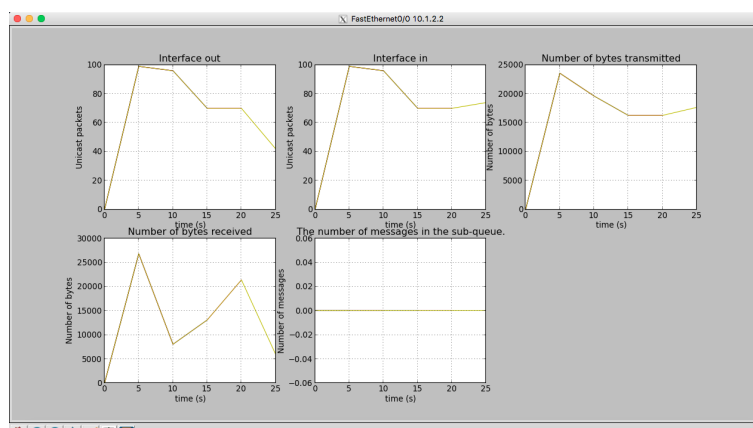


Figura 9: Inteface FastEthernet 0/0

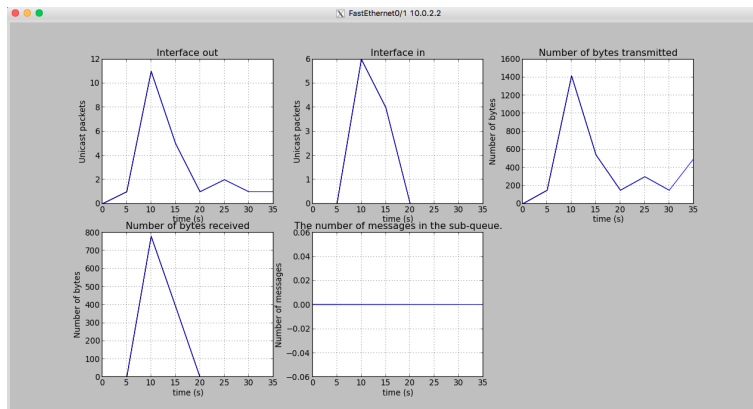


Figura 10: Interface FastEthernet 0/1

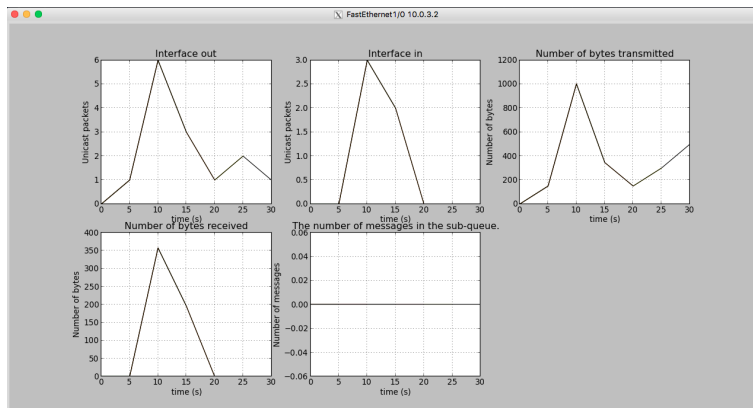


Figura 11: Interface FastEthernet 1/0

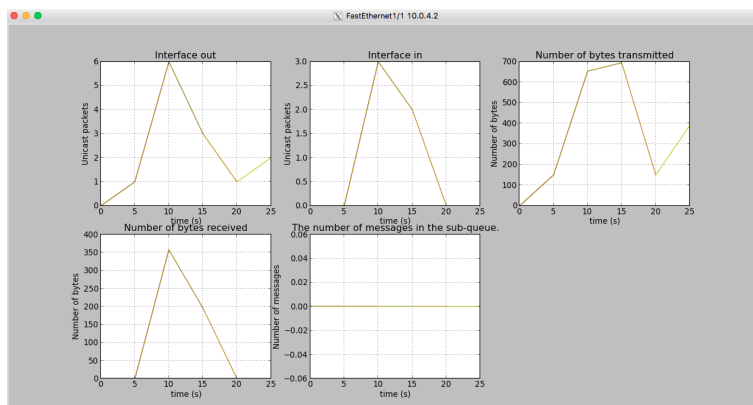


Figura 12: Interface FastEthernet 1/1

Todos os dados são guardados num ficheiro JSON, para que possam ser utilizados mais tarde, para análise, ou comparação. Foram ainda guardados num ficheiro de texto, os prints da execução do programa, onde também se encontram os dados que se guardam no ficheiro JSON.

2 Aquisição de dados com NetFlow/IPFix

Nesta segunda parte do guia prático, pretendia-se obter uma matriz tráfego entre o router de acesso à Internet e várias redes de clientes.

Numa primeira fase utilizou-se a versão 1 do Netflow, para capturar o tráfego que passava nas interfaces. Esta versão possui um formato do tipo, *!IIHHIIIIHHHBBBBBI*, onde o "I", corresponde a um inteiro de 4 bytes, o "H", um short de 2 bytes e o "B", corresponde a 1 byte.

Nas interfaces f0/1, f1/0 e f1/1 do Router 2 e f0/1 do Router 1, captura-se o tráfego na saída do Router, e todo o tráfego é redirecionado para o PC que está a receber as estatísticas e a apresentar as mesmas, que se encontra na ligado à interface f0/0 do Router 1.

Posteriormente, foi alterado nos routers o Netflow para ser utilizada a versão 5. Nesta versão utiliza-se o formato *!IIHHIIIIHHBBBBHHBBH*.

Com esta versão e com o comando *python baseNetFlow.py -r 10.0.0.2 10.0.0.1 -n 10.0.2.0/24 10.0.3.0/24 10.0.4.0/24*, obtiveram-se os seguintes resultados, numa matriz:

```
NetFlow version 5:
2
Version: 5, from 10.0.0.1 (2)
Updated at: (10.0.2.0/24, other)
```

	10.0.2.0/24	10.0.3.0/24	10.0.4.0/24	other
10.0.2.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(1, 5, 420)
10.0.3.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
10.0.4.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
other	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

(flows, packets, bytes)

Figura 13: Matriz Netflow

```
NetFlow version 5:
2
Version: 5, from 10.0.0.2 (2)
Updated at: (other, 10.0.2.0/24)
```

	10.0.2.0/24	10.0.3.0/24	10.0.4.0/24	other
10.0.2.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(2, 25, 1220)
10.0.3.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
10.0.4.0/24	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
other	(1, 5, 420)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

(flows, packets, bytes)

Figura 14: Matriz Netflow

Updated at: (10.0.3.0/24, 10.0.2.0/24)

	10.0.2.0/24	10.0.3.0/24	10.0.4.0/24	other
10.0.2.0/24	(0, 0, 0)	(2, 10, 840)	(1, 5, 420)	(2, 25, 1220)
10.0.3.0/24	(2, 10, 840)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
10.0.4.0/24	(1, 5, 420)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
other	(2, 30, 2040)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

(flows, packets, bytes)

Figura 15: Matriz Netflow

3 PCap

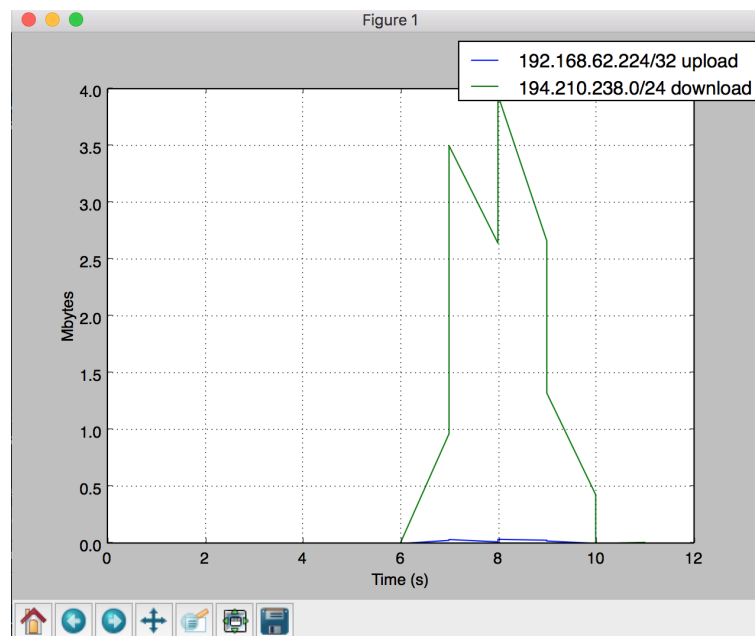


Figura 16: Gráfico ao receber um vídeo do YouTube

O algoritmo desenvolvido para processar os pacotes recebidos pelos serviços ou enviado pelo cliente foi elaborado com a ideia de ser simples e rápido de processar.

Com isto pensou-se que dicionário com o intervalo de tempo com o número de Mbytes recebidos ou enviados é igual ao timestamp atual menos o timestamp base (quando iniciou o script), sendo isto tudo, a dividir pelo intervalo que foi definido em segundos.

```
interval_num = int(math.trunc((timestamp - timestamp_init) / timestamp_interval))
```

Para atualizar o dicionário, assumiu-se que para o idx atual do dicionário, irá-se ver se corresponde ao intervalo atual. Após isso percorre-se esse inter-

valo obtido e cria-se intervalos a 0 que correspondem aos intervalos que não houve receção ou envio de pacotes. Para isso, fez-se:

```
idx = bytes_upload[network]["idx"] + 1

for i in range(idx, interval_num):
    bytes_upload[network]["idx"] += 1
    bytes_upload[network]["bytes"].append(0)
```

Depois vê-se se o intervalo atual corresponde ao intervalo do dicionário e se sim, adiciona-se o tamanho do pacote, em mega-bytes. Se não corresponder, incrementa-se o índice e adiciona-se ao dicionário o tamanho do pacote.

```
if interval_num == (len(bytes_upload[network]["bytes"]) - 1):
    idx = bytes_upload[network]["idx"]
    bytes_upload[network]["bytes"][idx] += pkt_len
else:
    bytes_upload[network]["idx"] += 1
    bytes_upload[network]["bytes"].append(pkt_len)
```

3.1 Mostrar os dados obtidos num gráfico

Os dados obtidos foram depois colocados num gráfico para se ter melhor perceção do que foi obtido.

3.2 Gravar num ficheiro

Os dados obtidos são guardados em dois ficheiros, um para download e outro para upload.