EA080: Laboratório 02

Vinícius Esperança Mantovani,
247395.

Introdução)

Neste experimento, visou-se aprofundar os conhecimentos na camada de transporte, com enfoque para os protocolos TCP e UDP. Para tanto, foi criada uma rede com o *mininet*, de topologia demonstrada na *Figura 1*, usando-se o *script python* "*tcp.py*". Nesse sentido, como modo de se conhecer mais a respeito desses protocolos, foram realizados testes e análises da rede usada.

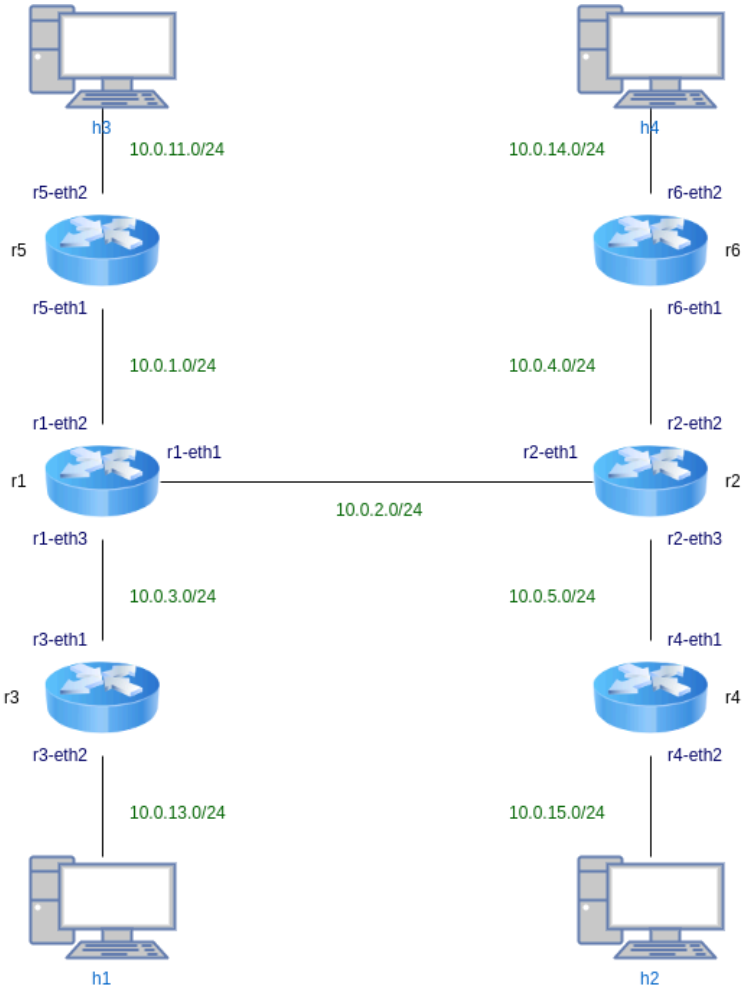


Figura 1: Topologia da rede usada.

Exercício 1)

Inicialmente, utilizando-se o comando "*nodes*", foi possível verificar a existência de todos os nós da rede, conforme *Figura 2* abaixo.

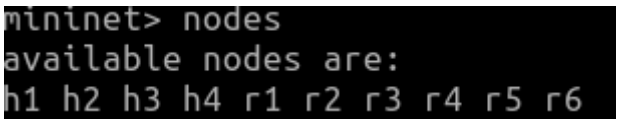


Figura 2: Resultado comando "nodes" no terminal mininet.

Seguindo, ainda em análise da corretude da rede com relação ao que se exibe na imagem, pode-se notar na *Figura 3* abaixo, que na execução do comando "*links*", tem-se as conexões corretas entre os nós da rede:

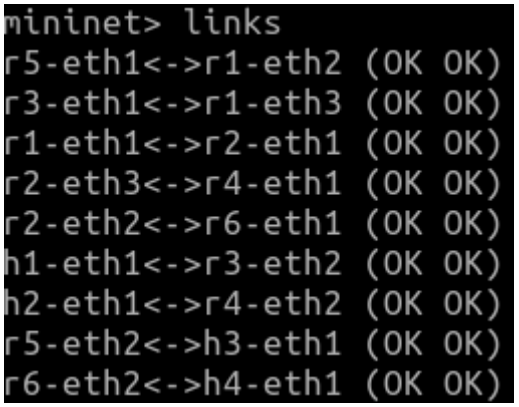


Figura 3: Resultado comando "links" no terminal mininet.

Em sequência, foram abertos os terminais dos *hosts h1* e *h4* com o comando "*xterm*" e neles, para evitar que os pacotes fossem fragmentados na placa de rede e não pelo *Kernel*, foram executados respectivamente os comandos "*ethtool -K h1-eth1 tso off*" e "*ethtool -K h4-eth1 tso off*", em cada um dos terminais. Deste modo, todos os pacotes capturados por meio do *Wireshark* apresentam seus tamanhos corretos, já segmentados. Isso é importante, pois o *Wireshark* captura os pacotes antes de chegarem à placa de rede, de modo que, caso fossem segmentados apenas na placa de rede, o *Wireshark* capturaria menos pacotes, com tamanhos maiores, ainda não segmentados.

Feitas as configurações citadas, foi aberto um receptor *netcat* no *host h1 (*com o comando "*nc -l 4444 -u > data1.txt*") e foi enviado um pacote UDP por *h4* para *h1* (com o comando "*nc -w 3 10.0.13.3 4444 -u < data1.txt*"), tendo sido tal pacote criado no próprio *h4* com o comando "*truncate -s 1200 data1.txt*". Durante tal processo, foi feita uma captura com o *Wireshark.* Vale destacar que nas execuções de *netcat*, foram usadas, no *host 1*, as *flags* "-l" e "-u", responsáveis respectivamente por criar um receptor e definir o uso de UDP ao invés de TCP. Além disso, no *host 4* foram usadas "-u" e "-w", sendo esta nova responsável por estabelecer o tempo de "timeout". Desse modo, obteve-se a captura exibida na *Figura 4*.

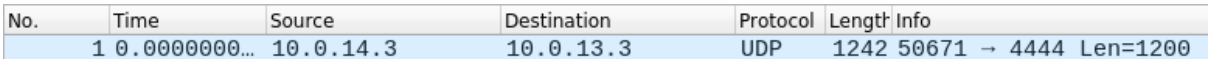
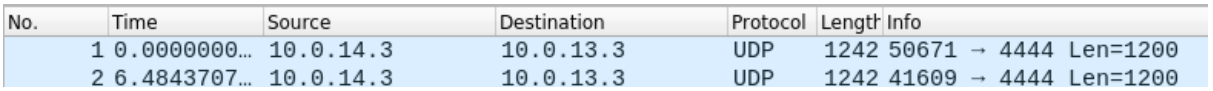


Figura 4: Resultado captura Wireshark pacote UDP recebido por h1 com pacote de 1200 Bytes.

Da captura de pacotes, podemos notar que é enviado um único pacote para o datagrama UDP de *1200 Bytes.* Porém, é possível afirmar que, para datagramas maiores que o limite do IPv4, o número de pacotes seria determinado por tal protocolo, segmentando-se o datagrama UDP em vários pacotes IPv4, nesse caso. Além disso, da captura, é possível notar que o tamanho do pacote UDP recebido tem *1242 Bytes*, de maneira a destoar do total de dados da aplicação *1200 Bytes*. Isso de dá,

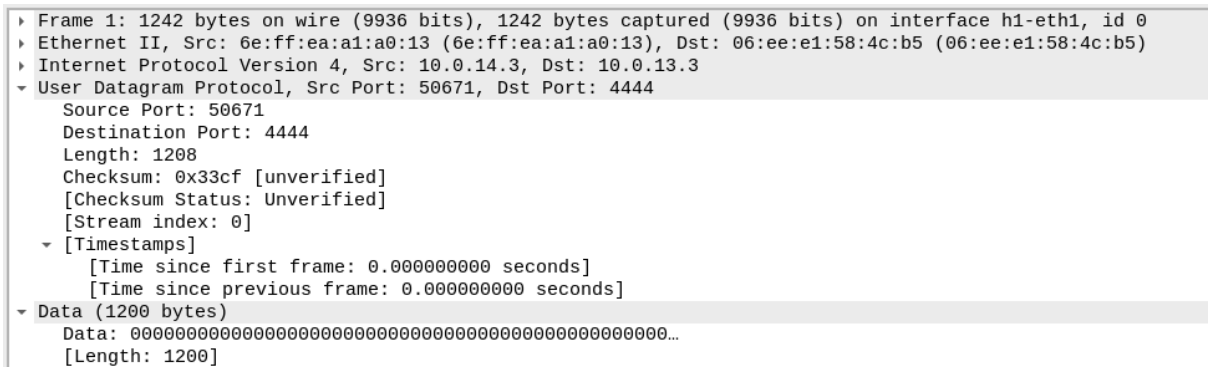
por conta dos cabeçalhos das camadas de enlace (*14 Bytes*), rede (*20 Bytes*) e transporte (*8 Bytes*), que somam os *42 Bytes* de diferença entre os dados de aplicação e o total do pacote.

Finalmente, terminando este exercício, foi feita uma segunda captura, de maneira a se ter o que se apresenta na *Figura 5*, na qual já se pode ver a diferença nos "*ports*" da fonte (emissor). Já em maior profundidade, analisando as *Figuras 6* e *7* que apresentam o detalhamento dos datagramas UDPs, nota-se que essa é a única diferença entre eles, uma vez que nem mesmo o *checksum* é diferente, uma vez que ele depende apenas do *payload* do pacote e este é o mesmo para ambos os pacotes.



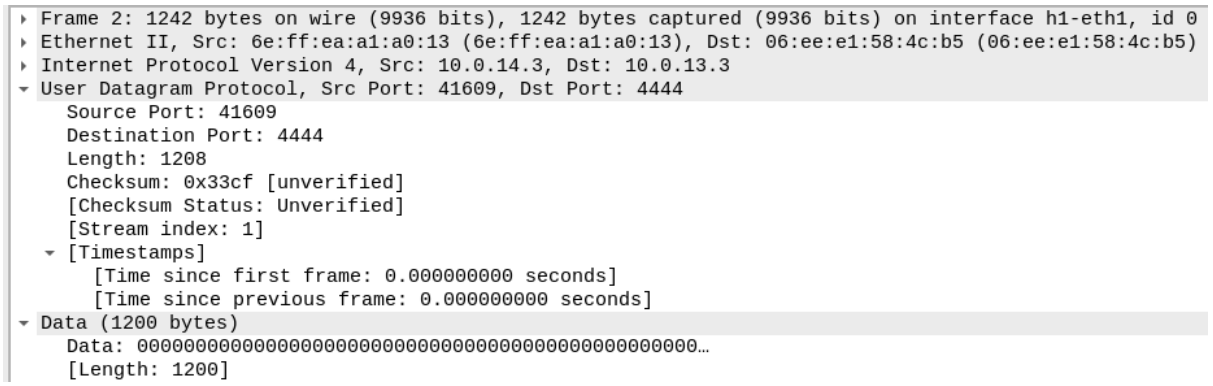
No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	10.0.14.3	10.0.13.3	UDP	1242	50671 → 4444 Len=1200
2	6.4843707…	10.0.14.3	10.0.13.3	UDP	1242	41609 → 4444 Len=1200

Figura 5: Resultado captura Wireshark primeiro e segundo pacote UDP recebido por h1.



```
▸ Frame 1: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on interface h1-eth1, id 0
▸ Ethernet II, Src: 6e:ff:ea:a1:a0:13 (6e:ff:ea:a1:a0:13), Dst: 06:ee:e1:58:4c:b5 (06:ee:e1:58:4c:b5)
▸ Internet Protocol Version 4, Src: 10.0.14.3, Dst: 10.0.13.3
▾ User Datagram Protocol, Src Port: 50671, Dst Port: 4444
    Source Port: 50671
    Destination Port: 4444
    Length: 1208
    Checksum: 0x33cf [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  ▾ [Timestamps]
      [Time since first frame: 0.000000000 seconds]
      [Time since previous frame: 0.000000000 seconds]
▾ Data (1200 bytes)
    Data: 000000000000000000000000000000000000000000000000…
    [Length: 1200]
```

Figura 6: Detalhamento primeiro diagrama UDP.

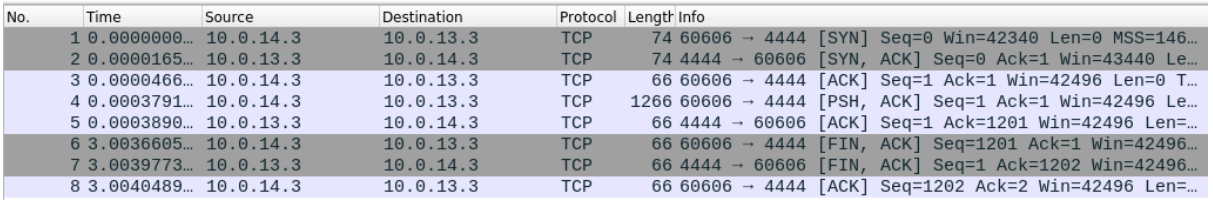


```
▸ Frame 2: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on interface h1-eth1, id 0
▸ Ethernet II, Src: 6e:ff:ea:a1:a0:13 (6e:ff:ea:a1:a0:13), Dst: 06:ee:e1:58:4c:b5 (06:ee:e1:58:4c:b5)
▸ Internet Protocol Version 4, Src: 10.0.14.3, Dst: 10.0.13.3
▾ User Datagram Protocol, Src Port: 41609, Dst Port: 4444
    Source Port: 41609
    Destination Port: 4444
    Length: 1208
    Checksum: 0x33cf [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
  ▾ [Timestamps]
      [Time since first frame: 0.000000000 seconds]
      [Time since previous frame: 0.000000000 seconds]
▾ Data (1200 bytes)
    Data: 000000000000000000000000000000000000000000000000…
    [Length: 1200]
```

Figura 7: Detalhamento segundo diagrama UDP.

Exercício 2)

Nesta atividade, foi feita a captura de pacotes para um procedimento de envio de pacotes de *h4* para *h1* novamente, mas agora usando o protocolo TCP. Desse modo, fez-se possível analisar os pacotes recebidos por *h1* e enviados por ele. Segue *Figura 8*, que contém tais pacotes:



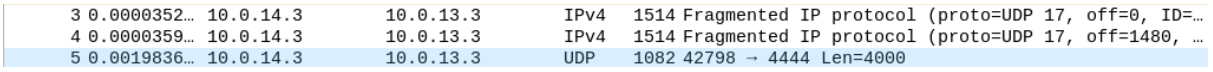
No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	10.0.14.3	10.0.13.3	TCP	74	60606 → 4444 [SYN] Seq=0 Win=42340 Len=0 MSS=146…
2	0.0000165…	10.0.13.3	10.0.14.3	TCP	74	4444 → 60606 [SYN, ACK] Seq=0 Ack=1 Win=43440 Le…
3	0.0000466…	10.0.14.3	10.0.13.3	TCP	66	60606 → 4444 [ACK] Seq=1 Ack=1 Win=42496 Len=0 T…
4	0.0003791…	10.0.14.3	10.0.13.3	TCP	1266	60606 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=42496 Le…
5	0.0003890…	10.0.13.3	10.0.14.3	TCP	66	4444 → 60606 [ACK] Seq=1 Ack=1201 Win=42496 Len=…
6	3.0036605…	10.0.14.3	10.0.13.3	TCP	66	60606 → 4444 [FIN, ACK] Seq=1201 Ack=1 Win=42496…
7	3.0039773…	10.0.13.3	10.0.14.3	TCP	66	4444 → 60606 [FIN, ACK] Seq=1 Ack=1202 Win=42496…
8	3.0040489…	10.0.14.3	10.0.13.3	TCP	66	60606 → 4444 [ACK] Seq=1202 Ack=2 Win=42496 Len=…

Figura 8: Pacotes TCP capturados por Wireshark em h1 com pacote de 1200 Bytes.

Da figura acima, pode-se observar que foram trocados um total de 8 pacotes TCP neste procedimento. Além disso, os segmentos TCP têm um tamanho de *32 Bytes*, com exceção dos pacotes de sincronização *SYN*, que têm *40 Bytes*. Por fim, calculando o número total de *Bytes* transmitidos de *h1* para *h4* tem-se *206 Bytes* e, somando o fluxo contrário, tem-se *1538 Bytes*, sendo destes últimos, somente *1200* Bytes de *payload*, o que provê uma razão (*Bytes* de dados) / (*Bytes totais da conexão*) de 1200/1744 ≅ 0,688 = 68,8%.

Exercício 3)

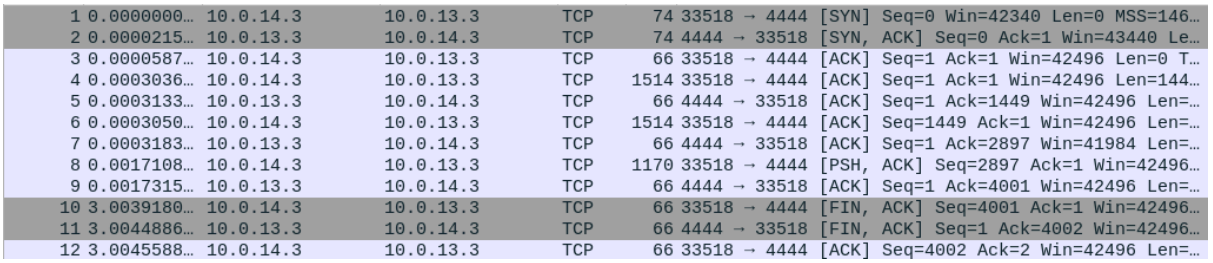
Continuando-se o experimento, neste exercício, foram repetidos os procedimentos dos dois exercícios anteriores, porém, com o pacote enviado contendo *4000 Bytes* de dados. Dessa maneira, obteve-se, inicialmente, para a experiência com o UDP o que se nota na *Figura 9* a seguir. Nesta figura, pode-se observar que o datagrama UDP foi fragmentado em camada de rede em três datagramas *IPv4* com *1514 Bytes* os dois primeiros e *1082* o último, que já é identificado no wireshark como datagrama UDP, pois já este já foi, então, remontado com a junção dos três datagramas *IPv4*.



3	0.0000352…	10.0.14.3	10.0.13.3	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=…
4	0.0000359…	10.0.14.3	10.0.13.3	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, …
5	0.0019836…	10.0.14.3	10.0.13.3	UDP	1082	42798 → 4444 Len=4000

Figura 9: Pacotes UDP capturados por Wireshark em h1 com pacote de 4000 Bytes.

Ademais, à critérios de análise e comparação, foi feito o mesmo com o protocolo TCP, conforme a *Figura 10* abaixo. Em tal figura, percebe-se que, em contraste com o UDP, em que há uma fragmentação em camada de rede, ou seja, em datagramas *IPv4*, com o TCP, ocorre uma fragmentação já em camada de transporte, em três segmentos TCP. Tais segmentos estão, por sua vez, em pacotes com *1514 Bytes* os primeiros dois e *1170* o terceiro.



1	0.0000000…	10.0.14.3	10.0.13.3	TCP	74	33518 → 4444 [SYN] Seq=0 Win=42340 Len=0 MSS=146…
2	0.0000215…	10.0.13.3	10.0.14.3	TCP	74	4444 → 33518 [SYN, ACK] Seq=0 Ack=1 Win=43440 Le…
3	0.0000587…	10.0.14.3	10.0.13.3	TCP	66	33518 → 4444 [ACK] Seq=1 Ack=1 Win=42496 Len=0 T…
4	0.0003036…	10.0.14.3	10.0.13.3	TCP	1514	33518 → 4444 [ACK] Seq=1 Ack=1 Win=42496 Len=144…
5	0.0003133…	10.0.13.3	10.0.14.3	TCP	66	4444 → 33518 [ACK] Seq=1 Ack=1449 Win=42496 Len=…
6	0.0003050…	10.0.14.3	10.0.13.3	TCP	1514	33518 → 4444 [ACK] Seq=1449 Ack=1 Win=42496 Len=…
7	0.0003183…	10.0.13.3	10.0.14.3	TCP	66	4444 → 33518 [ACK] Seq=1 Ack=2897 Win=41984 Len=…
8	0.0017108…	10.0.14.3	10.0.13.3	TCP	1170	33518 → 4444 [PSH, ACK] Seq=2897 Ack=1 Win=42496…
9	0.0017315…	10.0.13.3	10.0.14.3	TCP	66	4444 → 33518 [ACK] Seq=1 Ack=4001 Win=42496 Len=…
10	3.0039180…	10.0.14.3	10.0.13.3	TCP	66	33518 → 4444 [FIN, ACK] Seq=4001 Ack=1 Win=42496…
11	3.0044886…	10.0.13.3	10.0.14.3	TCP	66	4444 → 33518 [FIN, ACK] Seq=1 Ack=4002 Win=42496…
12	3.0045588…	10.0.14.3	10.0.13.3	TCP	66	33518 → 4444 [ACK] Seq=4002 Ack=2 Win=42496 Len=…

Figura 10: Pacotes TCP capturados por Wireshark em h1 com pacote de 4000 Bytes.

Assim, colocando-se em comparação os dois primeiros exercícios com o que se analisa neste terceiro, pode-se perceber que o tamanho máximo dos pacotes nos enlaces em que viajam estes é de *1514 Bytes*. Portanto, tanto no caso do UDP como no caso do TCP, foi necessária a fragmentação do pacote de *4000 Bytes*, enquanto que o de *1200* pode ser enviado normalmente, pois o limite do enlace ultrapassa esses *1200* conforme citado.

Além disso, quanto ao número de pacotes UDP enviados, ainda da *Figura 9*, é possível visualizar que existe um único pacote UDP, mas dividido em três pacotes na camada de rede, ou seja, embora sejam efetivamente três pacotes, a divisão não foi feita pela camada de transporte, mas pela camada de rede. Ainda assim, no caso do TCP, foram trocados vários pacotes entre os de sincronização, acknowledge e os de dados úteis e, os pacotes úteis foram fragmentados pelo protocolo TCP, ou seja, em camada de transporte.

Por fim, analisando as *Figuras 11, 12, 13* e *14* a seguir é possível ver que a quantia de dados nos pacotes fragmentados em *IPv4* do envio por UDP é de *1480 Bytes* no dois primeiros e, embora o número apresentado para o terceiro seja *4000*, a quantia é de *1048*, que é o tamanho do pacote menos os dados de cabeçalhos (*34 Bytes* = 1514 - 1480). Já para o TCP, nos primeiros dois pacotes com dados úteis, tem-se uma quantia desses dados igual a *1448 Bytes* em cada e, no terceiro, tem-se uma quantia igual a *1104 Bytes*. Dessa forma, explicita-se o fato de que o número de dados que não fazem parte do *payload* no TCP é consideravelmente maior que no UDP, mesmo em casos de fragmentação necessária (UDP com *42 Bytes* e TCP com *66 Bytes*).

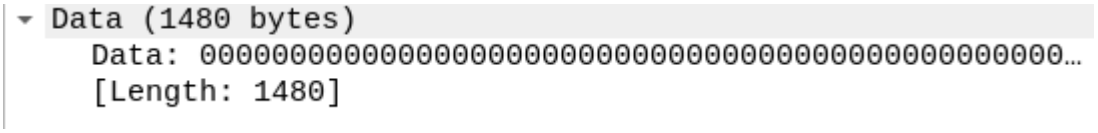


Figura 11: Quantia de dados de aplicação nos dois primeiros pacotes IPv4 (envio UDP).

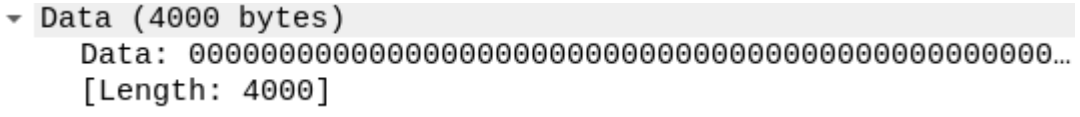


Figura 12: Quantia de dados de aplicação total dos três pacotes IPv4 (pacote UDP).

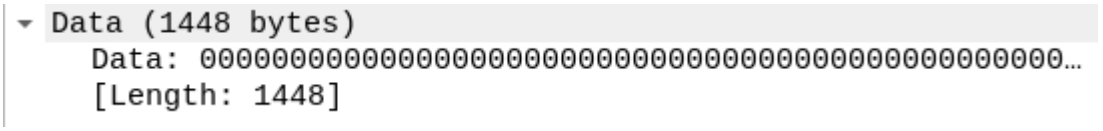


Figura 13: Quantia de dados de aplicação nos dois primeiros pacotes TCP.

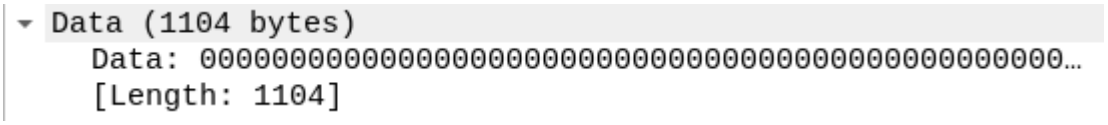


Figura 14: Quantia de dados de aplicação no terceiro pacote TCP.

Exercício 4)

De início, neste exercício, foi aberto, novamente, o *wireshark* pelo host *h1* e, este mesmo host, foi feito servidor com o *netcat*, usando-se o comando "*nc -l 4444*". Feito isso, foi executado o comando "*nc 10.0.13.3 4444*" no terminal de *h4* para que fosse iniciada uma conexão entre os dois hosts citados. Dessa forma, capturou-se os três pacotes, componentes do "*three-way handshake*" do TCP, conforme a *Figura 15* que se segue.

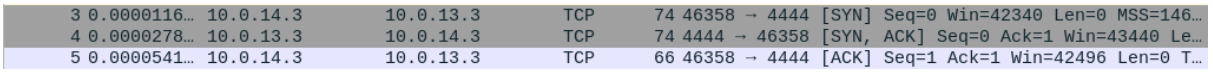


Figura 15: Pacotes TCP de three-way handshake capturados no wireshark .

Nesta figura, vê-se dois pacotes cujo destino foi 10.0.13.3, *h1*, e um cujo destino foi 10.0.14.3, *h4*. Neste contexto, o primeiro pacote (*h4 → h1*) foi um pedido de conexão, tendo, portanto, a *flag SYN* "*setada*", enquanto que o segundo foi um pacote (*h1 → h4*) e reconhecimento e concordância de sincronização, tendo, consequentemente, as *flags SYN* e *ACK* "*setadas*". Por fim, *h1* recebe um último pacote de *h4*, de *flag ACK* "*setada*", reconhecendo a concordância e dando início, de fato, à conexão TCP entre os dois hospedeiros. Vale destacar ainda, que os números de sequência dos pacotes, embora na figura sejam 0, 0 e 1 (números relativos), foram 972065880 (*h4 → h1*), 4042792649 (*h1 → h4*) e 972065881 (*h4 → h1*).

Em seguida, no processo experimental, foi enviado o texto "testando conexao" de *h4* para *h1* por meio do terminal desse primeiro host. Desse modo, tem-se os dois pacotes da *Figura 16*

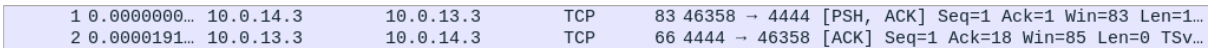


Figura 16: Pacotes TCP de dados e reconhecimento.

Na figura acima, vê-se o pacote de dados e o pacote de reconhecimento, de maneira que, no primeiro, é possível identificar o número de sequência do primeiro *Byte* de dados de aplicação, analisando-se a figura abaixo e reconhecendo que o pacote possui *66 Bytes* de dados "inúteis", conforme citado anteriormente. Assim, sabendo que o número de sequência do pacote é 972065881 e somando a isso 65, tem-se que o primeiro *Byte* de aplicação corresponde ao número de sequência 972065946.

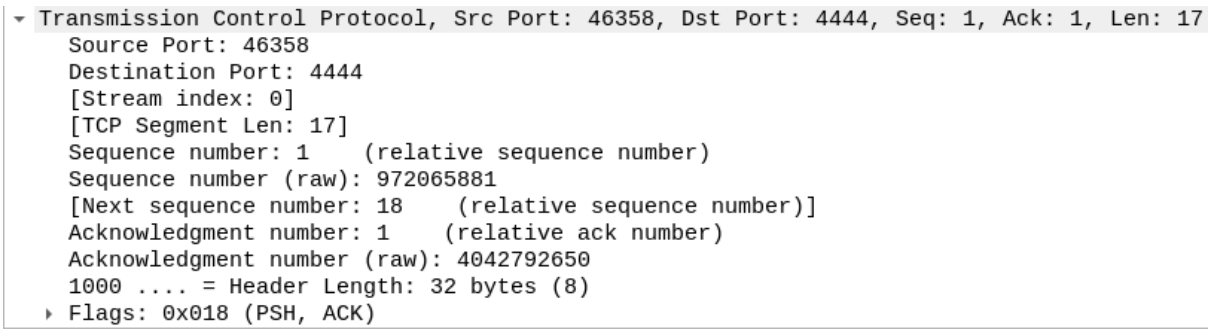


Figura 17: Detalhamento pacote TCP de dados.

Seguindo, das *Figuras* 18 e 19, vê-se respectivamente o tamanho da janela indicado no pacote de dados e no pacote de reconhecimento (o primeiro de *h4* informando *h1* e o segundo ao contrário), sendo elas de tamanho *83* e *85 Bytes*. No entanto, conforme se nota na *Figura 20*, esses valores não chegam nem perto do *MSS* negociado entre os dois hosts, tendo este o tamanho de *1460 Bytes*, logo, um único pacote é suficiente para ambos os casos na situação presente. Finalmente, a respeito ainda desses pacotes, pode-se visualizar melhor os números de sequência e de reconhecimento utilizados por meio da *Figura 21* que se segue. Vale ressaltar ainda que, durante a sincronização, o tamanho da janela enviado em ambos os pacotes *SYN* é de *43440 Bytes*, pois ainda não foram definidos limites reais e durante a sincronização os pacotes enviados não contém *payload*.

```
Window size value: 83
[Calculated window size: 83]
[Window size scaling factor: -1 (unknown)]
```

Figura 18: Janela do pacote de dados.

```
Window size value: 85
[Calculated window size: 85]
[Window size scaling factor: -1 (unknown)]
```

Figura 19: Janela do pacote de reconhecimento.

```
▾ Options: (20 bytes), Maximum segment size, SACK
  ▸ TCP Option - Maximum segment size: 1460 bytes
```

Figura 20: MSS TCP estipulado.

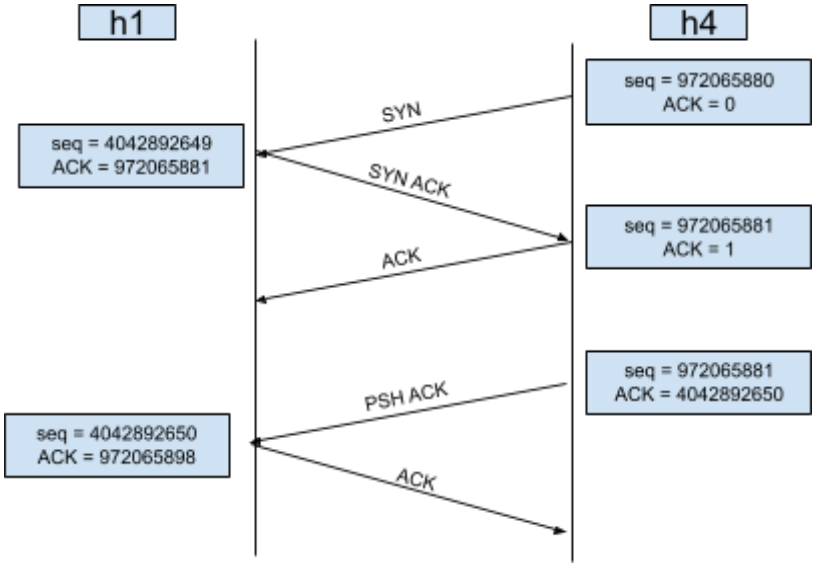


Figura 21: Ilustração de pacotes TCP.

Por fim, fechando a conexão no host *h4*, são capturados os pacotes da *Figura 22*. Nesses pacotes, pode-se ver que, nos dois primeiros, as *flags* marcadas são as *FIN* (*flag* usada para fechamento de conexão) e *ACK*, enquanto, no último pacote, enviado por *h4* para *h1*, tem-se somente a *ACK*, como forma de reconhecimento e correspondência do fechamento. Isso ocorre de modo que, ao receber o pacote com *flag FIN*, o servidor se prepara para encerrar a conexão e responde com um pacote que também contém *FIN*. Deste modo, o host *h4* finaliza a conexão do seu lado (fechando o socket) e envia um pacote de reconhecimento *ACK*, que permite que o servidor encerre a conexão do seu lado (fechando o socket).

1	0.0000000…	10.0.14.3	10.0.13.3	TCP	66 46358 → 4444 [FIN, ACK] Seq=1 Ack=1 Win=83 Len=0…
2	0.0002124…	10.0.13.3	10.0.14.3	TCP	66 4444 → 46358 [FIN, ACK] Seq=1 Ack=2 Win=85 Len=0…
3	0.0002853…	10.0.14.3	10.0.13.3	TCP	66 46358 → 4444 [ACK] Seq=2 Ack=2 Win=83 Len=0 TSva…

Figura 22: Captura de pacotes TCP no fechamento de conexão.

Exercício 5)

Para iniciar este último exercício, foi utilizado o gerador de tráfego *D-ITG* para gerar tráfego entre *h2* e *h3*, sendo o primeiro usado como receptor e o segundo como emissor. Desse modo, o

enlace entre *r1* e *r2* passou a receber uma quantia muito alta de dados, que foi incrementada pelo envio de um pacote enorme de *h4* para *h1* (*100 MBytes*,), que também passou por esse enlace, no entanto, em forma de vários pacotes com até *1514 Bytes*, por conta do limite de tamanho imposto pelos enlaces. Assim, capturando-se os pacotes, novamente, em *h1*, foi possível analisar o comportamento dessa transmissão em tráfego alto, de modo a se observar gráficos desejados para conhecimento da situação. Vale destacar que os comandos usados foram:

"*./ITGRecv*" → em *h2;*

"./ITGSend -a 10.0.15.3 -C 1000000000000 -c 40000 -t 60000 -T UDP" → em *h3.*

Neste segundo, as *flags* usadas são, respectivamente, responsáveis por definir o destino dos pacotes, o número de pacotes, o tamanho dos pacotes em *Bytes*, a duração em milissegundos e, finalmente, o protocolo da camada de transporte.

Assim, obteve-se em *h1* os gráficos a seguir da captura feita:

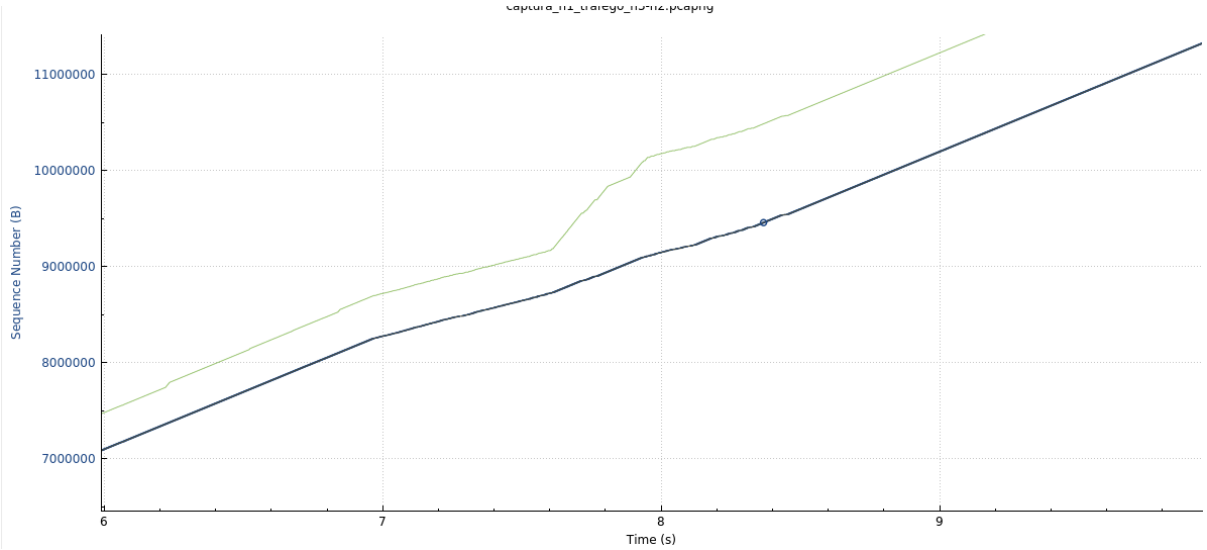


Figura 23: Gráfico de números de sequência com base nos pacotes recebidos em h1

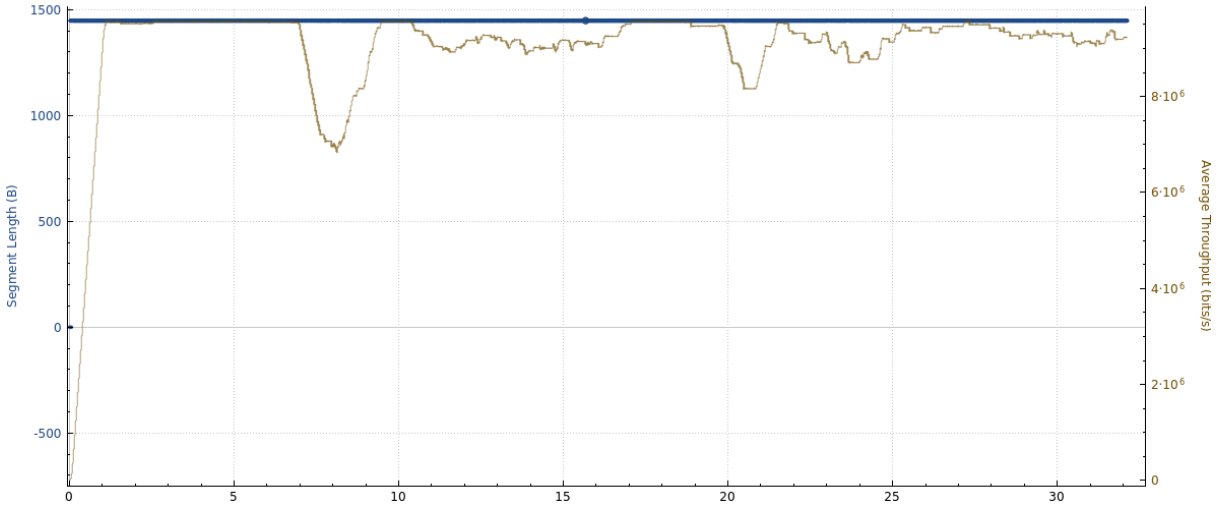


Figura 24: Gráfico de throughput baseado nos pacotes recebidos em h1

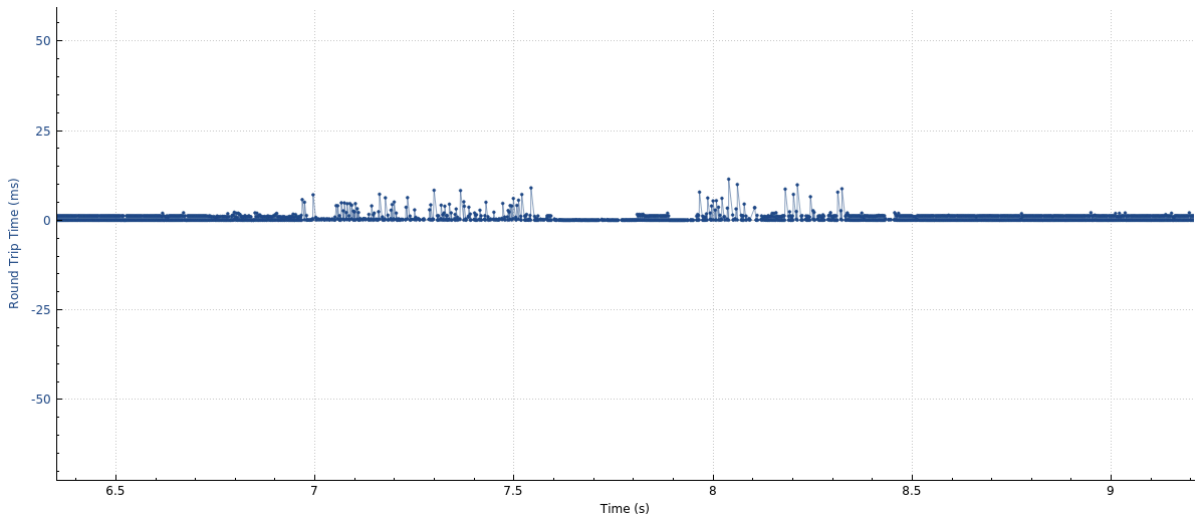


Figura 25: Gráfico de round trip time baseado nos pacotes recebidos em h1 com zoom

Entre os gráficos das figuras acima, encontra-se um gráfico de números de sequência em função do tempo, no qual é possível ver também, em verde, o tamanho da janela de recebimento calculada do cliente. Neste gráfico, com *zoom* no trecho entre 6 e quase 10 segundos, pode-se observar que a linha até aproximadamente 7 segundos tem uma inclinação bastante uniforme, sem muitas alterações. No entanto, pouco antes dos 7 segundos, essa linha tem sua inclinação reduzida com relação ao eixo *x*, o que indica uma queda na largura de banda da conexão. Isso ocorre por causa do congestionamento gerado no enlace entre *r1* e *r2*, que é compartilhado pela rota de *h3* para *h2* e de *h4* para *h1*, usadas simultaneamente. Com isso, para melhor visualização desse fenômeno, é possível recorrer aos gráficos de *throughput* e *RTT*, de maneira que ambos apresentam anomalias nesse mesmo intervalo de tempo citado.

Para melhor compreensão do que foi apresentado, o gráfico da *Figura 24* indica uma redução no *throughput* no mesmo intervalo em que a inclinação da curva de número de sequências tem sua inclinação reduzida, o que é mais uma indicação de congestionamento, que ocasionou atraso na chegada dos pacotes e, consequentemente, redução no número de *bits/s* recebidos (throughput). Além disso, outro fato que explicita esse congestionamento é a alteração repentina na constância do gráfico de *Round Trip Time*, exposto na *Figura 25*, que ocorre, também, no mesmo intervalo de tempo. Neste caso, o que acontece é o aumento do tempo de ida e volta de dados entre os hosts, medido em *h1*, também. Tal quebra de constância, com variações que atingiram até 9 ms de incremento com relação à reta constante que antecede as oscilações, desse modo, percebe-se um aumento de tempo de viagem dos pacotes e, por consequência, apresenta diretamente o efeito do congestionamento.

Conclusão)

Com todo o experimento realizado, pode-se afirmar que foi possível cumprir com o objetivo de aprofundamento nos conhecimentos relacionados ao TCP e ao UDP. Isso porque, foi possível analisar desde o funcionamento básico de cada um dos dois protocolos até uma análise mais aprofundada do TCP em situação de alto fluxo de dados, na qual houve congestionamento. Assim, mostrou-se muito proveitoso e interessante o experimento, retomando conhecimentos e aprofundando os mesmos para os alunos.