FCT-NOVA

Arquitectura e Protocolos de Redes de Computadores

Relatório Lab. 2

Passive Network Measurements and Analysis of NetFlow data

Autores
Rodrigo Faria Lopes nº 50435
João Machado nº 58722
Professores:
José Legatheaux Martins
Paulo Afonso Lopes
Turno: P1

28 de Maio de 2020

Conteúdo

1 Introdução

Neste relatório iremos mostrar e explicar os resultados obtidos da análise de dois ficheiros distintos (www.fct.unl.pt.csv e bigFlows.csv) respondendo às perguntas especificadas do enunciado fornecido. Todos os tópicos foram implementados desde as obrigatórias (a1, a2, a3, a4 e a5) às opcionais (b1, e b2).

A implementação deste enunciado é escrito em código Python na sua versão 3 [1], tendo sido também usada uma API externa e um ficheiro auxiliar, para a Geolocalização [2] (IPGeolocation.py) e listagem das portas TCP e UDP [3] respectivamente. O uso da API requis uma autenticação, pelo que usamos as nossas contas Gmail da faculdade para obter as chaves necessárias (estando estas *harcoded* nos ficheiros necessários)

Inicialmente começamos por implementar um *script* (intoDB.py) para inserir todos os dados do ficheiro .csv para uma base de dados relacional (SQLite [4]) de forma a mais tarde poder-mos fazer as perguntas utilizando SQL. Esta abordagem posteriormente não foi utilizada, porque achamos que talvez fosse mais confuso. Então decidimos manter o ficheiro intoDB.py para eventuais trabalhos de otimização futuros caso estes fossem necessários.

Nos tópicos opcionais (b1 e b2) optamos por criar os gráficos pretendidos utilizando a biblioteca *matplotlib* [5] de modo a converter a nossa informação em forma de mapa (key value) em gráficos 2D.

De seguida iremos, em cada capitulo, mostramos os nossos resultados para cada tópico do enunciado, falando da nossa abordagem ao problema e forma como esta foi implementada.

Inicialmente começamos por implementar uma estratégia de leitura dos ficheiros, onde abrimos o ficheiro pretendido iterando linha a linha a informação nele contida. O formato deste código foi repetido para os diferentes *scripts* de modo a facilitar e dividir cada tópico pretendido.

Cada coluna, em termos de código, será representada da seguinte forma:

- row[0] = time start
- row[1] = time end
- row[2] = duration (s)
- row[3] = source address
- row[4] = destination address
- row[5] = source port
- row[6] = destination port
- row[7] = protocol
- row[8] = flags
- row[9] = packets
- row[10] = bytes

Na figura 1.1 podemos ver esta parte do *script* comum a todos os ficheiros implementados, não sendo esta mais falava nos próximos capítulos por motivos de repetição:

```
with open(input_file_path) as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=",")
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print("[*] File: " + input_file_path)
        else:
            print("CALCULATE")
        line_count += 1
print(f"[*] Processed {line_count} lines")
print("\n=======================RESULT=====================================")
print("RESULT")
print("==================================================================")
print("\n\n")
```

Figura 1.1: *Código inicial para cada tópico*

Nota: O código está num repositório GitHub privado, pelo que caso queira ter acesso ao mesmo deverá enviar um email aos autores a pedir autorização de acesso.

Link: **https://github.com/rfa-lopes/APRC**

2

A1

’Compute the total number of packets and bytes (in and out) per protocol (TCP, UDP, ICMP, . . .) contained in the flows.’

2.1 Explicação do código desenvolvido

A computação deste tópico foi relativamente fácil, onde apenas foi necessário iterar todos os tuplos do ficheiro, verificar o seu protocolo e adicionar a uma mapa o eu número de *bytes* e *packets*. Como já foi referido, neste tópico utilizamos uma estrutura de dados *(Key, Value)* para guardar em memória os resultados de cada iteração. O código seguinte mostra a parte do *script* que trata de cada tuplo do ficheiro.

```
protocol = row[7]
packets = row[9]
bytes_ = row[10]

if protocol not in protocols_bytes or protocol not in protocols_packets:
    # print("[+] New protocol found: " + str(protocol))
    protocols_bytes[str(protocol)] = int(0)
    protocols_packets[str(protocol)] = int(0)

protocols_bytes[str(protocol)] += int(bytes_)
protocols_packets[str(protocol)] += int(packets)
```

Figura 2.1: *Código do tópico a1*

2.2 Resulados obtidos pelo ficheiro www.fct.unl.pt.csv

Como se pode ver na figura 2.2, foram processadas as 21362 linhas do ficheiro www.fct.unl.pt.csv, existindo fluxos TCP, UDP e ICMP, estando associados a cada um destes o número de *bytes* e *packets* como pretendido.

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines

=======================RESULT=====================================
By Bytes:
{'TCP': 950033233, 'UDP': 2995, 'ICMP': 424}

By Packets:
{'TCP': 729144, 'UDP': 29, 'ICMP': 4}
==================================================================
```

Figura 2.2: *Output do script a1.py*

2.3 Resulados obtidos pelo ficheiro bigFlows.csv

Na figura 2.3 verificamos a mesma estrutura de resultados mas agora para o ficheiro bigFlows.csv, onde podemos verificar que este apresenta um tipo de protocolo que não existia no ficheiro anterior, IGMP.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines

=======================RESULT=====================================
By Bytes:
{'TCP': 191081156, 'UDP': 36971693, 'ICMP': 303783, 'IGMP': 14144}

By Packets:
{'TCP': 432868, 'UDP': 152733, 'ICMP': 4266, 'IGMP': 286}
==================================================================
```

Figura 2.3: *Output do script a1.py*

3

A2

’Determine the 50 most popular IP addresses external to the domain by number of flows’ ´

3.1 Explicação do código desenvolvido

A computação deste tópico não apresentou uma dificuldade muito superior em comparação com o tópico anterior, no entanto foi usada uma função auxiliar para facilitar e tornar o código mais legível. A função auxiliar basicamente diz-nos se o endereço IP que estamos a tratar faz parte do domínio ou se é externo. Assim conseguimos com facilidade fazer uma lsita de tuplos onde agrupamos o *IP* com o número de *flows*. Por fim fazemos uma ordenação pela quantidade de flows e descartamos quaisquer resultados após os 50 primeiros. O código seguinte mostra a parte do *script* que trata de cada tuplo do ficheiro.

```
source_address = str(row[3])
destination_address = str(row[4])

if isExternalAddress(input_file_path, source_address):
    if source_address not in ip_addrs:
        # print("[+] New ip addr found: " + source_address)
        ip_addrs[source_address] = int(0)
    ip_addrs[source_address] += int(1)

if isExternalAddress(input_file_path, destination_address):
    if destination_address not in ip_addrs:
        # print("[+] New ip addr found: " + destination_address)
        ip_addrs[destination_address] = int(0)
    ip_addrs[destination_address] += int(1)
```

Figura 3.1: *Código do tópico a2*

3.2 Resulados obtidos pelo ficheiro www.fct.unl.pt.csv

Como se pode ver na figura 3.3, foram processadas as 21362 linhas do ficheiro www.fct.unl.pt.csv, e foram recolhidos e ordenados os endereços IPs com maior número de flows.

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines.

======================RESULT====================================

[('154.0.164.204', 808), ('54.175.144.198', 346), ('219.133.250.21
4', 184), ('207.46.13.120', 174), ('207.46.13.37', 156), ('157.55.
39.226', 144), ('207.46.13.100', 140), ('81.84.88.246', 72), ('82.
154.141.69', 70), ('79.169.61.164', 60), ('2.80.15.173', 58), ('81
.84.214.187', 58), ('85.246.219.7', 54), ('2.82.210.98', 52), ('85
.241.84.177', 52), ('95.93.0.151', 50), ('85.138.157.141', 50), ('
85.247.41.221', 48), ('85.247.12.218', 46), ('177.15.87.95', 42),
('94.62.44.122', 38), ('81.84.156.135', 38), ('46.189.209.226', 38
), ('85.138.238.35', 38), ('46.229.168.151', 36), ('217.129.138.75
', 36), ('217.129.180.152', 34), ('85.241.227.149', 34), ('46.229.
168.140', 32), ('46.229.168.145', 32), ('85.242.213.147', 32), ('1
88.81.24.9', 32), ('85.139.39.123', 32), ('46.229.168.139', 30), (
'148.71.94.158', 30), ('51.77.52.160', 30), ('89.115.115.180', 30)
, ('85.242.168.32', 30), ('46.229.168.147', 28), ('109.49.73.14',
28), ('46.229.168.153', 26), ('144.64.202.244', 26), ('213.180.203
.182', 26), ('93.108.20.203', 26), ('109.48.214.218', 26), ('95.95
.16.33', 26), ('88.157.92.155', 26), ('85.241.237.184', 26), ('94.
63.82.145', 26), ('81.84.80.76', 26)]
================================================================
```

Figura 3.2: *Output do script a2.py*

3.3 Resulados obtidos pelo ficheiro bigFlows.csv

Na figura 3.4 verificamos que foram processadas 42845 linhas do ficheiro bigFlows.csv, e foram recolhidos e ordenados os endereços IPs com maior número de flows. Fique de notar a diferença na quantidade de flows recolhida.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines.

=====================RESULT=================================

[('8.8.8.8', 2824), ('96.43.146.22', 732), ('96.43.146.48', 400),
('239.255.255.250', 375), ('96.43.146.176', 354), ('68.64.21.45',
318), ('8.8.4.4', 254), ('68.64.21.41', 216), ('68.64.21.42', 214)
, ('23.0.49.244', 196), ('68.87.152.185', 186), ('208.111.160.6',
185), ('199.102.234.31', 174), ('216.52.242.80', 162), ('108.170.1
93.53', 158), ('208.111.161.254', 156), ('216.115.219.126', 146),
('216.52.121.204', 142), ('68.64.24.250', 142), ('216.219.118.244'
, 142), ('78.108.118.250', 142), ('96.43.146.178', 140), ('64.56.2
03.22', 108), ('173.194.43.45', 108), ('8.27.243.253', 108), ('68.
64.21.62', 108), ('173.194.43.59', 106), ('68.64.21.40', 106), ('6
8.64.21.37', 106), ('68.64.29.40', 106), ('68.64.21.71', 106), ('9
6.43.146.50', 104), ('173.194.43.60', 104), ('68.64.21.44', 104),
('216.219.113.250', 101), ('216.115.223.200', 98), ('67.217.64.244
', 98), ('66.151.150.190', 98), ('108.59.243.194', 96), ('255.255.
255.255', 95), ('64.90.204.229', 94), ('224.0.0.22', 88), ('173.19
4.43.57', 86), ('173.194.43.58', 84), ('184.73.225.104', 82), ('69
.174.248.253', 82), ('157.56.242.198', 80), ('74.3.237.146', 78),
('67.217.78.31', 75), ('138.108.6.20', 74)]
============================================================
```

Figura 3.3: *Output do script a2.py*

4 A3

’Determine the 50 most popular IP addresses external to the domain by number of bytes.’

4.1 Explicação do código desenvolvido

Se compararmos com os outros códigos, a computação deste talvez seja considerada a mais simples. Isto não pela sua complexidade por si só, mas pela semelhança com o código em A2. No fundo, a única entre os dois devesse só a onde, nos ficheiros csv, ir buscar o resultado. Observe-se no código abaixo, que os *bytes* são conseguidos na coluna 10 do ficheiro.

```
source_address = str(row[3])
destination_address = str(row[4])
bytes_ = int(row[10])

if isExternalAddress(input_file_path, source_address):
    if source_address not in ip_addrs:
        # print("[+] New ip addr found: " + source_address)
        ip_addrs[source_address] = 0
    ip_addrs[source_address] += bytes_

if isExternalAddress(input_file_path, destination_address):
    if destination_address not in ip_addrs:
        # print("[+] New ip addr found: " + destination_address)
        ip_addrs[destination_address] = 0
    ip_addrs[destination_address] += bytes_
```

Figura 4.1: *Código do tópico a3*

4.2 Resultados obtidos pelo ficheiro www.fct.unl.pt.csv

Como se pode ver na figura 4.3, foram processadas as linhas do ficheiro www.fct.unl.pt.csv, tendo ordenado, por número de bytes, os endereços IP.

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines.

=========================RESULT=================================
[('99.197.17.22', 26445088), ('85.138.157.141', 22114832), ('217.1
180.152', 19610574), ('95.93.0.151', 19280904), ('207.46.13.37', 1
1376), ('217.129.40.154', 8302322), ('81.84.156.135', 7851399), ('
108.20.203', 7236810), ('104.249.195.179', 7072086), ('154.0.164.2
, 6960047), ('213.180.203.182', 6550481), ('94.62.49.34', 6332595)
'176.79.61.83', 5495730), ('85.245.217.67', 4928507), ('85.241.84.
', 4894769), ('109.49.48.166', 4772392), ('114.119.166.32', 468362
 ('207.46.13.100', 4646290), ('90.8.20.12', 4587318), ('89.115.115
0', 4555002), ('37.189.74.113', 4426559), ('213.22.168.126', 42057
, ('94.62.219.250', 4141312), ('94.63.82.145', 4103396), ('94.62.4
22', 4050920), ('109.49.73.14', 3999831), ('46.189.209.226', 37834
, ('186.214.138.14', 3684727), ('79.169.61.164', 3680309), ('89.11
8), ('81.84.80.76', 3128170), ('31.148.131.141', 3089061), ('188.8
1.24.9', 2978497), ('84.91.153.83', 2947341), ('85.246.38.252', 29
00145), ('81.84.48.90', 2892296), ('85.242.168.32', 2876199), ('85
.246.219.7', 2839234), ('89.115.174.49', 2768531), ('54.175.144.19
8', 2756360), ('109.48.196.197', 2747698)]
================================================================
```

Figura 4.2: *Output do script a3.py*

4.3 Resultados obtidos pelo ficheiro bigFlows.csv

Como se pode ver na figura 4.4, foram processadas as linhas do ficheiro bigFlows.csv, tendo ordenado, por número de bytes, os endereços IP.

Este exercício foi realizado de forma quase igual ao anterior, sendo a única diferença que em vez de nos vermos o flow, vimos o número de bytes como foi pedido.

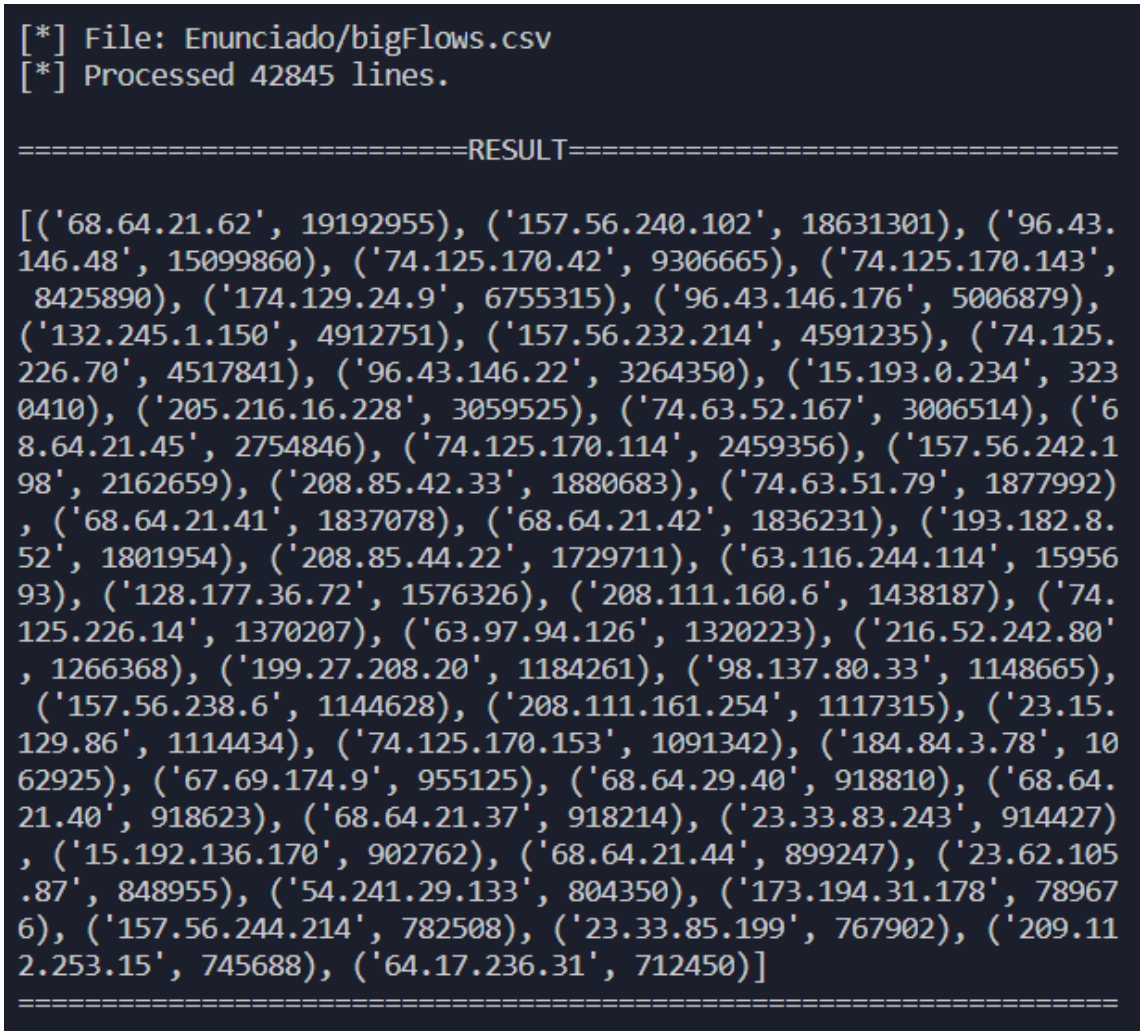

Figura 4.3: *Output do script a3.py*

5 A4

’What are the top 50 most popular applications used by the computers in the domain?’

Nota: There is a Wikipedia page that lists the assigned port numbers, https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers. Or you can Google for the IANA’s Well Known Port Numbers.

5.1 Explicação do código desenvolvido

A computação deste tópico já trouxe mais dificuldades, não propriamente no conseguir os resultados, mas em trata-los de forma a ter um output compreensível e legível. No excerto de código abaixo, mostramos que fomos simplesmente buscar os dados que nos interessavam, ou seja *Source Port, Destination Port e Packets*, às colunas correspondentes do ficheiro bigFlows.csv.

```
source_port = int(row[5])
destination_port = int(row[6])

packets = int(row[9])

init_or_insert_variables(source_port, packets)
init_or_insert_variables(destination_port, packets)
```

Figura 5.1: *Código do tópico a4*

É neste excerto de código que fazemos o tratamento de dados de forma semelhante aos tópicos anteriores.

```
def init_or_insert_variables(port, nr_packets):
    if port not in ports_usage_flows or port not in ports_usage_packets:
        # print("[+] New port found: " + source_port)
        ports_usage_flows[port] = 0
        ports_usage_packets[port] = 0

    ports_usage_flows[port] += 1
    ports_usage_packets[port] += nr_packets
```

Figura 5.2: *Código auxiliar do tópico a4*

E para termos um output compreensível usamos as seguintes funções auxiliares. A primeira dá-nos um mapa onde cada por corresponde um uma aplicação, como exemplo temos o *port 80* que corresponde a *Hypertext Transfer Protocol (http)*. A segunda "pega"nos ports dos nossos resultados e atribui-lhes um nome baseado nesse mapa, como exemplo se tivermos um *port 443* nos nossos resultados, esta função atribui-lhe o nome https como está no mapa.

```
def getPortNamesFromFile():
    dic = {}
    with open("Utils/service-names-port-numbers.csv", mode="r") as infile:
        reader = csv.reader(infile)
        line = 0
        for rows in reader:
            if line != 0:
                dic[rows[1]] = rows[0]
            line += 1
    return dic
```

Figura 5.3: *Código auxiliar do tópico a4*

```
def convertPortNumbers(data, port_dic):
    for i in port_dic.keys():
        value = port_dic[i]
        if value == "":
            value = i
        try:
            port_n = int(i)
            if port_n in data:
                data[value + "(" + str(port_n) + ")"] = data.pop(port_n)
        except ValueError:
            continue
    return data
```

Figura 5.4: *Código auxiliar do tópico a4*

5.2 Resultados obtidos pelo ficheiro bigFlows.csv

No enunciado do trabalho é nos perguntado se somos surpreendidos pelo resultado. A resposta é sim e não. Foi surpreendente a quantidade de ports que obtivemos que não tiveram nenhum mapeamento aos ports da lista que usamos, sendo que nem todos são de aplicações externas (ports 49152 a 65535). No entanto, dos ports que conseguimos identificar, já esperávamos quais é que teriam mais fluxo.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines.
```

By Flows:

	Port	Flows
0	5440	12141
1	http(80)	8760
2	https(443)	7526
3	domain(53)	4235
4	snmp(161)	2740
5	0(0)	1637
6	vids-avtp(1853)	1611
7	49151(49151)	970
8	trivnet1(8200)	949
9	lbc-watchdog(2816)	574
10	netbios-ns(137)	376
11	ssdp(1900)	371
12	53807	291
13	dls-monitor(2048)	277
14	db-lsp-disc(17500)	244
15	8014	235
16	59681	117
17	60838	113
18	rtip(771)	111
19	49514	110
20	56495	108
21	58246	108
22	63861	108
23	54654	108
24	62603	108
25	50477	107
26	51444	106
27	64264	106
28	49319	105
29	syslog(514)	104
30	ttl-publisher(5462)	86
31	60377	84
32	epmap(135)	81
33	62332	78
34	57478	76
35	55702	75
36	58247	74
37	63862	73
38	62560	73
39	49515	73

By Packets:

	Port	Packets
0	https(443)	157742
1	5440	131040
2	vids-avtp(1853)	126095
3	http(80)	121975
4	53807	41366
5	49358	20919
6	trivnet1(8200)	9382
7	60838	6072
8	54654	6059
9	0(0)	6055
10	62603	6035
11	49514	6032
12	58246	6026
13	64264	6026
14	50477	6026
15	63861	6022
16	56495	6019
17	51444	6017
18	49319	5900
19	50193	5299
20	60283	5081
21	49311	4693
22	domain(53)	4372
23	55363	4338
24	ssh(22)	4304
25	17000	4266
26	51035	4000
27	60658	3982
28	fcp-addr-srvr1(5500)	3818
29	snmp(161)	3418
30	58431	2808
31	56566	2721
32	56707	2526
33	http-alt(8080)	2504
34	55445	2495
35	52398	2336
36	tripe(4070)	2255
37	ttl-publisher(5462)	2228
38	60692	2208
39	52396	2123

Figura 5.5: *Output do script a4.py*

6

A5

’Aggregate the two flows representing the same TCP connection; count the total number of TCP connections collected, and the total number of TCP connections that started and finished correctly.’

6.1 Explicação do código desenvolvido

Este código foi sem dúvida alguma o mais complicado da parte "mandataria". Foi complicado não só codificar mas também perceber o pedido para poder começar a construí-lo. Usámos a mesma estratégia para obter os dados usada previamente. Feito isso precisámos de garantir que o que quer que fizéssemos fosse apenas sobre uma única conexão TCP, daí o *if* que verifica se duas linhas correspondem a mesma conexão. Aí precisámos de saber se a conexão tinha começado e acabado com sucesso. Para fazer isso tivemos que usar as *flags* e concluímos que uma conexão iniciada com sucesso necessita das *flags ACK* e *SYN,* e uma conexão terminada com sucesso necessita da *flag F*.

```
protocol = str(row[7])
if protocol == "TCP":
    if tmp == 0:
        line1 = row
        tmp = 1
    else:
        line2 = row

        src_addr1 = str(line1[3])
        dest_addr1 = str(line1[4])
        src_port1 = int(line1[5])
        dest_port1 = int(line1[6])
        flags1 = str(line1[8])

        src_addr2 = str(line2[3])
        dest_addr2 = str(line2[4])
        src_port2 = int(line2[5])
        dest_port2 = int(line2[6])
        flags2 = str(line2[8])

        if (
            src_addr1 == dest_addr2
            and dest_addr1 == src_addr2
            and src_port1 == dest_port2
            and dest_port1 == src_port2
        ):
            count_tcp += 1
            if (
                "S" in flags1
                and "F" in flags1
                and "A" in flags1
                and "S" in flags2
                and "F" in flags2
                and "A" in flags2
            ):
                count_conections += 1
            tmp = 0
        else:
            line1 = row
            tmp = 1
```

Figura 6.1: *Código do tópico a5*

6.2 Resulados obtidos pelo ficheiro www.fct.unl.pt.csv

Como se pode ver na figura 6.2, foram processadas as linhas do ficheiro www.fct.unl.pt.csv. Apresentamos no output o total de conexões TCP e quantos dessas começaram e terminaram com sucesso.

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines

========================RESULT=================================
Conexões TCP (all): 10645
Conexões TCP (started and finished): 8132
===============================================================
```

Figura 6.2: *Output do script a5.py*

6.3 Resulados obtidos pelo ficheiro bigFlows.csv

Como se pode ver na figura 6.3, foram processadas as linhas do ficheiro bigFlows.csv. Apresentamos no output o total de conexões TCP e quantos dessas começaram e terminaram com sucesso.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines

========================RESULT=================================
Conexões TCP (all): 8382
Conexões TCP (started and finished): 5518
===============================================================
```

Figura 6.3: *Output do script a5.py*

6.4 'Explain the several possible technical reasons that justify why there are TCP connections that were not started or ended correctly.'

Problemas no inicio das conexões, SYN+ACK. A conexão está a ser largada em algum sitio durante o envio, geralmente por sistemas intermédios como a *firewall* ou o *load-balancer*. Também pode ser algo tão simples como um problema de *routing*.

Problemas no fim da conexões, FIN. Terminações anormais de conexões podem dever-se a:

- Falta de recursos ou interrupção de *internet*.
- Uma quebra ou um *bug* na sessão.
- Quando um lado da conexão já terminou e fechou a mesma, mas o outro lado continua a enviar dados.
- O servidor recusa-se a abrir conexões com o cliente.

7 B1

’Provide two charts representing the average bit rate per time unit that crossed the interface or the router in and out during the collecting period.’

7.1 Explicação do código desenvolvido

A nossa abordagem para este problema foi tentar arranjar uma solução escalável para qualquer tipo de unidades de tempo, dado que para um ficheiro seria uma unidade de tempo de um minuto e outro com 5 segundos.

Implementamos então duas funções auxiliares para nos ajudar na recolha desta informação. Na figura 7.1 vemos a função *getTimeDiffInSecound(start, to)* onde dado um tempo inicial (start) e um tempo final (to), a função calcula a diferença em segundos desses tempos.

A função seguinte *getTimeFrameInSecound(file)* recebe um nome de um ficheiro e devolve a unidade de tempo, em segundos, que se pretende. Com esta arquitectura podemos acrescentar livremente novos ficheiros ou mesmo modificar os tempos de cada ficheiro sem ter que se modificar o código base da função b1. O resultado desta função é guardado na variável *timeFrame*.

Passando agora à explicação do código base da recolha de informação dos ficheiros, começamos por verificar o para cada tuplo (linha) o seu *start_time_* (a variável *end_time_* não é utilizada), que é o tempo inicial do fluxo que se encontra na primeira coluna dos ficheiros. Posteriormente incializamos a variavel *start_time* (diferente de *start_time_*) que será o tempo inicial do primeiro bloco de tempo a ser analizado.

A partir dai comparamos esse resultado ao *start_time_* de cada fluxo de modo a saber se já passou o tempo necessário para passar para o próximo bloco de tempo. Quando isso acontecer, passamos ao próximo bloco de tempo (counter += 1) e dizemos que agora o *start_time* agora é o *start_time_* do

```python
def getTimeDiffInSecound(start, to):
    # 28/04/2020  15:44:38
    fmt = "%Y-%m-%d %H:%M:%S"
    d1 = datetime.strptime(start, fmt)
    d2 = datetime.strptime(to, fmt)
    diff = d2 - d1
    return int(diff.total_seconds())


def getTimeFrameInSecound(file):
    if file == "Enunciado/www.fct.unl.pt.csv":
        return 60
    if file == "Enunciado/bigFlows.csv":
        return 5
```

Figura 7.1: *Funções auxiliares de b1*

primeiro fluxo do próximo bloco de tempo.

Durante cada bloco inicializamos a nossa estrutura de dados sempre que necessário e adicionamos a ela a informação necessária caso (como requerido pelo enunciado) os IP's sejam externos, utilizando a função *isExternalAddress(file, address)* já explicada nos capítulos anteriores.

```python
start_time_ = str(row[0])  # time start
end_time_ = str(row[1])  # time end

if line_count == 1:
    start_time = start_time_

time = getTimeDiffInSecound(start_time, start_time_)
if time >= timeFrame:
    counter += 1
    start_time = start_time_

source_addr = str(row[3])
dest_addr = str(row[4])
bytes_ = int(row[10])

if counter not in chart_by_time_out or counter not in chart_by_time_in:
    chart_by_time_out[counter] = 0
    chart_by_time_in[counter] = 0

if isExternalAddress(input_file_path, source_addr):
    chart_by_time_out[counter] += bytes_

if isExternalAddress(input_file_path, dest_addr):
    chart_by_time_in[counter] += bytes_
```

Figura 7.2: *Código do tópico b1*

7.2 Resultados obtidos pelo ficheiro www.fct.unl.pt.csv

Os resultados seguintes são o *output* do *byte rate* IN e OUT proposto por intervalos de tempo. Onde o tempo 0 para o ficheiro www.fct.unl.pt.csv equivale ao intervalo de tempo 2020-04-28 14:42:57 -> 2020-04-28 14:43:57 e o tempo 55 (último) equivale ao intervalo de tempo 28/04/2020 15:43:40 -> 28/04/2020 15:44:40, equivalente aos períodos do ficheiro.

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines from file: Enunciado/www.fct.unl.pt.csv

=========================RESULT==================================

IN: {0: 7468631, 1: 1662060, 2: 217226, 3: 6144206, 4: 5420714, 5:
 8297553, 6: 5880329, 7: 3136132, 8: 13619545, 9: 4658508, 10: 325
6179, 11: 19464509, 12: 7817714, 13: 9087638, 14: 3578767, 15: 115
87310, 16: 2198798, 17: 37311291, 18: 1846350, 19: 6687649, 20: 51
27693, 21: 3703533, 22: 3593115, 23: 15491303, 24: 1241014, 25: 72
86314, 26: 7924037, 27: 8558391, 28: 8343847, 29: 23670683, 30: 85
01650, 31: 3343076, 32: 1843815, 33: 12451821, 34: 7718990, 35: 42
30265, 36: 10481913, 37: 15209985, 38: 22540665, 39: 20404017, 40:
 3817252, 41: 3949143, 42: 3486027, 43: 7895144, 44: 3577391, 45:
2470962, 46: 2073520, 47: 1510486, 48: 12025999, 49: 6045943, 50:
16148318, 51: 5209537, 52: 11770234, 53: 3945743, 54: 20235448, 55
: 34748297}

OUT: {0: 286631, 1: 133610, 2: 19151, 3: 244122, 4: 282802, 5: 256
819, 6: 267145, 7: 167885, 8: 436269, 9: 185062, 10: 155214, 11: 4
21115, 12: 226684, 13: 382319, 14: 181760, 15: 513597, 16: 129616,
 17: 798625, 18: 111834, 19: 409659, 20: 221851, 21: 196698, 22: 2
31503, 23: 384851, 24: 101916, 25: 349906, 26: 325881, 27: 361225,
 28: 265360, 29: 449066, 30: 313789, 31: 161020, 32: 196188, 33: 4
96327, 34: 354919, 35: 276191, 36: 505775, 37: 447833, 38: 496681,
 39: 443751, 40: 202271, 41: 244489, 42: 142107, 43: 298231, 44: 2
08349, 45: 185978, 46: 202448, 47: 67709, 48: 384363, 49: 363946,
50: 499945, 51: 163807, 52: 429555, 53: 231815, 54: 752335, 55: 13
78505}
=================================================================
```

Figura 7.3: *Output do script b1.py*

Utilizando a biblioteca *matplotlib* [5] foi-nos possível e tempo de execução do código mostrar um gráfico visual dos dados da figura 7.3.

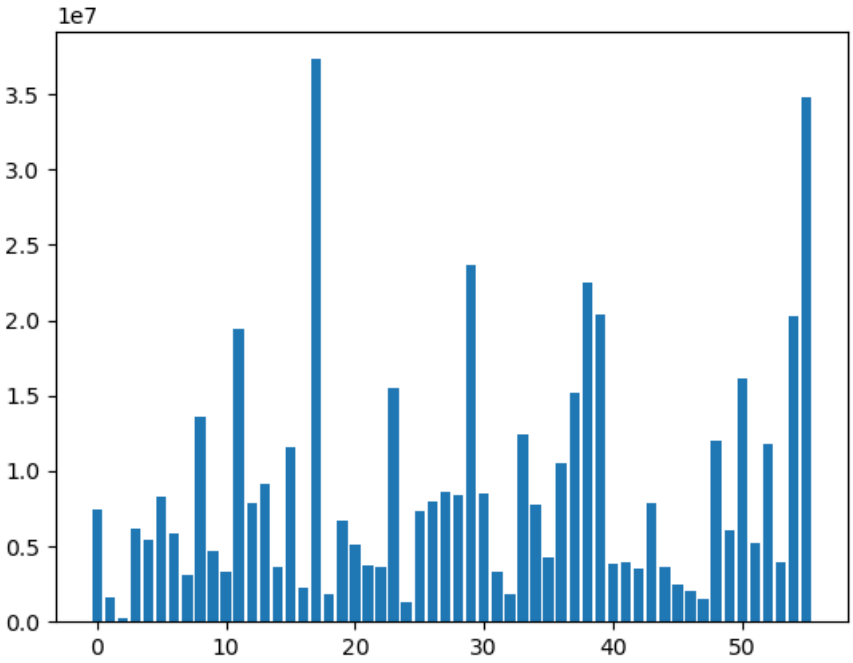

Figura 7.4: *Gráfico do fluxo médio de bytes de entrada*

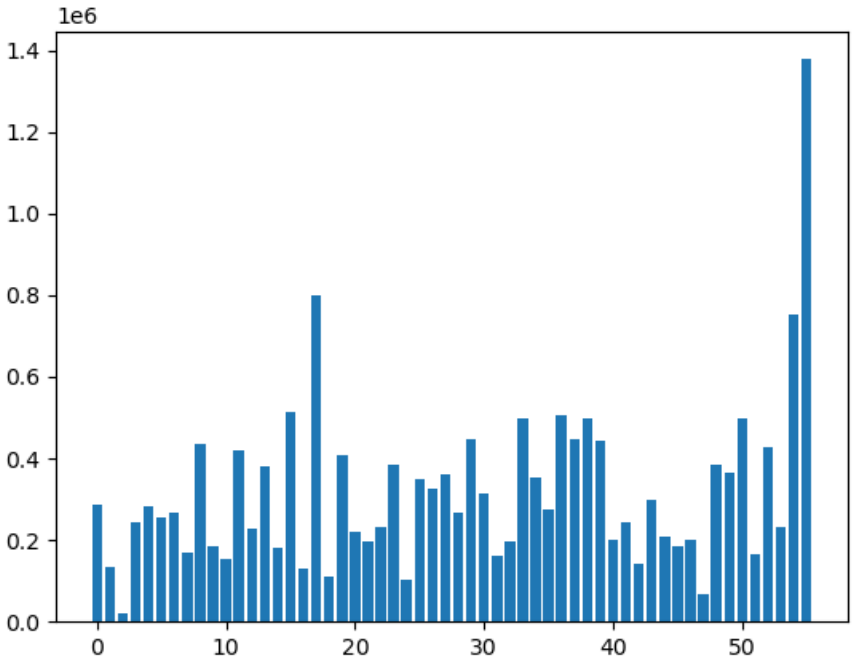

Figura 7.5: *Gráfico do fluxo médio de bytes de saída*

7.3 Resultados obtidos pelo ficheiro bigFlows.csv

Os resultados seguintes são o *output* do *byte rate* IN e OUT proposto por intervalos de tempo. Onde o tempo 0 para o ficheiro bigFlows.csv equivale ao intervalo de tempo 26/03/2020 20:27:35 -> 26/03/2020 20:27:40 e o tempo 56 (último) equivale ao intervalo de tempo 26/03/2020 20:32:30 -> 26/03/2020 20:32:35, equivalente aos períodos do ficheiro.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines from file: Enunciado/bigFlows.csv

========================RESULT===================================

IN: {0: 62866, 1: 57346, 2: 32417, 3: 19400, 4: 110510, 5: 15364,
6: 11507, 7: 29920, 8: 3633, 9: 213279, 10: 9977, 11: 10660, 12: 4
83607, 13: 346725, 14: 276853, 15: 275215, 16: 303466, 17: 483990,
 18: 458750, 19: 680452, 20: 97844, 21: 364741, 22: 11686790, 23:
970305, 24: 524573, 25: 994529, 26: 803051, 27: 545201, 28: 331528
, 29: 569680, 30: 578400, 31: 450931, 32: 294400, 33: 852837, 34:
495438, 35: 1082460, 36: 681093, 37: 403277, 38: 636214, 39: 40234
5, 40: 487382, 41: 333166, 42: 499602, 43: 354095, 44: 492388, 45:
 612029, 46: 742378, 47: 387171, 48: 824643, 49: 469783, 50: 10823
59, 51: 349836, 52: 544007, 53: 263043, 54: 742028, 55: 1163766, 5
6: 563579, 57: 35561638}

OUT: {0: 273041, 1: 131498, 2: 132440, 3: 41893, 4: 154549, 5: 354
48, 6: 86256, 7: 43323, 8: 1020, 9: 721916, 10: 82577, 11: 23837,
12: 4699341, 13: 2301673, 14: 1868443, 15: 826565, 16: 2165298, 17
: 622209, 18: 4293945, 19: 7700454, 20: 592549, 21: 1607406, 22: 1
0965906, 23: 1605295, 24: 427742, 25: 8787575, 26: 3136592, 27: 26
95815, 28: 357333, 29: 797258, 30: 2468346, 31: 434111, 32: 206832
, 33: 971237, 34: 827350, 35: 3185486, 36: 564486, 37: 479371, 38:
 2595831, 39: 1044228, 40: 868303, 41: 860193, 42: 1048684, 43: 42
2545, 44: 4940105, 45: 1830386, 46: 1760758, 47: 1582572, 48: 2204
487, 49: 1162526, 50: 4597968, 51: 836379, 52: 1780705, 53: 345368
, 54: 2434489, 55: 1247267, 56: 2494979, 57: 38565239}
=================================================================
```

Figura 7.6: *Output do script b1.py*

Utilizando a biblioteca *matplotlib* [5] foi-nos possível e tempo de execução do código mostrar um gráfico visual dos dados da figura 7.6.

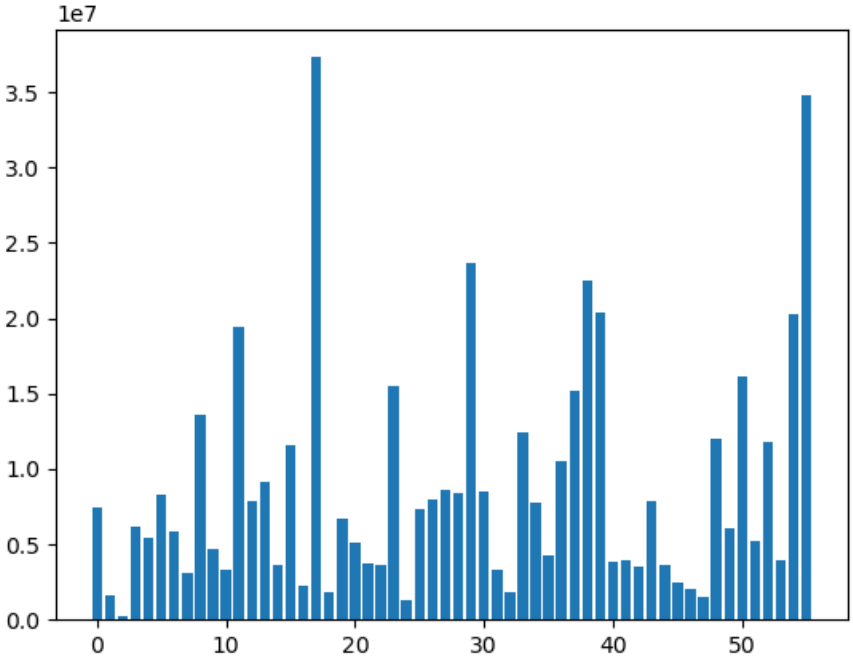

Figura 7.7: *Gráfico do fluxo médio de bytes de entrada*

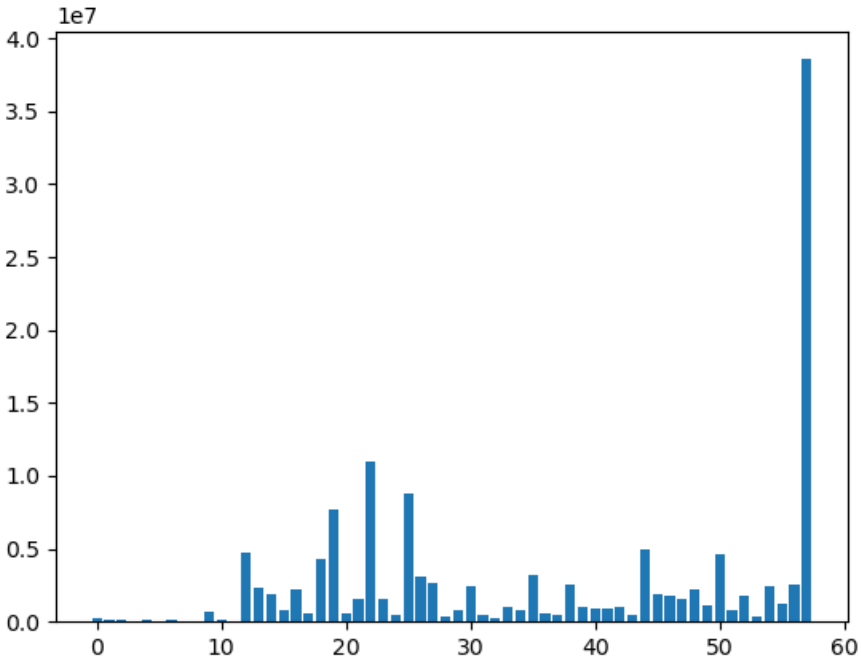

Figura 7.8: *Gráfico do fluxo médio de bytes de saída*

8

B2

Consider the two files and represent in a 2D chart the geographic position of the IP addresses outside the domain computed in a.3.

8.1 Explicação do código desenvolvido

Para o desenvolvimento deste tópico decidimos utilizar a API de Geolocalização [2] que nos forneceu algum código (IPGeolocation.py) de modo a facilitar a sua utilização. Na figura 8.1 pode-se ver o código criado por nós para a conversão dos IP's por localização geográfica.

Dado que a API só nos deixava utilizar alguns recursos por dia, fomos obrigados a utilizar duas contas para gerar duas chaves de modo a cobrir o nosso código com o maior número de testes possível. Esta função *getGeoLocation(ips_addrs)* recebe um conjunto de IP's (já computados previamente) e converte-os para uma localização geográfica. Utilizamos mais uma vez um mapa para sabermos quantas vezes aparecem IP's de uma dada localização, sendo este mapa no fim ordenado da localidade mais utilizada, para a menos utilizada.

Na figura 8.2 podemos verificar como esta função é utilizada depois da computação dos IP's e a forma como essa informação é ordenada.

Passando agora à analise do código que itera todas as linhas do ficheiro, podemos verificar que o código é muito simples de se entender. Inicialmente inicializamos as variáveis úteis, e caso se trate de um fluxo exterior (de origem ou destino) inicializamos a nossa estrutura de dados (mapa) e adicionamos o IP's associado.

```python
def getGeoLocation(ips_addrs):
    key1 = "2b86ff119e424fb1b83352f12b1344aa"
    key2 = "dad4b6d19dd04b05a5becebc78404901"
    ipgeolocationApi = IPGeolocationAPI(key2)

    ip_geo_location = {}
    i = 0

    for ip in ips_addrs:
        geolocation = ipgeolocationApi.getGeolocation()
        geolocationParams = GeolocationParams()
        geolocationParams.setIPAddress(ip)
        geolocation = ipgeolocationApi.getGeolocation(geolocationParams)
        country_name = geolocation["country_name"]

        if country_name not in ip_geo_location:
            ip_geo_location[country_name] = int(0)

        ip_geo_location[country_name] += int(1)

        print("                                ", end="\r")
        print("[" + str(i) + "]" + country_name, end="\r")
        i += 1

    sorted_list = {
        k: v
        for k, v in sorted(
            ip_geo_location.items(), key=lambda item: item[1], reverse=True
        )
    }
    return sorted_list
```

Figura 8.1: *Código do tópico b2*

```python
sorted_list = sorted(ip_addrs.items(), key=lambda x: x[1], reverse=True)[0:50]
ips_addrs = []
for ip_bytes in sorted_list:
    ips_addrs.append(ip_bytes[0])
print(f"[*] Processed {line_count} lines from file: " + input_file_path)
print("\n=========================RESULT================================")
data = getGeoLocation(ips_addrs)
print(str(data))
getGraph(data)
print("==============================================================")
print("\n\n")
```

Figura 8.2: *Print e gráfico do resultado b2*

8.2 Resultados obtidos pelo ficheiro www.fct.unl.pt.csv

Para os resultados do ficheiro www.fct.unl.pt.csv era de esperar que Portugal dominasse como localização mais comum entre os IP's. Nas figuras 8.4 e 8.5 podemos ver os resultados em forma de gráfico e em forma de mapa.

```
source_address = str(row[3])
destination_address = str(row[4])
bytes_ = int(row[10])

if isExternalAddress(input_file_path, source_address):
    if source_address not in ip_addrs:
        # print("[+] New ip addr found: " + source_address)
        ip_addrs[source_address] = int(0)
    ip_addrs[source_address] += bytes_

if isExternalAddress(input_file_path, destination_address):
    if destination_address not in ip_addrs:
        # print("[+] New ip addr found: " + destination_address)
        ip_addrs[destination_address] = int(0)
    ip_addrs[destination_address] += bytes_
```

Figura 8.3: *Código do tópico b2*

```
[*] File: Enunciado/www.fct.unl.pt.csv
[*] Processed 21362 lines from file: Enunciado/www.fct.unl.pt.csv

=========================RESULT==================================

{'Portugal': 38, 'United States': 4, 'Russia': 2, 'Canada': 1, 'So
uth Africa': 1, 'China': 1, 'France': 1, 'Brazil': 1, 'Netherlands
': 1}
================================================================
```

Figura 8.4: *Output do script b2.py*

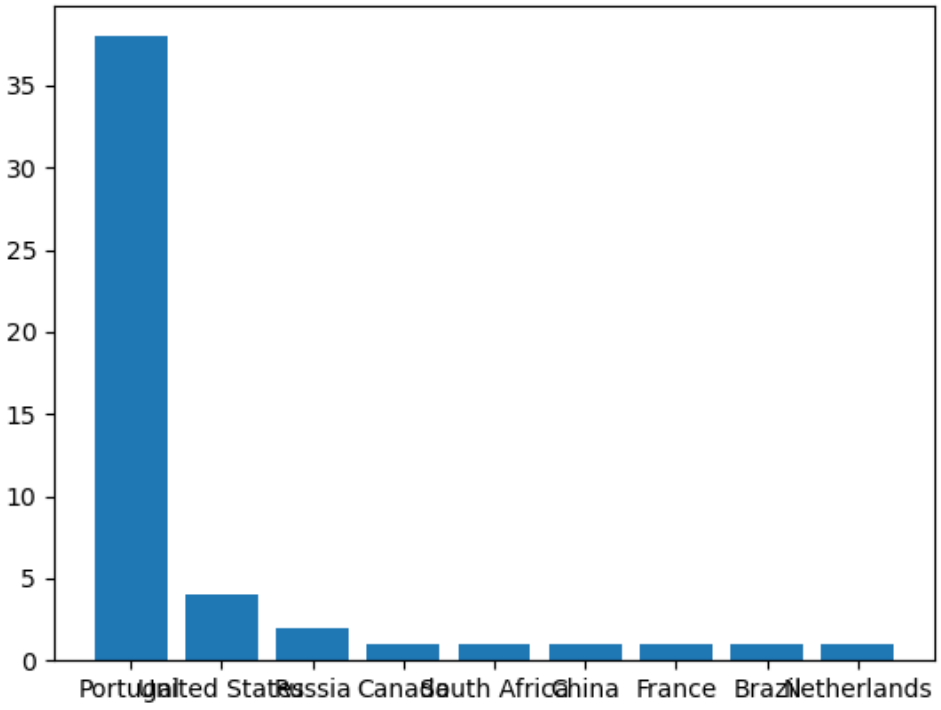

Figura 8.5: *Gráfico das posições geográficas do endereços IP*

8.3 Resultados obtidos pelo ficheiro bigFlows.csv

Para os resultados do ficheiro bigFlows.csv os Estados Unidos dominam consideravelmente, estando os resultados apresentados nas figuras 8.6 e 8.7 seguintes.

```
[*] File: Enunciado/bigFlows.csv
[*] Processed 42845 lines from file: Enunciado/bigFlows.csv

=========================RESULT==================================

{'United States': 46, 'France': 2, 'Canada': 1, 'United Kingdom':
1}
================================================================
```

Figura 8.6: *Output do script b2.py*

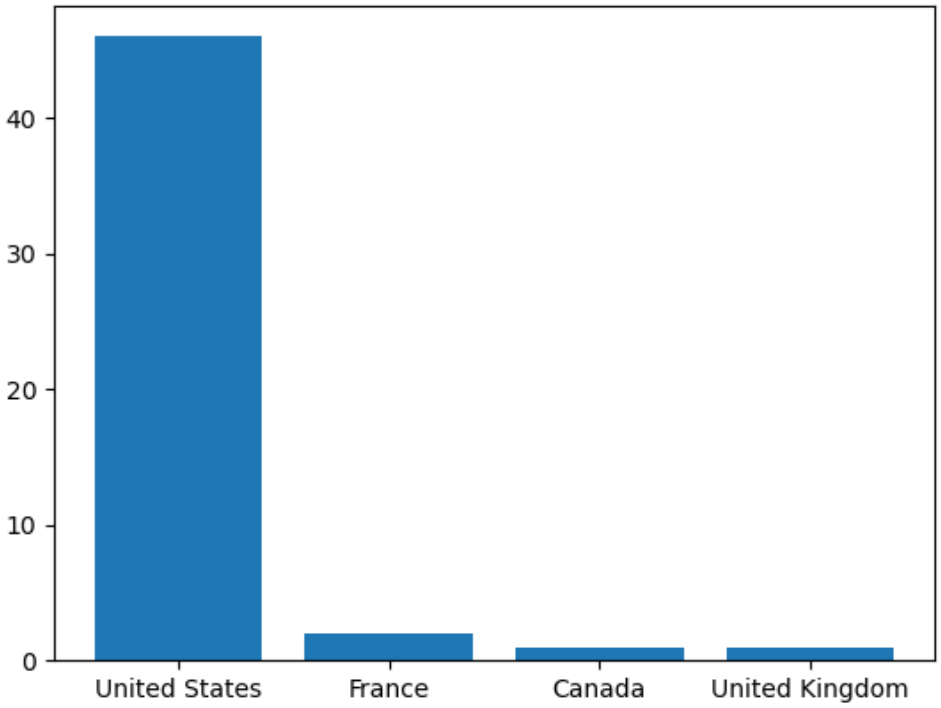

Figura 8.7: *Gráfico das posições geográficas do endereços IP*

9 Conclusão

Para concluir, todos os tópicos foram concluídos com sucesso e explicados da melhor forma possível. Houve por parte dos alunos uma nova aprendizagem mais profunda da linguagem *Python* assim como uma nova aprendizagem de como filtrar e ler informações sobre uma recolha de pacotes de uma rede. Achamos que os resultados poderiam ter sido analisados de uma forma mais óptima se fosse usada uma base de dados e posteriormente feitas as perguntas dos tópicos, mas em termos de tempo foi mais rápido implementar uma procura linear de complexidade 0(n) sobre os ficheiros fornecidos.

Bibliografia

[1] https://www.python.org/download/releases/3.0/

[2] https://github.com/IPGeolocation/ip-geolocation-api-python-sdk
https://api.ipgeolocation.io/

[3] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[4] https://www.sqlite.org/index.html

[5] https://matplotlib.org/

´

A

A1-Anexos

```
import csv

def a1(input_file_path):

    protocols_bytes = {}
    protocols_packets = {}

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                protocol = row[7]
                packets = row[9]
                bytes_ = row[10]

                if (protocol not in protocols_bytes
                    or protocol not in protocols_packets):

                    # print("[+] New protocol found: " + str(protocol))
                    protocols_bytes[str(protocol)] = int(0)
                    protocols_packets[str(protocol)] = int(0)

                protocols_bytes[str(protocol)] += int(bytes_)
                protocols_packets[str(protocol)] += int(packets)
            line_count += 1
    print(f"[*] Processed {line_count} lines")
    print("\n======RESULT======")
    print("By Bytes:")
```

```
    print(protocols_bytes)
    print("\nBy Packets:")
    print(protocols_packets)
    print("====================")
    print("\n\n")

def main():
    print("\n\n")
    print(
        "[Compute the total number of packets and bytes (in and out) per"
        +" protocol (TCP, UDP, ICMP,     ) contained in the flows.]\n"
    )
    a1("Enunciado/www.fct.unl.pt.csv")
    a1("Enunciado/bigFlows.csv")

if __name__ == "__main__":
    main()
```

B

A2-Anexos

```
import csv


def a2(input_file_path):

    ip_addrs = {}

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                source_address = str(row[3])
                destination_address = str(row[4])

                if isExternalAddress(input_file_path, source_address):
                    if source_address not in ip_addrs:
                        # print("[+] New ip addr found: " + source_address)
                        ip_addrs[source_address] = int(0)
                    ip_addrs[source_address] += int(1)

                if isExternalAddress(input_file_path, destination_address):
                    if destination_address not in ip_addrs:
                        # print("[+] New ip addr found: "
                        #+ destination_address)
                        ip_addrs[destination_address] = int(0)
                    ip_addrs[destination_address] += int(1)
            line_count += 1
    print(f"[*] Processed {line_count} lines.")
```

```
    print("\n======RESULT============")
    print(sorted(ip_addrs.items(), key=lambda x: x[1], reverse=True)[0:50])
    print("============================")
    print("\n\n")


def isExternalAddress(file, address):
    if file == "Enunciado/www.fct.unl.pt.csv":
        if address != "193.136.126.43" and address.split(".")[0] != "10":
            return True

    if file == "Enunciado/bigFlows.csv":
        addr_split = address.split(".")
        if addr_split[0] != "172" and (
            addr_split[1] != "16" or addr_split[1] != "31"):
            return True
    return False


def main():

    print("\n\n")
    print(
        "[Determine the 50 most popular IP addresses external"
        + " to the domain by number of flows..]\n"
    )
    a2("Enunciado/www.fct.unl.pt.csv")
    a2("Enunciado/bigFlows.csv")


if __name__ == "__main__":
    main()
```

C

A3-Anexos

```
import csv


def a3(input_file_path):

    ip_addrs = {}

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                source_address = str(row[3])
                destination_address = str(row[4])
                bytes_ = int(row[10])

                if isExternalAddress(input_file_path, source_address):
                    if source_address not in ip_addrs:
                        # print("[+] New ip addr found: " + source_address)
                        ip_addrs[source_address] = 0
                    ip_addrs[source_address] += bytes_

                if isExternalAddress(input_file_path, destination_address):
                    if destination_address not in ip_addrs:
                        # print("[+] New ip addr found: "
                        #+ destination_address)
                        ip_addrs[destination_address] = 0
                    ip_addrs[destination_address] += bytes_
            line_count += 1
```

```
    print(f"[*] Processed {line_count} lines.")
    print("\n==========RESULT============")
    print(sorted(ip_addrs.items(), key=lambda x: x[1], reverse=True)[0:50])
    print("============================")
    print("\n\n")


def isExternalAddress(file, address):
    if file == "Enunciado/www.fct.unl.pt.csv":
        if address != "193.136.126.43" and address.split(".")[0] != "10":
            return True

    if file == "Enunciado/bigFlows.csv":
        addr_split = address.split(".")
        if addr_split[0] != "172" and (
            addr_split[1] != "16" or addr_split[1] != "31"):
            return True
    return False


def main():
    print("\n\n")
    print(
        "[Determine the 50 most popular IP addresses external"
        + " to the domain by number of bytes.]\n"
    )
    a3("Enunciado/www.fct.unl.pt.csv")
    a3("Enunciado/bigFlows.csv")


if __name__ == "__main__":
    main()
```

D

A4-Anexos

```
import csv
import pandas as pd


def a4(input_file_path, port_dic):

    ports_usage_flows = {}
    ports_usage_packets = {}

    def init_or_insert_variables(port, nr_packets):
        if port not in ports_usage_flows or port not in ports_usage_packets:
            # print("[+] New port found: " + source_port)
            ports_usage_flows[port] = 0
            ports_usage_packets[port] = 0

        ports_usage_flows[port] += 1
        ports_usage_packets[port] += nr_packets

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                source_port = int(row[5])
                destination_port = int(row[6])

                packets = int(row[9])

                init_or_insert_variables(source_port, packets)
```

```
                init_or_insert_variables(destination_port, packets)

            line_count += 1
    print(f"[*] Processed {line_count} lines.")

    dataFlows = sort_to_key_value(
        convertPortNumbers(ports_usage_flows, port_dic))
    dataPackets = sort_to_key_value(
        convertPortNumbers(ports_usage_packets, port_dic))

    print("\n======RESULT=========")
    print("By Flows:")
    print(pd.DataFrame(dataFlows.items(), columns=["Port", "Flows"]))

    print("\nBy Packets:")
    print(pd.DataFrame(dataPackets.items(), columns=["Port", "Packets"]))
    print("======================")
    print("\n\n")


def sort_to_key_value(data):
    return {
        k: v
        for k, v in sorted(data.items(),
            key=lambda item: item[1], reverse=True)[0:50]
    }


def convertPortNumbers(data, port_dic):
    for i in port_dic.keys():
        value = port_dic[i]
        if value == "":
            value = i
        try:
            port_n = int(i)
            if port_n in data:
                data[value + "(" + str(port_n) + ")"] = data.pop(port_n)
        except ValueError:
            continue
    return data


def getPortNamesFromFile():
    dic = {}
```

```
    with open("Utils/service-names-port-numbers.csv", mode="r") as infile:
        reader = csv.reader(infile)
        line = 0
        for rows in reader:
            if line != 0:
                dic[rows[1]] = rows[0]
            line += 1
    return dic


def main():
    print("\n\n")
    print(
        "[What are the top 50 (or less if their variety "
        + "is smaller) most popular "
        + "applications used by the computers in the domain?]\n"
    )
    # Only for the bigFlows.csv file:
    a4("Enunciado/bigFlows.csv", getPortNamesFromFile())


if __name__ == "__main__":
    main()
```

E

A5-Anexos

```
import csv


def a5(input_file_path):

    count_tcp = 0
    count_conections = 0

    line1 = {}
    line2 = {}
    tmp = 0

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                protocol = str(row[7])
                if protocol == "TCP":
                    if tmp == 0:
                        line1 = row
                        tmp = 1
                    else:
                        line2 = row

                        src_addr1 = str(line1[3])
                        dest_addr1 = str(line1[4])
                        src_port1 = int(line1[5])
                        dest_port1 = int(line1[6])
```

```
                            flags1 = str(line1[8])

                            src_addr2 = str(line2[3])
                            dest_addr2 = str(line2[4])
                            src_port2 = int(line2[5])
                            dest_port2 = int(line2[6])
                            flags2 = str(line2[8])

                            if (
                                src_addr1 == dest_addr2
                                and dest_addr1 == src_addr2
                                and src_port1 == dest_port2
                                and dest_port1 == src_port2
                            ):
                                count_tcp += 1
                                if (
                                    "S" in flags1
                                    and "F" in flags1
                                    and "A" in flags1
                                    and "S" in flags2
                                    and "F" in flags2
                                    and "A" in flags2
                                ):
                                    count_conections += 1
                                tmp = 0
                            else:
                                line1 = row
                                tmp = 1
                    line_count += 1
        print(f"[*] Processed {line_count} lines")
        print("\n==========RESULT==========")
        print("Conexes TCP (all): " + str(count_tcp))
        print("Conexes TCP (started and finished): " + str(count_conections))
        print("==========================")
        print("\n\n")


def main():
    print("\n\n")
    print(
        "[Aggregate the two flows representing the same TCP"
        + " connection; count the total number of TCP connections "
        + "collected, and the total number of TCP connections "
        + "that started and finished correctly (the ones where flags show "
```

```
        + "that the connection has been opened, used, and finalized "
        + "by both sides). Explain the several possible technical reasons"
        + " that justify why there are TCP connections that were "
        + "not started or ended correctly.]\n"
    )
    a5("Enunciado/www.fct.unl.pt.csv")
    a5("Enunciado/bigFlows.csv")


if __name__ == "__main__":
    main()
```

F

B1-Anexos

```
import csv
from datetime import datetime
from datetime import timedelta
import matplotlib.pyplot as plt


def b1(input_file_path):

    chart_by_time_in = {}
    chart_by_time_out = {}

    start_time = ""
    counter = 0

    timeFrame = getTimeFrameInSecound(input_file_path)

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        csv_line = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                start_time_ = str(row[0])  # time start
                end_time_ = str(row[1])  # time end

                if line_count == 1:
                    start_time = start_time_

                time = getTimeDiffInSecound(start_time, start_time_)
```

```
                    if time >= timeFrame:
                        counter += 1
                        start_time = start_time_

                    source_addr = str(row[3])
                    dest_addr = str(row[4])
                    bytes_ = int(row[10])

                    if (counter not in chart_by_time_out
                        or counter not in chart_by_time_in):
                        chart_by_time_out[counter] = 0
                        chart_by_time_in[counter] = 0

                    if isExternalAddress(input_file_path, source_addr):
                        chart_by_time_out[counter] += bytes_

                    if isExternalAddress(input_file_path, dest_addr):
                        chart_by_time_in[counter] += bytes_

            line_count += 1
    print(f"[*] Processed {line_count} lines from file: " + input_file_path)

    print("\n==========RESULT==========")
    print("IN: " + str(chart_by_time_in))
    getGraph(chart_by_time_in)
    print("\nOUT: " + str(chart_by_time_out))
    getGraph(chart_by_time_out)
    print("==================================")
    print("\n\n")


def getGraph(data):
    keys = data.keys()
    values = data.values()
    plt.bar(keys, values)
    plt.show()


def getTimeDiffInSecound(start, to):
    # 28/04/2020  15:44:38
    fmt = "%Y-%m-%d %H:%M:%S"
    d1 = datetime.strptime(start, fmt)
    d2 = datetime.strptime(to, fmt)
    diff = d2 - d1
```

```
    return int(diff.total_seconds())


def getTimeFrameInSecound(file):
    if file == "Enunciado/www.fct.unl.pt.csv":
        return 60
    if file == "Enunciado/bigFlows.csv":
        return 5


def isExternalAddress(file, address):
    if file == "Enunciado/www.fct.unl.pt.csv":
        if address != "193.136.126.43" and address.split(".")[0] != "10":
            return True

    if file == "Enunciado/bigFlows.csv":
        addr_split = address.split(".")
        if (addr_split[0] != "172" and
            (addr_split[1] != "16" or addr_split[1] != "31")):
            return True
    return False


def main():
    print("\n\n")
    print(
        "[Option 1: Provide two charts representing the "
        + "average bit rate per time unit that "
        + "crossed the interface or the router in and out "
        + "during the collecting period. The "
        + "resolution of these charts in the horizontal axe "
        + "(time) should contain at least 60 "
        + "bars, or values. Thus, if the collection period is "
        + "1 hour, each bar represents at most "
        + "1 minute. If the collection period is 5 minutes, each "
        + "bar represents at most 5 seconds.]\n"
    )
    b1("Enunciado/www.fct.unl.pt.csv")
    b1("Enunciado/bigFlows.csv")


if __name__ == "__main__":
    main()
```

G
B2-Anexos

```
import csv
from IPGeolocation import IPGeolocationAPI
from IPGeolocation import GeolocationParams
import matplotlib.pyplot as plt


def b2(input_file_path):

    ip_addrs = {}

    with open(input_file_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                print("[*] File: " + input_file_path)
            else:
                source_address = str(row[3])
                destination_address = str(row[4])
                bytes_ = int(row[10])

                if isExternalAddress(input_file_path, source_address):
                    if source_address not in ip_addrs:
                        # print("[+] New ip addr found: " + source_address)
                        ip_addrs[source_address] = int(0)
                    ip_addrs[source_address] += bytes_

                if isExternalAddress(input_file_path, destination_address):
                    if destination_address not in ip_addrs:
                        # print("[+] New ip addr found: "
                        # + destination_address)
```

```
                        ip_addrs[destination_address] = int(0)
                    ip_addrs[destination_address] += bytes_

            line_count += 1

    sorted_list = sorted(ip_addrs.items(),
        key=lambda x: x[1], reverse=True)[0:50]
    ips_addrs = []
    for ip_bytes in sorted_list:
        ips_addrs.append(ip_bytes[0])
    print(f"[*] Processed {line_count} lines from file: " + input_file_path)
    print("\n==============RESULT==============")
    data = getGeoLocation(ips_addrs)
    print(str(data))
    getGraph(data)
    print("====================================")
    print("\n\n")


def isExternalAddress(file, address):
    if file == "Enunciado/www.fct.unl.pt.csv":
        if address != "193.136.126.43" and address.split(".")[0] != "10":
            return True

    if file == "Enunciado/bigFlows.csv":
        addr_split = address.split(".")
        if (addr_split[0] != "172"
            and (addr_split[1] != "16" or addr_split[1] != "31")):
            return True
    return False


def getGraph(data):
    keys = data.keys()
    values = data.values()
    plt.bar(keys, values)
    plt.show()


def getGeoLocation(ips_addrs):
    key1 = "2b86ff119e424fb1b83352f12b1344aa"
    key2 = "dad4b6d19dd04b05a5becebc78404901"
    ipgeolocationApi = IPGeolocationAPI(key2)
```

```
    ip_geo_location = {}
    i = 0

    for ip in ips_addrs:
        geolocation = ipgeolocationApi.getGeolocation()
        geolocationParams = GeolocationParams()
        geolocationParams.setIPAddress(ip)
        geolocation = ipgeolocationApi.getGeolocation(geolocationParams)
        country_name = geolocation["country_name"]

        if country_name not in ip_geo_location:
            ip_geo_location[country_name] = int(0)

        ip_geo_location[country_name] += int(1)

        print("                                    ", end="\r")
        print("[" + str(i) + "]" + country_name, end="\r")
        i += 1

    sorted_list = {
        k: v
        for k, v in sorted(
            ip_geo_location.items(), key=lambda item: item[1], reverse=True
        )
    }
    return sorted_list


def main():
    print("\n\n")
    print(
        "[Option 2: Consider the two files and represent in a 2D chart the"
        + " geographic position of the IP addresses outside the domain"
        + " computed in a.3.]\n"
    )
    b2("Enunciado/www.fct.unl.pt.csv")
    b2("Enunciado/bigFlows.csv")


if __name__ == "__main__":
    main()
```