EA080 O - Atividade 5

Pedro Bueno de Castro RA:118355

Gustavo Ciotto Pinton RA:117136

Exercício 0

A primeira atividade a ser realizada foi a configuração das máquinas virtuais instanciadas nos *hosts* físicos presentes no laboratório, conforme figura 4 do enunciado. No nosso caso, ficamos responsáveis pelas máquinas da primeira fileira, isto é, *hosts* 101, 106, 111 e 116.

Conforme a figura 4, o *host* 101 possui duas máquinas virtuais sendo executadas: um roteador e um *host* cujo sistema operacional é o Ubuntu. Para o *host* virtual, de acordo com as orientações dadas em aula, o endereço IP escolhido foi *20.0.1.190*, sendo que o endereço da sub-rede é determinado pelos 24 bits mais siginificativos que compõem cada endereço, isto é, *20.190.1.0/24*. Além disso, foi necessária a configuração do *gateway* padrão, que foi modificado para o endereço da *eth4* do roteador virtual. Para esta interface, foi escolhido o endereço *20.0.1.21/24* e para a *eth1,* a fim de evitar conflito com o endereço da interface *eth2* do roteador físico 21, cujo endereço já é *10.0.1.21*, foi escolhido *10.0.1.**211***. Ainda para o roteador virtual, destaca-se que a ativação do protocolo OSPF na *eth1* também foi configurada com o intuito de habilitar a comunicação entre todos os roteadores, permitindo assim a determinação das respectivas rotas entre as sub-redes. Sendo assim, após a configuração de todos os *hosts*, obtém-se as figuras 1 e 2 a seguir, que mostram, respectivamente, o resultado do *ping* entre os *hosts x* e *y,* e a tabela de roteamento do roteador virtual.

```
ea080@maumagal-VirtualBox:~$ ping -c 5  30.0.1.190
PING 30.0.1.190 (30.0.1.190) 56(84) bytes of data.
64 bytes from 30.0.1.190: icmp_req=1 ttl=60 time=1.96 ms
64 bytes from 30.0.1.190: icmp_req=2 ttl=60 time=1.77 ms
64 bytes from 30.0.1.190: icmp_req=3 ttl=60 time=1.73 ms
64 bytes from 30.0.1.190: icmp_req=4 ttl=60 time=1.52 ms
64 bytes from 30.0.1.190: icmp_req=5 ttl=60 time=1.49 ms

--- 30.0.1.190 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 1.496/1.700/1.966/0.172 ms
```

Figura 1: Resultado do *ping* entre hosts *20.0.1.190* e *30.0.1.190*

```
[admin@MikroTik] > ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 A S  0.0.0.0/0                          10.0.1.26                 1
 1 ADC  10.0.1.0/24        10.0.1.211      ether1                    0
 2 ADo  10.0.2.0/24                        10.0.1.21               110
 3 ADo  10.0.3.0/24                        10.0.1.21               110
 4 ADo  10.0.4.0/24                        10.0.1.21               110
 5 ADo  10.0.5.0/24                        10.0.1.21               110
 6 ADC  20.0.1.0/24        20.0.1.21       ether4                    0
 7 ADo  30.0.1.0/24                        10.0.1.21               110
```

Figura 2: Tabela de roteamento para o roteador virtual.

Observa-se na figura 1 que o valor do atributo **TTL** vale 60, significando que os pacotes passaram por 4 roteadores antes de alcançarem o destino. Conforme figura 4 do enunciado, este valor está correto e corresponde à rota *Roteador virtual 1* <-> *R21* <-> *R22* <-> *Roteador virtual 2*. Na figura 2, destaca-se que rotas para todas as sub-redes presentes na topologia foram corretamente construídas, indicando que a configuração do protocolo OSPF foi bem sucedida.

Para o host virtual conectado à interface *eth4* de R21, o endereço IP configurado foi *10.0.4.190/24* e seu *gateway* padrão foi setado para o endereço de *eth4*, isto é, *10.0.4.21/24*. As figuras 3 e 4 abaixo possuem os resultados dos comandos *ping* para *20.0.1.190* e *30.0.1.190* e, conforme esperado, os atributos **TTL** valem respectivamente 62 e 61. Esse fato é conforme pela topologia de rede proposta: para atingir a sub-rede *20.0.1.0/24*, são necessários apenas 2 roteadores, enquanto para *30.0.1.0/24*, utiliza-se 3.

```
ea080@maumagal-VirtualBox:~$ ping 20.0.1.190 -c 5
PING 20.0.1.190 (20.0.1.190) 56(84) bytes of data.
64 bytes from 20.0.1.190: icmp_req=1 ttl=62 time=0.893 ms
64 bytes from 20.0.1.190: icmp_req=2 ttl=62 time=1.22 ms
64 bytes from 20.0.1.190: icmp_req=3 ttl=62 time=1.30 ms
64 bytes from 20.0.1.190: icmp_req=4 ttl=62 time=1.24 ms
64 bytes from 20.0.1.190: icmp_req=5 ttl=62 time=1.21 ms

--- 20.0.1.190 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 0.893/1.177/1.305/0.148 ms
```

Figura 3: Resultado do *ping* entre hosts *10.0.4.190* e *20.0.1.190*. **TTL** vale 62.

```
ea080@maumagal-VirtualBox:~$ ping 30.0.1.190 -c 5
PING 30.0.1.190 (30.0.1.190) 56(84) bytes of data.
64 bytes from 30.0.1.190: icmp_req=1 ttl=61 time=2.06 ms
64 bytes from 30.0.1.190: icmp_req=2 ttl=61 time=1.26 ms
64 bytes from 30.0.1.190: icmp_req=3 ttl=61 time=1.30 ms
64 bytes from 30.0.1.190: icmp_req=4 ttl=61 time=1.28 ms
64 bytes from 30.0.1.190: icmp_req=5 ttl=61 time=1.07 ms

--- 30.0.1.190 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 1.070/1.394/2.060/0.343 ms
```

Figura 4: Resultado do *ping* entre hosts *10.0.4.190* e *30.0.1.190*. **TTL** vale 61.

O mesmo processo se repete para o *host* virtual conectado à *eth4* de R22. Atribui-se a ele o endereço IP *10.0.5.190/24* e configura-se seu *gateway* padrão para esta mesma interface (*10.0.5.22/24*). O comando *ping* para o *host 20.0.1.190* está ilustrado abaixo na figura 5. Oposto ao caso precedente, o **TTL** vale 61, indicando que a rota é composta por 3 roteadores. Essa afirmação é novamente confirmada pela topologia de rede proposta para este experimento.

```
maumagal@maumagal-VirtualBox:~$ ping 20.0.1.190
PING 20.0.1.190 (20.0.1.190) 56(84) bytes of data.
64 bytes from 20.0.1.190: icmp_req=1 ttl=61 time=2.05 ms
64 bytes from 20.0.1.190: icmp_req=2 ttl=61 time=1.39 ms
64 bytes from 20.0.1.190: icmp_req=3 ttl=61 time=1.39 ms
64 bytes from 20.0.1.190: icmp_req=4 ttl=61 time=1.43 ms
64 bytes from 20.0.1.190: icmp_req=5 ttl=61 time=1.43 ms
64 bytes from 20.0.1.190: icmp_req=6 ttl=61 time=1.48 ms
64 bytes from 20.0.1.190: icmp_req=7 ttl=61 time=1.16 ms
64 bytes from 20.0.1.190: icmp_req=8 ttl=61 time=1.38 ms
64 bytes from 20.0.1.190: icmp_req=9 ttl=61 time=1.37 ms
```

Figura 5: Resultado do *ping* entre hosts *10.0.5.190* e *20.0.1.190*. **TTL** vale 61.

Finalmente, para o *host* físico 116, que contém um *host* e um roteador virtuais, configura-se o endereço IP do primeiro para *30.0.1.190/24* e seu *gateway* padrão para a interface *eth4* do roteador virtual, cujo endereço foi configurado para *30.0.1.22/24*. *Eth1* tem seu endereço setado para *10.0.3.222/24*. Enfim, habilita-se o protocolo OSPF na interface *eth1* do roteador virtual para que os roteadores possam se comunicar. Assim como no primeiro caso, capturamos a tabela de roteamento do roteador virtual e o resultado de um *ping* entre os dois *hosts* virtuais, representados, respectivamente, pelas figuras 6 e 7.

```
[admin@MikroTik] > ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADo  10.0.1.0/24                        10.0.3.22               110
 1 ADo  10.0.2.0/24                        10.0.3.22               110
 2 ADC  10.0.3.0/24        10.0.3.222      ether1                    0
 3 ADo  10.0.4.0/24                        10.0.3.22               110
 4 ADo  10.0.5.0/24                        10.0.3.22               110
 5 ADo  20.0.1.0/24                        10.0.3.22               110
 6 ADC  30.0.1.0/24        30.0.1.1        ether4                    0
```

Figura 6: Tabela de roteamento para o roteador virtual.

```
maumagal@maumagal-VirtualBox:~$ ping 20.0.1.190
PING 20.0.1.190 (20.0.1.190) 56(84) bytes of data.
64 bytes from 20.0.1.190: icmp_req=1 ttl=60 time=1.63 ms
64 bytes from 20.0.1.190: icmp_req=2 ttl=60 time=1.49 ms
64 bytes from 20.0.1.190: icmp_req=3 ttl=60 time=1.85 ms
64 bytes from 20.0.1.190: icmp_req=4 ttl=60 time=1.89 ms
64 bytes from 20.0.1.190: icmp_req=5 ttl=60 time=1.57 ms
64 bytes from 20.0.1.190: icmp_req=6 ttl=60 time=4.36 ms
64 bytes from 20.0.1.190: icmp_req=7 ttl=60 time=1.80 ms
64 bytes from 20.0.1.190: icmp_req=8 ttl=60 time=1.65 ms
```

Figura 7: Resultado do *ping* entre hosts *30.0.1.190* e *20.0.1.190*

Assim como no primeiro caso, o atributo **TTL** vale 60 indicando que os pacotes passaram por 4 roteadores.

Exercício 1

Uma vez que a topologia da rede foi testada e validada, podemos começar a capturar o tráfego UDP e TCP da rede. Para tal, habilita-se um ***receptor*** **`ITGRecv`** no *host virtual* ***x,*** isto é, no *host* virtual cujo endereço é ***20.0.1.190/24,*** e um ***transmissor*** **`ITGSend`** no *host virtual* ***y,*** cujo endereço é ***30.0.1.190/24.*** Habilita-se igualmente o *Wireshark* no ***receptor*** e monitora-se somente o tráfego UDP. Para esta etapa, utiliza-se um tráfego do tipo UDP, a uma taxa de 1000 pacotes por segundo, com dado de 1200 bytes e duração de transmissão de 30000 ms. O arquivo de *log* gerado no ***receptor*** pelo `ITGDec` está representado na tabela 1 abaixo:

Tabela 1: *Log* gerado por ITGDec no *receptor* 20.0.1.190 para tráfego UDP

```
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
------------------------------------------------------------
Flow number: 1
From 30.0.1.190:33432
To   20.0.1.190:8999
------------------------------------------------------------
Total time                =     29.999999 s
Total packets             =     25273
```

```
Minimum delay              =      0.158879 s
Maximum delay              =      0.170885 s
Average delay              =      0.159968 s
Average jitter             =      0.000175 s
Delay standard deviation    =     0.000510 s
Bytes received             =      30327600
Average bitrate            =   8087.360270 Kbit/s
Average packet rate        =      842.433361 pkt/s
Packets dropped            =            0 (0.00 %)
Average loss-burst size =          0.000000 pkt
------------------------------------------------------------


------------------------------------------------------------
****************  TOTAL RESULTS   ******************

------------------------------------------------------------
Number of flows            =            1
Total time                 =      29.999999 s
Total packets              =      25273
Minimum delay              =      0.158879 s
Maximum delay              =      0.170885 s
Average delay              =      0.159968 s
Average jitter             =      0.000175 s
Delay standard deviation    =     0.000510 s
Bytes received             =      30327600
Average bitrate            =    8087.360270 Kbit/s
Average packet rate        =      842.433361 pkt/s
Packets dropped            =      0 (0.00 %)
Average loss-burst size =                0 pkt
Error lines                =            0
------------------------------------------------------------
```

Um dos pacotes gerados pelos comandos acima foi capturado pelo *Wireshark* e está representado na tabela 2 a seguir. Observa-se que o pacote representado na tabela 2 corresponde a um daqueles enviados pelos comandos acima, visto que as portas dos processos responsáveis pelo envio e recepção são os mesmos nos mesmos nos dois casos: a transmissão foi feita através da porta **33432** e a recepção, atraves de **8999**. Apesar das oritentações dadas pelo professor através do links (o tráfego capturado pelo Wireshark não corresponde completamente a aquele que é enviado pela rede), iremos assumir que o tráfego capturado não sofreu grandes modificações e, portanto, é confiável.

Tabela 2: *Log* gerado por Wireshark no ***receptor*** 20.0.1.190

```
  25677 472.059450000  30.0.1.190 20.0.1.190 UDP  Source port: 33432  Destination port: 8999

Frame 25677: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on interface 0
       Interface id: 0 (eth2)
       Encapsulation type: Ethernet (1)
       Arrival Time: Oct 15, 2015 15:24:31.352414000 BRT
       [Time shift for this packet: 0.000000000 seconds]
       Epoch Time: 1444933471.352414000 seconds
       [Time delta from previous captured frame: 0.003403000 seconds]
       [Time delta from previous displayed frame: 59.707418000 seconds]
       [Time since reference or first frame: 472.059450000 seconds]
       Frame Number: 25677
       Frame Length: 1242 bytes (9936 bits)
       Capture Length: 1242 bytes (9936 bits)
       [Frame is marked: False]
       [Frame is ignored: False]
       [Protocols in frame: eth:ethertype:ip:udp:data]
       [Coloring Rule Name: UDP]
       [Coloring Rule String: udp]
Ethernet II, Src: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2), Dst: CadmusCo_96:07:e0
(08:00:27:96:07:e0)
       Destination: CadmusCo_96:07:e0 (08:00:27:96:07:e0)
       Address: CadmusCo_96:07:e0 (08:00:27:96:07:e0)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
       Source: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2)
```

```
       Address: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
       Type: IP (0x0800)
Internet Protocol Version 4, Src: 30.0.1.190 (30.0.1.190), Dst: 20.0.1.190 (20.0.1.190)
       Version: 4
       Header Length: 20 bytes
       Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
       0000 00.. = Differentiated Services Codepoint: Default (0x00)
       .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
       Total Length: 1228
       Identification: 0xb63d (46653)
       Flags: 0x02 (Don't Fragment)
       0... .... = Reserved bit: Not set
       .1.. .... = Don't fragment: Set
       ..0. .... = More fragments: Not set
       Fragment offset: 0
       Time to live: 60
       Protocol: UDP (17)
       Header checksum: 0x4e68 [validation disabled]
       [Good: False]
       [Bad: False]
       Source: 30.0.1.190 (30.0.1.190)
       Destination: 20.0.1.190 (20.0.1.190)
User Datagram Protocol, Src Port: 33432 (33432), Dst Port: 8999 (8999)
       Source Port: 33432 (33432)
       Destination Port: 8999 (8999)
       Length: 1208
       Checksum: 0x08f4 [validation disabled]
       [Good Checksum: False]
       [Bad Checksum: False]
       [Stream index: 5]
Data (1200 bytes)
```

a. De acordo com a Tabela 1, foram trocados 25273 pacotes, sendo que a taxa de transmissão foi determinada sendo de 1000 pacotes por segundo. Logo, deveríamos ter tido 30000 pacotes e não 25273. Conclui-se que alguns pacotes foram perdidos. No caso do protocolo UDP, já que a quantidade de dados (1200 bytes) somado com o requerido para os cabeçalhos é inferior ao MTU, então cada datagrama equivale a um pacote. Além disso, é possível verificar que a *flag Don't fragment* do protocolo IP está setada. Portanto, os datagramas não serão divididos. Esta quantidade de pacotes enviados também é verificado via *Wireshark*: o primeiro pacote trocado entre os hosts tem numero *25677* e o último, *50955*. 50955 - 25677 = ***25278*** que é aproximadamente igual a 25273 (provavelmente alguma outra aplicação enviou estes 5 pacotes UDP de diferença).

b. De acordo com a tabela 2, os datagramas possuem um tamanho de 1208 bytes. Considerando que 8 bytes são usados para o cabeçalho (2 bytes para o destino, 2 bytes para a fonte, 2 bytes para o tamanho do datagrama e 2 bytes para *checksum*, que é utilizado para verificar a integridade do pacote), então o tamanho da carga é 1200 bytes.

c. O tamanho total do pacote capturado pelo Wireshark é de 1242 bytes, sendo que 1208 bytes pertencem à camada de transporte (UDP: 8 bytes de cabeçalho + 1200 bytes de dados), 20 bytes à camada de rede (IP - explicitado em negrito na tabela 2) e o restante (20 bytes) para a camada Ethernet.

d. O único campo que é alterado no cabeçalho dos datagramas UDP é o *checksum*. Os endereços de destino e fonte continuam os mesmos, assim como o tamanho dos pacotes (1208 bytes).

Exercício 2

Neste exercício, configura-se o *Wireshark* para capturar somente tráfego TCP e o transmissor para enviar pacotes TCP a uma taxa de 1000 pacotes por segundo, com dado de 1200 bytes e duração de transmissão de 30000 ms. O arquivo de *log* gerado no ***receptor*** pelo `ITGDec` está representado na tabela 3 abaixo:

Tabela 3: *Log* gerado por ITGDec no *receptor* 20.0.1.190 para tráfego TCP

```
------------------------------------------------------------
Flow number: 1
From 30.0.1.190:48941
To     20.0.1.190:8999
------------------------------------------------------------
Total time                =      30.000110 s
Total packets             =      24756
Minimum delay             =      0.169295 s
Maximum delay             =      0.177927 s
Average delay             =      0.171263 s
Average jitter            =      0.000664 s
Delay standard deviation   =     0.000659 s
Bytes received            =      29707200
Average bitrate           =    7921.890953 Kbit/s
Average packet rate       =      825.196974 pkt/s
Packets dropped           =            0 (0.00 %)
Average loss-burst size    =     0.000000 pkt
------------------------------------------------------------

------------------------------------------------------------
****************  TOTAL RESULTS   ******************
------------------------------------------------------------
Number of flows           =            1
Total time                =      30.000110 s
Total packets             =      24756
Minimum delay             =      0.169295 s
Maximum delay             =      0.177927 s
Average delay             =      0.171263 s
Average jitter            =      0.000664 s
Delay standard deviation   =     0.000659 s
Bytes received            =      29707200
Average bitrate           =    7921.890953 Kbit/s
Average packet rate       =      825.196974 pkt/s
Packets dropped           =            0 (0.00 %)
Average loss-burst size    =     0 pkt
Error lines               =            0
------------------------------------------------------------
```

Ao contrário do UDP, o protocolo TCP oferece uma série de recursos, como a detecção de erros por exemplo, para garantir a transmissão. O primeiro passo entre a troca de mensagens entre cliente e servidor é o estabelecimento da conexão. Essa tarefa é realizada em três passos:

- O cliente envia um pacote contendo **SYN** para o servidor com um número de sequência aleatório A;
- O servidor, ao receber uma requisição de conexão, envia uma mensagem de *acknowledge* **SYN-ACK.** O número *acknowledge* é modificado para A + 1 e o número de sequência escolhido pelo servidor é outro aleatório B.
- O cliente envia **ACK** indicando que a comunicação foi bem sucedida.

A figura 8 a seguir ilustra esta ação do protocolo, que é tambem chamada de *three-way handshake*.

```
17 23.461326000  30.0.1.190    20.0.1.190    TCP   74 48941→8999 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=953496
18 23.461352000  20.0.1.190    30.0.1.190    TCP   74 8999→48941 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TS
19 23.462616000  30.0.1.190    20.0.1.190    TCP   66 48941→8999 [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=953496 TSecr=1161287
```

Figura 8: Estabelecimento de conexão TCP entre cliente 30.0.1.190 e servidor 20.0.1.190

Uma vez terminada a troca de mensagens referentes aos dados, cliente e servidor começam o procedimento de fechamento de conexão, ilustrado pela figura 9 (considerar apenas pacotes sendo transmitidos entre portas 8999 e 48941).

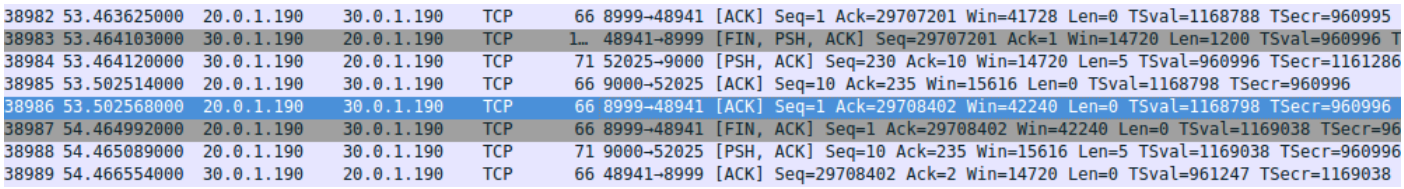


Figura 9: Fechamento de conexão TCP entre cliente 30.0.1.190 e servidor 20.0.1.190

A tabela 4 possui informações sobre um dos pacotes enviados pelo comando `ITGSend` capturados pelo *Wireshark*. Assim como no item anterior, os números das portas do pacote capturado corresponde a aquele transmitido pelo `ITGSend` (**8999** para o receptor e **48941** para o transmissor).

Tabela 4: *Log* gerado por Wireshark no ***receptor*** 20.0.1.190 para tráfego TCP

```
       57 23.489197000    30.0.1.190            20.0.1.190          TCP    1514   48941→8999
[ACK] Seq=22801 Ack=1 Win=14720 Len=1448 TSval=953503 TSecr=1161293

Frame 57: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
       Interface id: 0 (eth2)
       Encapsulation type: Ethernet (1)
       Arrival Time: Oct 15, 2015 15:36:55.177261000 BRT
       [Time shift for this packet: 0.000000000 seconds]
       Epoch Time: 1444934215.177261000 seconds
       [Time delta from previous captured frame: 0.000543000 seconds]
       [Time delta from previous displayed frame: 0.000543000 seconds]
       [Time since reference or first frame: 23.489197000 seconds]
       Frame Number: 57
       Frame Length: 1514 bytes (12112 bits)
       Capture Length: 1514 bytes (12112 bits)
       [Frame is marked: False]
       [Frame is ignored: False]
       [Protocols in frame: eth:ethertype:ip:tcp:data]
       [Coloring Rule Name: TCP]
       [Coloring Rule String: tcp]
Ethernet II, Src: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2), Dst: CadmusCo_96:07:e0
(08:00:27:96:07:e0)
       Destination: CadmusCo_96:07:e0 (08:00:27:96:07:e0)
       Address: CadmusCo_96:07:e0 (08:00:27:96:07:e0)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
       Source: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2)
       Address: CadmusCo_b9:11:a2 (08:00:27:b9:11:a2)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
       Type: IP (0x0800)
Internet Protocol Version 4, Src: 30.0.1.190 (30.0.1.190), Dst: 20.0.1.190 (20.0.1.190)
       Version: 4
       Header Length: 20 bytes
       Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
       0000 00.. = Differentiated Services Codepoint: Default (0x00)
       .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
       Total Length: 1500
       Identification: 0x5d1c (23836)
       Flags: 0x02 (Don't Fragment)
       0... .... = Reserved bit: Not set
       .1.. .... = Don't fragment: Set
       ..0. .... = More fragments: Not set
       Fragment offset: 0
       Time to live: 60
       Protocol: TCP (6)
       Header checksum: 0xa684 [validation disabled]
```

```
       [Good: False]
       [Bad: False]
       Source: 30.0.1.190 (30.0.1.190)
       Destination: 20.0.1.190 (20.0.1.190)
Transmission Control Protocol, Src Port: 48941 (48941), Dst Port: 8999 (8999), Seq: 22801,
Ack: 1, Len: 1448
       Source Port: 48941 (48941)
       Destination Port: 8999 (8999)
       [Stream index: 1]
       [TCP Segment Len: 1448]
       Sequence number: 22801     (relative sequence number)
       [Next sequence number: 24249     (relative sequence number)]
       Acknowledgment number: 1   (relative ack number)
       Header Length: 32 bytes
       .... 0000 0001 0000 = Flags: 0x010 (ACK)
       000. .... .... = Reserved: Not set
       ...0 .... .... = Nonce: Not set
       .... 0... .... = Congestion Window Reduced (CWR): Not set
       .... .0.. .... = ECN-Echo: Not set
       .... ..0. .... = Urgent: Not set
       .... ...1 .... = Acknowledgment: Set
       .... .... 0... = Push: Not set
       .... .... .0.. = Reset: Not set
       .... .... ..0. = Syn: Not set
       .... .... ...0 = Fin: Not set
       Window size value: 115
       [Calculated window size: 14720]
       [Window size scaling factor: 128]
       Checksum: 0xea7e [validation disabled]
       [Good Checksum: False]
       [Bad Checksum: False]
       Urgent pointer: 0
       Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
       No-Operation (NOP)
              Type: 1
              0... .... = Copy on fragmentation: No
              .00. .... = Class: Control (0)
              ...0 0001 = Number: No-Operation (NOP) (1)
       No-Operation (NOP)
              Type: 1
              0... .... = Copy on fragmentation: No
              .00. .... = Class: Control (0)
              ...0 0001 = Number: No-Operation (NOP) (1)
       Timestamps: TSval 953503, TSecr 1161293
              Kind: Time Stamp Option (8)
              Length: 10
              Timestamp value: 953503
              Timestamp echo reply: 1161293
       [SEQ/ACK analysis]
       [iRTT: 0.001290000 seconds]
       [Bytes in flight: 1448]
Data (1448 bytes)
[...]
       Data: 0000000100000014000105c700001846000004b0acbf659f...
       [Length: 1448]
```

a. De acordo com a captura de tráfego pelo *Wireshark,* o primeiro pacote de estabelecimento de conexão possui um número de identificação *17* e o último pacote, utilizado para finalizar a respectiva conexão, *38987*. Logo, foram trocadas 38987 - 17 = ***38970*** mensagens entre os *hosts 20.0.1.190* e *30.0.1.190*. Em comparação com o mesmo item do exercício anterior, verifica-se que o protocolo TCP utiliza muito mais pacotes que o UDP. Esse comportamento era perfeitamente esperado, uma vez que a comunicação entre os *hosts* no caso TCP é completamente síncrono, isto é, cada *host* deve conhecer se o outro recebeu corretamente os dados transmitidos.

b. Cada cabeçalho TCP é composto pelas portas de origem e destino (16 bits para cada), um número de sequência (32 bits), um número de *acknowledgment* (32 bits), um campo de 4 bits chamado *data offset,* cujo propósito é especificar o tamanho do cabeçalho TCP em palavras de 32 bits, 3 bits

reservados, 9 bits para as *flags*, 16 bits para o campo *window size*, 16 bits para *checksum*, 16 bits para *urgent pointer* e um número variavel de 0 a 320 bits para o campo *options*. O cabeçalho, portanto, tem um valor mínimo de 20 bytes e um valor máximo de 60 bytes. No nosso caso, cada pacote transmitido possui 66 bytes, sendo compostos pelos cabeçalhos TCP (32 bytes), Ethernet (14 bytes) e IP (20 bytes).

c. Para a aplicação TCP foram transmitidos, somando-se cabeçalhos e dados, aproximadamente *32.280.106* bytes. Esta quantia foi dada pelo *Summary* do *Wireshark* após definirmos o filtro *tcp && tcp.port == 48941* (48941 é a porta do processo que está transmitindo). . Para o UDP, obtém-se, com o filtro *udp && udp.port == 33432* (33432 é porta do processo que está transmitindo os dados), *31.389.066* bytes. No caso TCP, foram enviados *891.040* bytes a mais que o UDP relativos às garantias de transmissão implementadas por este protocolo. Para efeito de comparação, esta quantia corresponde a aproximadamente 3% do total dos dados transmitidos no caso UDP. Para grandes transmissões, 3% pode representar um tráfego considerável pela rede.

Exercício 3

Nesta parte do roteiro, a taxa de transmissão de datagramas UDP foi sendo aumentada progressivamente esperando que houvesse uma taxa de perda de datagramas de pelo menos 5%. Assim, foi observado que a uma taxa de 21000, houve cerca de 10% de perda. É curioso observar que aumentando um pouco mais esta taxa, para 25000 por exemplo, o percentual de perda cresce abruptamente para mais de 36%, evidenciando que após certo valor a transmissão por UDP torna-se inviável.

Em relação ao envio de dados por meio do protocolo de transporte TCP, pode-se aumentar a taxa de transmissão o quanto se queira que não haverá perda de pacotes, visto que diferentemente do protocolo UDP, a transmissão é dita confiável, ou seja, a troca de mensagens é orientada à conexão. Isto quer dizer que um circuito virtual é criado em cima da topologia de rede fazendo com que a troca de mensagens ponto a ponto seja entregue de forma garantida e respeitando a ordem de envio, mesmo que o protocolo de rede não seja orientado à conexão.

É possível observar pela tabela 6 que mesmo com um número muito maior de pacotes enviados, uma taxa de transmissão de 30000, não houve perda alguma de dados

Tabela 5: *Log* gerado por ITGDec no *receptor* 20.0.1.190 para tráfego UDP

```
------------------------------------------------------------
Flow number: 1
From 30.0.1.190:53374
To    20.0.1.190:8999
------------------------------------------------------------
Total time               =     30.107805 s
Total packets            =        293051
Minimum delay            =      0.189495 s
Maximum delay            =      0.455214 s
Average delay            =      0.294890 s
Average jitter           =      0.000067 s
Delay standard deviation =      0.012873 s
Bytes received           =     351661200
Average bitrate          =  93440.541414 Kbit/s
Average packet rate      =   9733.389731 pkt/s
Packets dropped          =         34620 (10.57 %)
Average loss-burst size  =      1.350708 pkt
------------------------------------------------------------


------------------------------------------------------------
```

```
****************  TOTAL RESULTS   ******************
------------------------------------------------------------
Number of flows          =             1
Total time               =     30.107805 s
Total packets            =        293051
Minimum delay            =      0.189495 s
Maximum delay            =      0.455214 s
Average delay            =      0.294890 s
Average jitter           =      0.000067 s
Delay standard deviation =      0.012873 s
Bytes received           =     351661200
Average bitrate          =  93440.541414 Kbit/s
Average packet rate      =   9733.389731 pkt/s
Packets dropped          =         34620 (10.57 %)
Average loss-burst size  =      1.350708 pkt
Error lines              =             0
------------------------------------------------------------
```

Tabela 6: *Log* gerado por ITGDec no *receptor* 20.0.1.190 para tráfego TCP.

```
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
------------------------------------------------------------
Flow number: 1
From 30.0.1.190:48947
To    20.0.1.190:8999
------------------------------------------------------------
Total time               =     30.005067 s
Total packets            =        292274
Minimum delay            =      0.176016 s
Maximum delay            =      0.224644 s
Average delay            =      0.180720 s
Average jitter           =      0.000110 s
Delay standard deviation =      0.002407 s
Bytes received           =     350728800
Average bitrate          =  93511.885842 Kbit/s
Average packet rate      =   9740.821442 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =      0.000000 pkt
------------------------------------------------------------


------------------------------------------------------------
****************  TOTAL RESULTS   ******************
------------------------------------------------------------
Number of flows          =             1
Total time               =     30.005067 s
Total packets            =        292274
Minimum delay            =      0.176016 s
Maximum delay            =      0.224644 s
Average delay            =      0.180720 s
Average jitter           =      0.000110 s
Delay standard deviation =      0.002407 s
Bytes received           =     350728800
Average bitrate          =  93511.885842 Kbit/s
Average packet rate      =   9740.821442 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
------------------------------------------------------------
```

Exercício 4

As tabelas 7 e 8 contém, respectivamente, os resultados capturados no **receptor** pelo *Wireshark* para TCP e UDP para taxa de transmissão igual a 5000 pacotes por segundo e tamanho dos pacotes, 4000 bytes. É importante destacar as recomendações enviadas pelo professor que dizem respeito ao tamanho dos pacotes mostrados por este *software*. O problema reside no fato de que os pacotes são capturados pelo Wireshark após terem sido reunidos e, portanto, é possível visualizarmos pacotes com tamanho superior ao MTU.

Entretanto, as flags utilizadas, tanto pelo protocolo IP tanto como pelos protocolos TCP ou UDP, não são alteradas. Assim, é possível verificar se estes mesmos pacotes, já reunidos, foram fragmentados ou não na sua transmissão.

Tabela 7: *Log* gerado por Wireshark no ***receptor*** 20.0.1.190 para tráfego TCP para tamanho dos pacotes 4000 bytes e taxa de transmissão 5000 pacotes/seg.

```
[....]
Internet Protocol Version 4, Src: 30.0.1.190 (30.0.1.190), Dst: 20.0.1.190 (20.0.1.190)
       Version: 4
       Header length: 20 bytes
       Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
       0000 00.. = Differentiated Services Codepoint: Default (0x00)
       .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
       Total Length: 1500
       Identification: 0x9613 (38419)
       Flags: 0x02 (Don't Fragment)
       0... .... = Reserved bit: Not set
       .1.. .... = Don't fragment: Set
       ..0. .... = More fragments: Not set
       Fragment offset: 0
       Time to live: 60
       Protocol: TCP (6)
       Header checksum: 0x6d8d [correct]
       [Good: True]
       [Bad: False]
       Source: 30.0.1.190 (30.0.1.190)
       Destination: 20.0.1.190 (20.0.1.190)
Transmission Control Protocol, Src Port: 56647 (56647), Dst Port: bctp (8999), Seq: 100001,
Ack: 1, Len: 1448
       Source port: 56647 (56647)
       Destination port: bctp (8999)
       [Stream index: 1]
       Sequence number: 100001    (relative sequence number)
       [Next sequence number: 101449    (relative sequence number)]
       Acknowledgement number: 1  (relative ack number)
       Header length: 32 bytes
       Flags: 0x010 (ACK)
       000. .... .... = Reserved: Not set
       ...0 .... .... = Nonce: Not set
       .... 0... .... = Congestion Window Reduced (CWR): Not set
       .... .0.. .... = ECN-Echo: Not set
       .... ..0. .... = Urgent: Not set
       .... ...1 .... = Acknowledgement: Set
       .... .... 0... = Push: Not set
       .... .... .0.. = Reset: Not set
       .... .... ..0. = Syn: Not set
       .... .... ...0 = Fin: Not set
       Window size value: 115
       [Calculated window size: 14720]
       [Window size scaling factor: 128]
       Checksum: 0x78f4 [validation disabled]
       [Good Checksum: False]
       [Bad Checksum: False]
       Options: (12 bytes)
       No-Operation (NOP)
       No-Operation (NOP)
       Timestamps: TSval 1520427, TSecr 1982759
              Kind: Timestamp (8)
              Length: 10
              Timestamp value: 1520427
              Timestamp echo reply: 1982759
       [SEQ/ACK analysis]
       [Bytes in flight: 1448]
Data (1448 bytes)
[....]
```

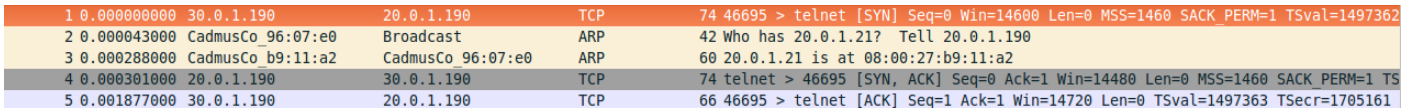
Tabela 8: *Log* gerado por Wireshark no ***receptor*** 20.0.1.190 para tráfego UDP para tamanho dos pacotes 4000 bytes e taxa de transmissão 5000 pacotes/seg.

```
[...]
Internet Protocol Version 4, Src: 30.0.1.190 (30.0.1.190), Dst: 20.0.1.190 (20.0.1.190)
       Version: 4
       Header length: 20 bytes
       Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
       0000 00.. = Differentiated Services Codepoint: Default (0x00)
       .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
       Total Length: 1068
       Identification: 0x295f (10591)
       Flags: 0x00
       0... .... = Reserved bit: Not set
       .0.. .... = Don't fragment: Not set
       ..0. .... = More fragments: Not set
       Fragment offset: 2960
       Time to live: 60
       Protocol: UDP (17)
       Header checksum: 0x1a75 [correct]
       [Good: True]
       [Bad: False]
       Source: 30.0.1.190 (30.0.1.190)
       Destination: 20.0.1.190 (20.0.1.190)
       [3 IPv4 Fragments (4008 bytes): #19(1480), #20(1480), #21(1048)]
       [Frame: 19, payload: 0-1479 (1480 bytes)]
       [Frame: 20, payload: 1480-2959 (1480 bytes)]
       [Frame: 21, payload: 2960-4007 (1048 bytes)]
       [Fragment count: 3]
       [Reassembled IPv4 length: 4008]
User Datagram Protocol, Src Port: 55268 (55268), Dst Port: bctp (8999)
       Source port: 55268 (55268)
       Destination port: bctp (8999)
       Length: 4008
       Checksum: 0x438a [validation disabled]
       [Good Checksum: False]
       [Bad Checksum: False]
Data (4000 bytes)
[...]
```

Verifica-se que na tabela 7, *flag Don't Segment* do protocolo IP está setada. Portanto, os segmentos TCP, quando chegam na camada de rede, não podem ser fragmentados. Isto indica que o próprio protocolo da camada de transporte calcula o melhor tamanho de cada segmento para que não haja nenhuma fragmentação. É justamente essa a função desempenhada pelo parâmetro MSS (*Maximum segment size*). Observa-se ainda que, para evitar a fragmentação na camada IP, o valor de *MSS* deve ser igual ao tamanho do maior *datagrama* IP menos os tamanhos dos cabeçalhos *IP* e *TCP*. Conclui-se, portanto, que pequenos valores de MSS reduzirão a fragmentação IP, mas causarão uma sobrecarga na rede, uma vez haverá muitos mais pacotes a serem enviados.

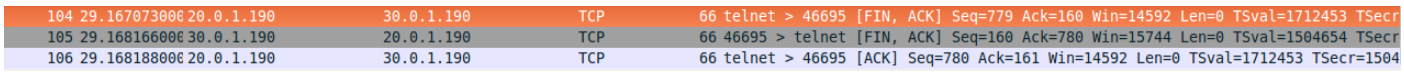
O protocolo, por outro lado, não se preocupa em dividir os dados em menores unidades e, portanto, não utiliza parâmetros como o *MSS*. Sendo assim, os dados de 4000 bytes são transmitidos diretamente à camada de rede, que, por sua vez, deve detectar que os pacotes a enviar são maiores que o MTU e, assim, devem ser fragmentados. A tabela 8 ilustra esse caso: a *flag Don't Segment* não está setada, indicando que os pacotes podem ser divididos, e os 3 fragmentos estão identificados (19, 20 e 21), assim como o tamanho que cada um possui.

Exercício 5



1	0.000000000	30.0.1.190	20.0.1.190	TCP	74 46695 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1497362
2	0.000043000	CadmusCo_96:07:e0	Broadcast	ARP	42 Who has 20.0.1.21? Tell 20.0.1.190
3	0.000288000	CadmusCo_b9:11:a2	CadmusCo_96:07:e0	ARP	60 20.0.1.21 is at 08:00:27:b9:11:a2
4	0.000301000	20.0.1.190	30.0.1.190	TCP	74 telnet > 46695 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TS
5	0.001877000	30.0.1.190	20.0.1.190	TCP	66 46695 > telnet [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=1497363 TSecr=1705161

Figura 10: abertura de uma conexão TCP por meio da aplicação Telnet



104	29.167073000	20.0.1.190	30.0.1.190	TCP	66 telnet > 46695 [FIN, ACK] Seq=779 Ack=160 Win=14592 Len=0 TSval=1712453 TSecr
105	29.168166000	30.0.1.190	20.0.1.190	TCP	66 46695 > telnet [FIN, ACK] Seq=160 Ack=780 Win=15744 Len=0 TSval=1504654 TSecr
106	29.168188000	20.0.1.190	30.0.1.190	TCP	66 telnet > 46695 [ACK] Seq=780 Ack=161 Win=14592 Len=0 TSval=1712453 TSecr=1504

Figura 11: fechamento de uma conexão TCP por meio da aplicação Telnet

a. Para verificar o funcionamento do *three-way handshake* do protocolo TCP, foi utilizado a aplicação Telnet, que utiliza deste protocolo de transporte, com o Wireshark para observar as mensagens de abertura e fechamento envolvidas. Observa-se na figura 10 e 11 as mensagens trocadas entre cliente e servidor na abertura e no fechamento da conexão respectivamente. Lembrando que o *host* com *IP* 30.0.1.190 é o servidor e o *host* com *IP* 20.0.1.190 é o cliente.

   Em relação ao *handshake* abertura, o servidor envia um pacote com a flag SYN em seu cabeçalho TCP, o cliente responde com um pacote com as flags SYN e ACK e em seguida o servisor responde com uma mensagem de ACK. Essa troca é vista nas linhas 1, 4 e 5 da figura 10.

b. Como pode ser visto na figura 10, os valores das sequencias iniciais do servidor e do cliente são ambas zero, apenas quando uma mensagem ACK é recebida que incrementa-se o valor de SEQ.

c. O número de sequencia do primeiro pacote de dados enviado pelo cliente é SEQ=1, visto que após o handshake de abertura da conexão o cliente recebe um ACK=1. Por isso soma-se ‘ ao valor de SEQ anterior que é zero.

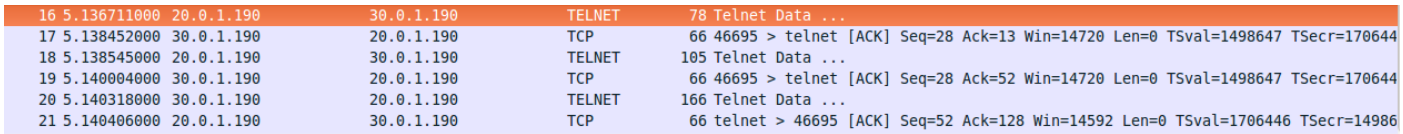
d. As janelas indicadas pelo cliente e pelo servidor são, respectivamente, 14720 e 14592 bytes.

e. O MSS é de 1460 bytes, nota-se que o tamanho do MTU é de 1500 bytes dado que os 40 bytes faltantes são do cabeçalho do TCP.

f. Primeiramente, como foi explicado no item c., o número de sequencia do primeiro pacote de dados é 1 e pela figura 13 o próximo n´mero de sequencia é 13 e número de Ack é 28. Assim, espera-se que o pacote que o servidor enviar para o cliente contenha uma número de sequencia de 28 e o Ack de 13, pois esse é o método para manter o canal de transmissão orientado a conexão, pois assim não há perda de pacotes.

   Como é possível observar por meio da figura 14, os valores esperados de número de sequencia e Ack enviados do servidor para o cliente são 28 e 13 respectivamente.

g. O *handshake* de fechamento acontece analogamente ao de abertura como mostra a figura 11, ou seja, o servidor envia um pacote de fechamento com as flags FIN e ACK em seu cabeçalho, o cliente responde com as flags FIN e ACK e em seguida o servidor envia finalmente a mensagem de ACK.



16	5.136711000	20.0.1.190	30.0.1.190	TELNET	78 Telnet Data ...
17	5.138452000	30.0.1.190	20.0.1.190	TCP	66 46695 > telnet [ACK] Seq=28 Ack=13 Win=14720 Len=0 TSval=1498647 TSecr=170644
18	5.138545000	20.0.1.190	30.0.1.190	TELNET	105 Telnet Data ...
19	5.140004000	30.0.1.190	20.0.1.190	TCP	66 46695 > telnet [ACK] Seq=28 Ack=52 Win=14720 Len=0 TSval=1498647 TSecr=170644
20	5.140318000	30.0.1.190	20.0.1.190	TELNET	166 Telnet Data ...
21	5.140406000	20.0.1.190	30.0.1.190	TCP	66 telnet > 46695 [ACK] Seq=52 Ack=128 Win=14592 Len=0 TSval=1706446 TSecr=14986

Figura 12: mensagens trocadas por meio do Telnet entre cliente e servidor

```
▼Transmission Control Protocol, Src Port: telnet (23), Dst Port: 46695 (46695), Seq: 1, Ack: 28, Len: 12
  Source port: telnet (23)
  Destination port: 46695 (46695)
  [Stream index: 0]
  Sequence number: 1    (relative sequence number)
  [Next sequence number: 13    (relative sequence number)]
  Acknowledgment number: 28    (relative ack number)
```

Figura 13: expansão da mensagem da linha 16 da figura 12

```
▼Transmission Control Protocol, Src Port: 46695 (46695), Dst Port: telnet (23), Seq: 28, Ack: 13, Len: 0
  Source port: 46695 (46695)
  Destination port: telnet (23)
  [Stream index: 0]
  Sequence number: 28    (relative sequence number)
  Acknowledgment number: 13    (relative ack number)
```

Figura 14: expansão da mensagem da linha 17 da figura 12

Exercício 6

a. Quando aumentamos o tráfego UDP na rede, aumentamos também as chances de algum segmento TCP se perder. Sendo assim, como o protocolo TCP é resistente à falhas, os pacotes que foram perdidos devem ser retransmitidos. O gráfico representado na figura 15 ilustra este fato. Considerando que números de sequências iguais representam pacotes iguais, se número de sêquencia permanece constante, então uma retransmissão foi requisitada. Detecta-se, nesta figura, que dois pacotes foram perdidos e, portanto, foram retransmitidos pelo cliente TCP.

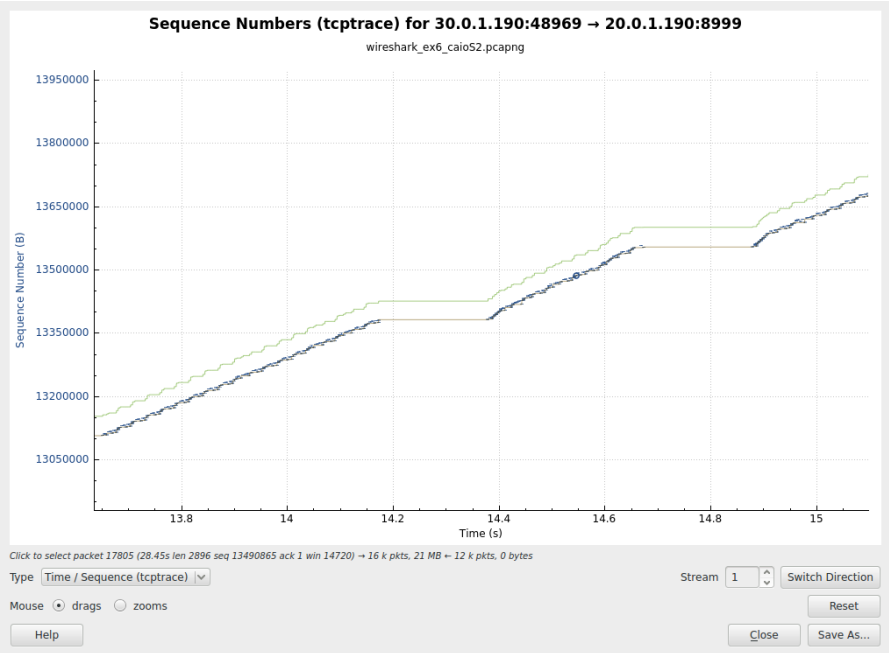


Figura 15: retransmissões dos segmentos TCP

b. A figura 16 ilustra a evolução da taxa de transmissão ao longo das transmissões do pacotes. Observa-se que a taxa de transmissão diminui com o aumento do tráfego na rede, isto é, com a ativação do tráfego UDP entre os outros *hosts*. Este fato é explicado pelo aumento de retransmissões devidas às perdas de pacotes. Se um mesmo pacote necessita ser enviado mais de uma vez, a taxa de transmissao é evidentemente menor.

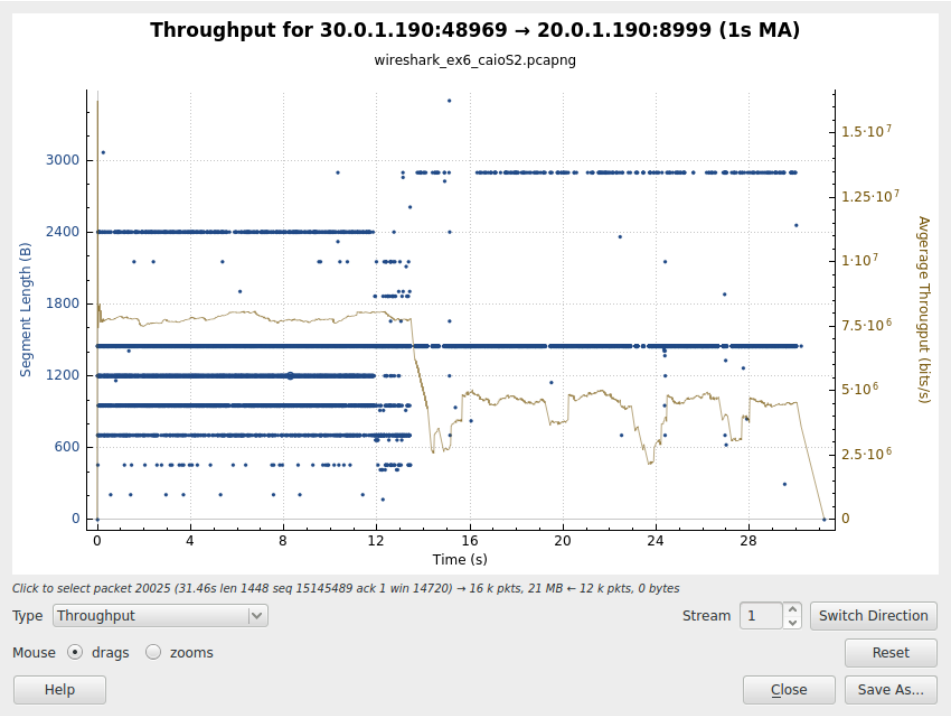


Figura 16: retransmissões dos segmentos TCP