Gabriel Kendy Faria Komatsu - 10816711
João Gabriel de Carvalho Ribeiro - 10783027
Vinicius Alves Matias - 10783052

Relatório sobre EP de simulação
ACH2026 - Redes de Computadores

Comparação na transferência de pacotes de dados entre os protocolos TCP e UDP

São Paulo
Outubro / 2020

1 - Resumo da Simulação

Neste EP nosso grupo tem como objetivo analisar diferenças entre dois protocolos de transmissão de pacotes da internet: TCP e UDP. Para tal medimos o tempo tomado por simulações equivalentes utilizando ambos os protocolos enquanto variamos parâmetros como passar arquivos de tamanhos diferentes, alternar a largura de banda, alternar o nó cliente e servidor, alterar o endereço (IPV4) dos nós, alterar tempo para enviar os pacotes, número de pacotes, congestionamento, velocidade e taxa de transmissão.

Para este EP a linguagem utilizada foi C++, o software onde as simulações foram conduzidas foi o ns-3 versão 30.1 e o sistema operacional utilizado foi o Linux Ubuntu 20.04.

2 - Detalhes da implementação

Com a finalidade de comparar redes, tanto a rede TCP quanto a rede UDP possuem topologias similares e a única diferença entre elas é o protocolo usado. As redes possuem 7 nós no total, variando de n0 até n6, com 5 Mbps e 2 ms de delay nos enlaces entre os dispositivos. Estes valores serão alterados em diferentes cenários de testes. Observe que a topologia proposta visa criar uma ligação entre dois dispositivos, criando um link dedicado para a comunicação entre as duas máquinas, mas ligando todos os dispositivos da rede (isto é, há um caminho possível para todos os nós, mesmo que não esteja diretamente conectados).

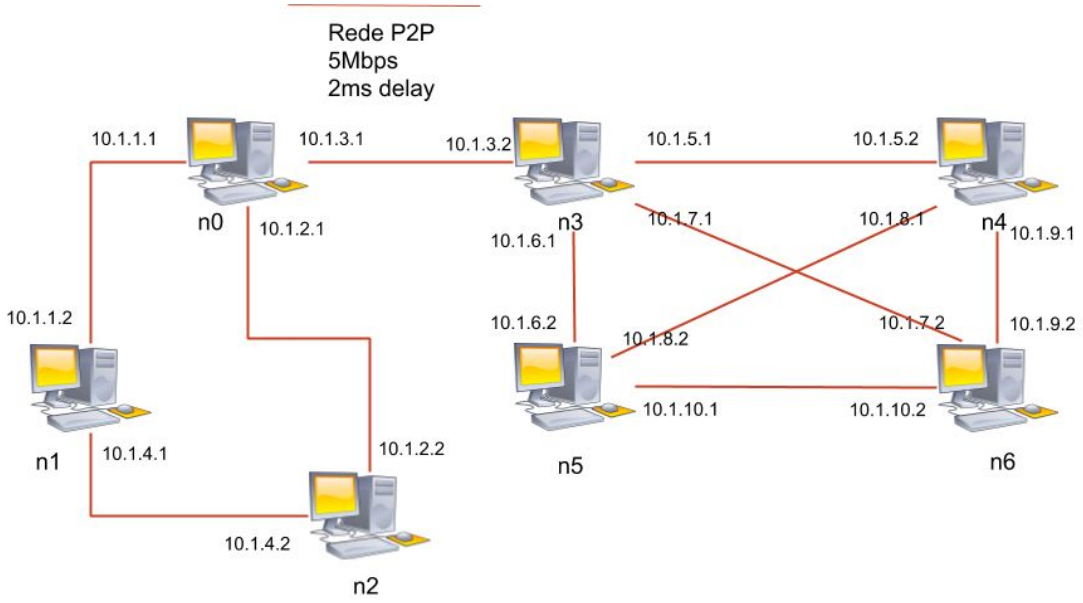

Figura 1 - Rede P2P

O protocolo UDP tem a capacidade de transmitir em broadcast, portanto, é interessante a criação de uma rede mista P2P e Broadcast. A topologia da Rede UDP mista consiste em uma arquitetura Peer-to-Peer (P2P) ligando três dispositivos diferentes e um canal Broadcast que, por sua vez, se comunica com outros três dispositivos. Todos estes dispositivos podem se comunicar a partir de um dispositivo que está ligado a ambas as redes (note que como o protocolo TCP é ideal para a comunicação entre dois pontos, ele não pode suportar o envio “multicast”/ simultâneo a dois dispositivos). A arquitetura da rede é apresentada na Figura 2.

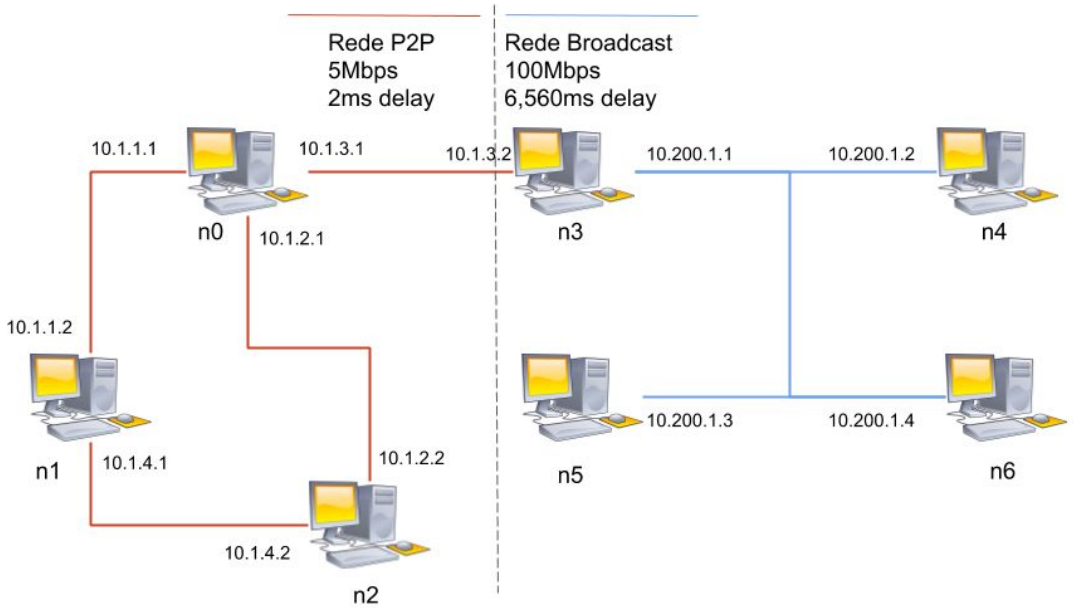

Figura 2 - Rede mista (P2P e Broadcast). Ambas de quatro nós, sendo um nó (n3) a ligação entre as duas redes

Na implementação do script foram utilizadas as classes UdpEchoClientHelper e UdpEchoServerHelper para estabelecer as comunicações entre os nós remetentes e destinatário da rede P2P (protocolo UDP). A instância “server” vai receber os dados do remetente e retornar uma resposta para que possamos analisar o tráfego. Escolhemos isso por facilitar a visualização das respostas de ida e volta, mas lembramos que uma arquitetura Peer to Peer não tem um Cliente e um Servidor (ou seja UdpEchoClientHelper é uma aplicação que está no nó remetente que envia um pacote ao UdpEchoServerHelper, que é o nó destinatário que retornará a mensagem de “ok” de Echo recebido).

Para estabelecer a conexão entre os nós da rede de protocolo TCP nós utilizamos um Socket TCP na porta 49153 dos remetentes, e um Socket TCP na porta 9 do destinatário (nó 6).

3 - Como compilar e executar

Para EPs de simulação, digam qual é o simulador e se preciso fazer alguma configuração especial antes de carregar e rodar a simulação.

1. Colocar o nosso script na pasta scratch que está dentro da pasta do ns-3.
2. Abrir um terminal na pasta do ns-3, no nosso caso “ns-allinone-3.30.1/ns-3.30.1”.
3. Digitar o comando:

```
./waf --run scratch/tcp
./waf --run scratch/udpNoBroadcast
./waf --run scratch/udpWithBroadcast
```

Note que os scripts possuem .cc como extensão, mas nós somente executamos com o nome do arquivo.

- Os arquivos PCAP são logs feitos para cada nó e suas respectivas conexões (as “arestas” entre nós) dependendo da ordem em que foram instanciados. Para executar os outputs gerados com a extensão .pcap utilize o comando tcpdump -r arquivo-nó-aresta.pcap
    - Note que os arquivos foram gerados após compilar o arquivo e serão gerados na pasta ns-allinone-3.30.1/ns-3.30.1
    - Exemplo: tcpdump -r sim1_udp-0-0.pcap vai retornar o cabeçalho UDP do nó 0 com relação a sua conexão com o nó 1. sim1_udp-0-2.pcap é referente à conexão do nó 0 com o nó 3.

- Para executar a animação da simulação no NetAnim (caso deseje), você precisa:
    - Executar o NetAnim (digitando ./NetAnim no diretório netanim-versao do seu ns-3);
    - Escolher então o arquivo simX_protocolo.xml (encontrado no diretório raíz do NS-3 após ter compilado a simulação) pela interface do NetAnim.
    - Exemplo: sim1_udpNoBroadcast.xml

4 - Como ler o código

Os scripts anexados junto ao relatório estão escritos na linguagem C++ e muito bem comentados. Devem ser executados a partir do diretório “ns-allinone-3.30.1/ns-3.30.1” e o script deve estar dentro da pasta “ns-allinone-3.30.1/ns-3.30.1/scratch” usando o comando “./waf --run scratch/redes” do ns-3.

Os “Log do terminal” exibidos durante as simulações na parte de Testes deste relatório foram gerados através de echos e por esse motivo apontam uma duração ligeiramente maior do que a duração verdadeira das simulações.

Os arquivos estão nomeados como simX_protocolo.cc
Onde:

- X é o número da Simulação
- protocolo é o protocolo usado (tcp ou udp)

Ainda que cada código tenha suas particularidades, eles seguem um fluxo comum:

- Definição dos nós
- Configuração e estabelecimento dos canais entre 2 nós
- Instalação da Pilha de Internet (para permitir o uso dos protocolos TCP, UDP e IP)
- Definição dos endereços IPs dos canais
- Construção da Simulação (definição dos pacotes que serão enviados e a forma de envio)

5 - Testes, Análise e Conclusões

Foram definidos 4 cenários diferentes para realização de testes envolvendo transmissão de dados de uma aplicação na rede UDP e também na rede TCP, tendo eles as seguintes características:

1. **Cenário 1:**

Transferência de dados em uma rede P2P de 7 nós comparando o desempenho dos protocolos TCP e UDP na transferência de dados (cenário não extremo, apenas para observar os outputs de tempo, dados, organização das saídas etc). Deverá ser enviado um pacote de 1024 bytes do nó 1 (remetente) ao nó 6 (destinatário) a cada 1 segundo, abrindo a conexão com o remetente entre os segundos 1 e 10 da simulação.

Detalhes:

- Delay de 2ms para envio dos pacotes na rede P2P
- Taxa de Transferência de dados de 5 Mbps na rede P2P
- “Servidor” da aplicação UDP (instância que vai receber os dados do remetente e retornar uma resposta para que possamos analisar o tráfego, escolhemos isso por facilitar a visualização das respostas de ida e volta, mas lembramos que uma arquitetura Peer to Peer não tem um Cliente e um Servidor) aberto na porta 9

1.1 UDP:

Log do terminal:

At time 1s client sent 1024 bytes to 10.1.7.2 port 9
At time 1.01106s server received 1024 bytes from 10.1.1.2 port 49153
At time 1.01106s server sent 1024 bytes to 10.1.1.2 port 49153
At time 1.02212s client received 1024 bytes from 10.1.7.2 port 9

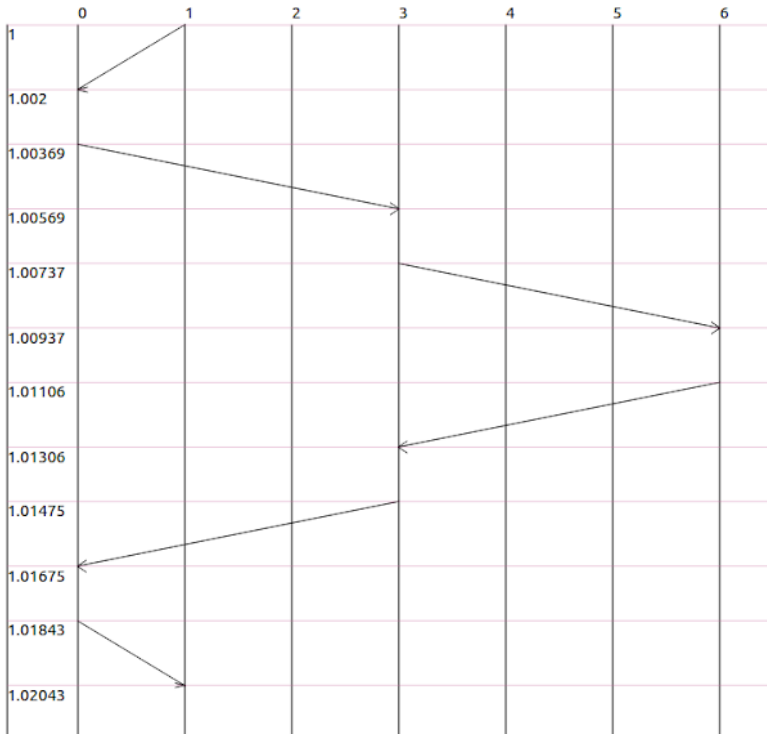

Figura 3 - fluxo do pacote pela rede na Simulação 1 pelo protocolo UDP

O log e a figura acima representam o caminho feito pelo pacote de dados em um rede P2P sob o protocolo UDP. Nós queremos transferir 1024 bytes, então a aplicação remetente cria um pacote dentro de um segmento UDP e o direciona à camada de rede (a que está ligando os computadores). Nosso pacote sairá do nó 1 e deverá chegar ao nó 6, esse pacote deverá encontrar um possível canal que permita a conexão ao destinatário, e como não há uma conexão direta a esse nó, o pacote irá passar pelos computadores que permitam a realização de um caminho à aplicação que queremos chegar. Assim, o pacote vai ao nó 0, que tem uma ligação ao nó 3 (ponto à ponto), que por sua vez tem uma conexão direta a três diferentes dispositivos, sendo um deles o nó 6. Como temos uma aplicação nesse nó 6, ela vai receber esse pacote de dados na porta 9 de seu IP e devolvê-la (a aplicação "servidor" somente replica-o), com o pacote voltando pelo caminho 6->3->0->1.

Diferente do TCP, o UDP não precisa estabelecer a conexão por três vias, apenas envia o pacote com o segmento UDP. Por esse motivo, não há atraso por padrão no estabelecimento da conexão entre dois nós. Na nossa simulação implementamos um atraso de 2ms para simbolizar o reconhecimento, manipulação e envio de um pacote (ou seja, não é um atraso de conexão). Isso é notável ao analisar a figura 3 quando, por exemplo, o pacote sai do nó 0 ao nó 3 com um atraso de propagação de 2ms (sai em 1.00369 e chega em 1.00569), como só estamos enviando um pacote de 1024 bytes e estes cabem nos links dedicados de 5Mbps entre os computadores, não é necessário esperar o intervalo de temporização de 1 segundo para enviar outro pacote (pois esse já chegou em menos de 1.03 segundos).

Vale lembrar que a aplicação remetente está instalada na máquina de IP 10.1.1.2 (nó 1 da ligação nó 1 - nó 0) com um socket implementado para enviar pela porta 49153 os pacotes de dados, especificando que deseja chegar na porta 9 do nó 6 (que tem o IP 10.1.7.2 na ligação nó 3 - nó 6), que é a máquina com a aplicação remetente, ou seja, que recebe o pacote de dados na porta 9 (a aplicação instanciada no código tem uma implementação de socket que permite essa manipulação).

Figura 4 - Estrutura do Segmento UDP (KUROSE, 2014)

PCAPs (referente ao Segmento UDP):
1-0:-2:-2:-59:-59.000000 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.022118 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024

0-2:-2:-2:-59:-59.003686 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.018432 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024
.
.
.

3-0:-2:-59:-59.007372 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.014745 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024

0-0-: -2:-59:-59.003686 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.018432 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024

O UDP é um protocolo de transporte bem mais simples que o TCP, o que é perceptível pelo cabeçalho do segmento UDP. Os logs acima estão em ordem cronológica, isto é, na ordem que acontecem as conexões Conexões nesse contexto se referem às ligações entre os pontos da rede, não o processo de conexão em três vias, que é o caso somente do protocolo TCP, que é um protocolo orientado à conexão, isto é, pelo protocolo UDP será enviado o pacote sem a garantia do estabelecimento da conexão entre os nós.

Assim como descrito no tópico 3, o início do log / nome do arquivo é um nó, seguido por hífen (“-”) e uma a ordem de ligação desse nó. Por exemplo:

- nó 1, ligação 0 -> nó 1 - nó 0 (tem duas conexões, a primeira com o nó 0 e a segundo com o nó 2);
- nó 0, ligação 2 -> nó 0 - nó 3 (tem três conexões, a primeira com n1, a segunda com n2 e a terceira com n3).

Esse log também traz uma maneira de reconhecer o momento de envio (59:59:xyz) mas não será necessário usá-lo.
Os segmentos trocados entre as ligações da rede P2P praticamente solicitam uma transferência de 1024 bytes (length) do remetente 10.1.1.2:49153 ao destinatário 10.1.7.2, informando que pode descartar o pacote ao chegar no destinatário (não enviar à outra camada, apenas realizar o tratamento da aplicação que é enviar um echo). A checksum não é exibida, mas todos os segmentos chegaram aos nós que deveriam chegar sem estarem corrompidos (não há nenhuma mensagem necessária a ser enviada também). O protocolo de transporte é especificado como UDP, obviamente, e também é, posteriormente, indicado o caminho de volta.

1.2 TCP:

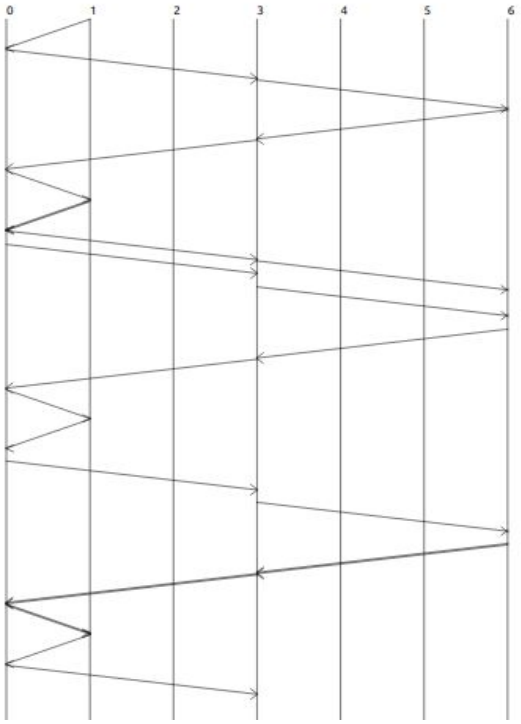

Figura 5 - fluxo do pacote pela rede na Simulação 1 pelo protocolo TCP

De Id	Para Id	Tempo
1	0	1
0	3	1.00209
3	6	1.00419
6	3	1.00628
3	0	1.00837
0	1	1.01046
1	0	1.01256
1	0	1.01264
0	3	1.01464
0	3	1.01559
3	6	1.01673
3	6	1.01853
6	3	1.02148
3	0	1.02356
0	1	1.02565
1	0	1.02773
0	3	1.0306
3	6	1.03347
6	3	1.03634
6	3	1.03642

3	0	1.03842
3	0	1.03851
0	1	1.04051
0	1	1.0406
1	0	1.04268
0	3	1.04477

Tabela 1 - Nós e tempo da Figura 5

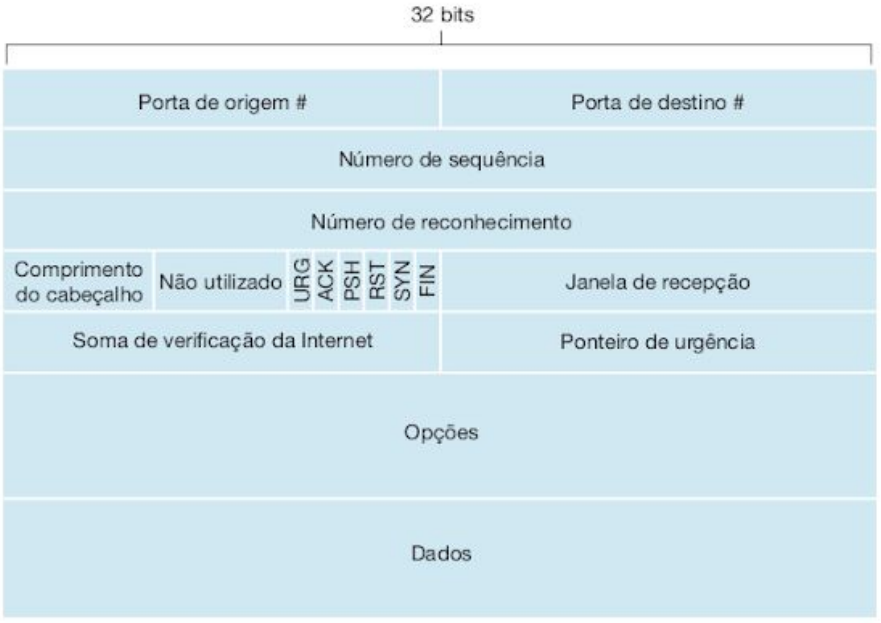

Figura 6 - Estrutura do Segmento TCP

PCAPS do nó remetente (n1) ao nó destinatário(n6) (cabeçalho TCP):
reading from file node_tcp-0-0.pcap, link-type PPP (PPP)
IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [S], seq 0, win 65535, options [TS val 1000 ecr 0,wscale 2,sackOK,eol], length 0
IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [S.], seq 0, ack 1, win 65535, options [TS val 1006 ecr 1000,wscale 2,sackOK,eol], length 0
IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 0
IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 1:537, ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 536
IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 537, win 32768, options [TS val 1021 ecr 1012,eol], length 0
IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [F.], seq 537:1025, ack 1, win 32768, options [TS val 1027 ecr 1021,eol], length 488
IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 1026, win 32768, options [TS val 1036 ecr 1027,eol], length 0

IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [F.], seq 1, ack 1026, win 32768, options [TS val 1036 ecr 1027,eol], length 0
IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], ack 2, win 32768, options [TS val 1042 ecr 1036,eol], length 0

O cabeçalho TCP é bem mais complexo que o UDP, o que é perceptível pelos logs acima. A imagem gerada no terminal foi cortada apenas para exibir o nó remetente ao destinatário (para resumir os dados, geramos os mesmos logs do nó 1 ao nó 6 à todas as conexões entre dois computadores, sendo essas ligações as apresentadas na Figura 5 e tabela 1 - isto é, todos os logs de conexões possíveis vão exibir a conexão n1 - n6), já que a ideia é semelhante às outras conexões Point to Point.

Sendo assim, o primeiro log (conexão entre os nós n0 e n6 ponto à ponto) mostra primeiramente o envio do pacote de um segmento com a solicitação de conexão ao outro ponto (no caso, nó 1), ou seja, o socket do endereço 10.1.1.2:49153 envia um segmento com a Flag SYN = 1 ao endereço 10.1.7.2 (nó 6) que tem um socket que captura essa solicitação e envia um segmento TCP com um SYN = 1 e ACK = 1, ou seja, avisando que recebeu a solicitação de conexão e deu o ok para iniciar. Quando chega essa informação no nó 0, ele tem a permissão de enviar os dados ao nó 1, enviando um ACK = 1 ao 10.1.7.2 (aqui temos a apresentação de três vias para gerar a conexão TCP).

As opções do cabeçalho enviado são:

- TS val: É utilizado para calcular o Round-Trip-Time (RTT), ou seja, sendo útil para calcular quando deve enviar novamente um pacote devido o timeout
- ECR: Echo Reply, calculado sob o valor TS para identificar o retorno do Echo
- sackOK: permite o recebimento apenas dos dados não recebidos
- wscale: Opção para aumentar o tamanho padrão da janela
- eol: End of List, sem mais opções

A janela / buffer de dados do remetente e do destinatário no protocolo TCP são equivalentes, isto é, o valor de win do remetente deverá ser igual ao do destinatário e com isso pode-se enviar pacotes sabendo que não haverá uma janela maior do que a outra e reduzir a possível perda de pacotes na rede.

Segundo o log, assim que finalizada a apresentação em três etapas, o remetente constrói um pacote de 526 bytes, numseq=537 e ACK=1 (porque recebeu a mensagem anterior) e envia ao nó destinatário. Esse recebe, envia o ACK = 537 (próximo byte que está esperando) e atualiza sua janela para ficar igual à do destinatário. O que está enviando os dados envia um novo pacote com os 488 bytes restantes para completar os 1024 e já envia junto uma flag FIN para finalizar a conexão, o remetente recebe e envia o ACK = 1026 junto ao FIN. O nó 1 recebe a confirmação do envio dos dados e envia um ACK para registrar isso. Os números de sequência e processo mais detalhado estão no log.

Assim sendo, comprovamos que nessa rede de configurações o mais próximo possível, o cenário com UDP tende a ser mais rápido que o cenário com protocolo TCP. Isso ocorre porque o protocolo TCP deve garantir que há uma conexão entre os nós remetente e destinatário e pelo atraso padrão que esse protocolo precisa para garantir a integridade das

informações (janela, as opções do cabeçalho sobre tempo etc). O protocolo não tem essas garantias e, também, já enviou o pacote de 1024 bytes de uma vez (o TCP separou em dois pacotes, reconhecidos pelo ACK e número de sequência) usando toda a largura de banda que lhe foi disponibilizada. No nosso cenário de transferência simples de dados, não há muita necessidade de se usar TCP e seus recursos de entrega confiável, sendo o UDP mais rápido para esse cenário (assim como o esperado).

2. Cenário 2:

Exemplo mais extremo de fluxo de dados que o cenário 1: Queremos enviar um pacote de dados de 20480 bytes com atraso alto para o envio na Rede com protocolo TCP e UDP e identificar como ela se comporta, inclusive o controle de fluxo. Esse cenário é útil para identificar uma rede com atraso alto que não estava preparada para receber um alto tráfego.

Detalhes:

- Delay de 100ms na rede P2P
- Taxa de Transferência de dados de 256 kbps nos enlaces da rede P2P
- Aplicação destinatária aberta na porta 9
- Tamanho total a ser enviado: 20480

2.1 UDP:

Log do terminal:

At time 1s client sent 20480 bytes to 10.1.7.2 port 9
At time 2.04375s server received 20480 bytes from 10.1.1.2 port 49153
At time 2.04375s server sent 20480 bytes to 10.1.1.2 port 49153
At time 3.0875s client received 20480 bytes from 10.1.7.2 port 9

O remetente quer enviar um pacote de 20480 bytes do nó 1 (IP 10.1.1.2) a partir da porta 49153 para o servidor estabelecido no nó 6 (IP 10.1.7.2) pela porta 9. Porém o protocolo UDP somente permite o envio de pacotes com no máximo 1472 bytes, portanto o que aconteceu foi a divisão dos 20480 bytes em 14 pacotes de 1462,8 bytes, uma vez que o protocolo UDP tenta extrair todo o potencial da taxa de transmissão não se importando com outros utilizadores da rede. Todos os 14 pacotes foram enviados para o servidor dentro de um intervalo de 0.90406 segundos (nó 3 para nó 6), a partir do segundo 1, e após 1.94781 segundos de simulação o último pacote foi retornado ao cliente (nó 0 para nó 1), como pode ser notado a partir da Figura 7 e Tabela 2 mais abaixo.

Obs: Alguns conceitos fundamentais que foram citados no cenário 1 que se repetem nos outros cenários não serão necessariamente repetidos. Daqui pra frente exaltaremos as diferenças.

reading from file sim2_udp-6-0.pcap, link-type PPP (PPP)
-2:-59:-59.440812 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, bad length 20480 > 1472
-2:-59:-59.487750 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.534687 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.581625 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.628562 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.675500 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.722437 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.769375 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.816312 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.863250 IP 10.1.1.2 > 10.1.7.2: udp

-2:-59:-59.910187 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-59.957125 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-58.004062 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-58.043750 IP 10.1.1.2 > 10.1.7.2: udp
-2:-59:-58.043750 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, bad length 20480 > 1472
-2:-59:-58.090687 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.137625 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.184562 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.231500 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.278437 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.325375 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.372312 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.419250 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.466187 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.513125 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.560062 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.607000 IP 10.1.7.2 > 10.1.1.2: udp
-2:-59:-58.653937 IP 10.1.7.2 > 10.1.1.2: udp

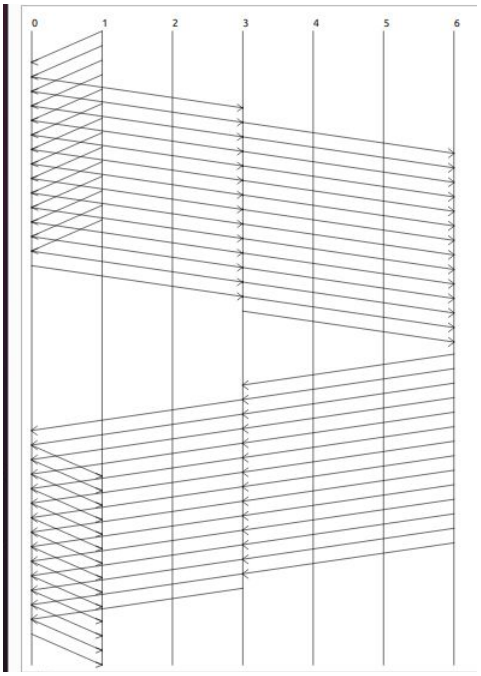

Figura 7 - Representação do pacote viajando pela rede UDP(gerado pelo NetAnim)

De Id	Para Id	Tempo
1	0	1

1	0	1.04694
1	0	109.387
1	0	114.081
0	3	114.694
1	0	118.775
...	...	...
0	3	138.163
3	6	138.775
1	0	142.244
...	...	...
3	6	1.90406
6	3	204.375
...	...	...
3	0	280.088
0	1	2.807
0	1	285.394
0	1	290.088
0	1	294.781

Tabela 2 - Nós e tempo da Figura 7

2.2 TCP:

reading from file sim2_tcp-6-0.pcap, link-type PPP (PPP)
-2:-59:-59.305437 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [S], seq 0, win 65535, options [TS val 1000 ecr 0,wscale 2,sackOK,eol], length 0
-2:-59:-59.305437 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [S.], seq 0, ack 1, win 65535, options [TS val 1305 ecr 1000,wscale 2,sackOK,eol], length 0
-2:-59:-59.915937 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], ack 1, win 32768, options [TS val 1610 ecr 1305,eol], length 0
-2:-59:-59.967874 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 1:537, ack 1, win 32768, options [TS val 1610 ecr 1305,eol], length 536
-2:-59:-59.967874 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 537, win 32768, options [TS val 1967 ecr 1610,eol], length 0
-2:-59:-58.628249 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 537:1073, ack 1, win 32768, options [TS val 2272 ecr 1967,eol], length 536
-2:-59:-58.646687 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 1073:1609, ack 1, win 32768, options [TS val 2272 ecr 1967,eol], length 536
-2:-59:-58.646687 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 1609, win 32768, options [TS val 2646 ecr 2272,eol], length 0
-2:-59:-57.307062 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 1609:2145, ack 1, win 32768, options [TS val 2951 ecr 2646,eol], length 536

-2:-59:-57.325499 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 2145:2681, ack 1, win 32768, options [TS val 2951 ecr 2646,eol], length 536
...
-2:-59:-54.222312 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 19833, win 32768, options [TS val 6222 ecr 5830,eol], length 0
-2:-59:-54.240749 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 19833:20369, ack 1, win 32768, options [TS val 5866 ecr 5561,eol], length 536
-2:-59:-54.245937 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [F.], seq 20369:20481, ack 1, win 32768, options [TS val 5866 ecr 5561,eol], length 112
-2:-59:-54.245937 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 20482, win 32768, options [TS val 6245 ecr 5866,eol], length 0
-2:-59:-54.247624 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [F.], seq 1, ack 20482, win 32768, options [TS val 6245 ecr 5866,eol], length 0
-2:-59:-54.857749 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], ack 2, win 32768, options [TS val 6552 ecr 6245,eol], length 0

Após realizar a apresentação de três vias, o protocolo TCP identifica o tamanho dos bytes e envia em pacotes de 532 bytes, assim como no cenário 1. Esse é um valor que não ocupa toda a largura de banda, visto que o TCP tem essa preocupação. A janela do destinatário não chegou em momento nenhum a não comportar mais dados, portanto, não foi um problema. Para cada pacote recebido no destinatário foi retornado o ACK equivalente, não chegando a um problema de congestionamento da rede, até porque a taxa de transmissão conseguia suportar esses segmentos sendo enviados nas duas vias. O último pacote de dados, para fechar a conta, deveria ter 112 bytes e apontar o FIN da conexão (assim como no cenário 1). A figura 8 mostra graficamente esse fluxo de pacotes, nela é possível identificar que há um maior volume de dados na metade inferior da figura, muito decorrente dos ACKs cumulativos que vão sendo enviados praticamente no mesmo momento que são enviados novos bytes ao remetente. A simulação durou até os 6.75606 segundos (5.75606 segundo após a ativação das máquinas, no segundo 1)

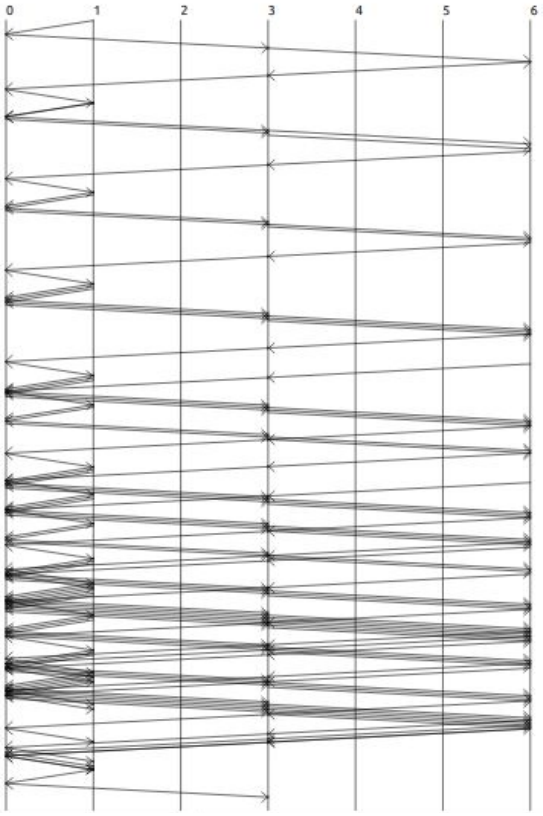

Figura 8 - Representação dos pacotes TCP viajando pela rede (gerado pelo NetAnim)

De Id	Para Id	Tempo
1	0	1
0	3	1.10181
3	6	1.20362
6	3	1.30544
3	0	1.40725
0	1	1.50906
1	0	1.61088
1	0	1.61256
0	3	1.71256
0	3	1.731
3	6	1.81425
3	6	1.84944
...	...	...
3	6	6.14075
3	0	6.14244
6	3	6.18544

6	3	6.22231
0	1	6.24412
6	3	6.24594
6	3	6.24762
3	0	6.28712
3	0	6.324
3	0	6.34762
3	0	6.34931
0	1	6.38881
0	1	6.42569
0	1	6.44931
0	1	6.451
1	0	6.55269
0	3	6.65437
3	6	6.75606

Tabela 3 - Nós e tempo da Figura 8

Comparando o desempenho das duas redes, percebemos que, ainda que o protocolo UDP continue sendo o mais rápido para transmissão de dados, ele não faz uma gestão do tamanho dos pacotes, utilizando todos os bytes que couberem em um canal de transmissão. Precisamos notar também que na rede com protocolo UDP, assim como há um envio seguido de pacotes, há um retorno constante também (nesse caso foram todos em ordem, algo que não necessariamente acontece em UDP), já no protocolo TCP há um feedback constante dos dados que estão sendo transmitidos (e ainda assim, com a garantia de recebimento ordenado).

3. Cenário 3:

Controle de fluxo no UDP e TCP: Queremos identificar como os protocolos reagem a pontos de congestionamento na rede utilizando 2 remetentes, os nós 1 e 2, ambos enviando respectivamente 1024 e 20480 bytes para um destinatário em comum, o nó 6, sem nenhum intervalo para envio de pacotes.

Detalhes:

- Delay de 2ms na rede P2P
- Taxa de Transferência de dados de 512 kbps na rede P2P
- Tamanho total a ser enviado: 1024 bytes do nó 1
- Tamanho total a ser enviado: 20480 bytes do nó 2

3.1 UDP

Log do terminal:

At time 1s client sent 1024 bytes to 10.1.7.2 port 9
At time 1s client sent 20480 bytes to 10.1.7.2 port 9
At time 1.05541s server received 1024 bytes from 10.1.1.2 port 49153
At time 1.05541s server sent 1024 bytes to 10.1.1.2 port 49153
At time 1.11081s client received 1024 bytes from 10.1.7.2 port 9
At time 1.38734s server received 20480 bytes from 10.1.2.2 port 49153
At time 1.38734s server sent 20480 bytes to 10.1.2.2 port 49153
At time 1.76522s client received 20480 bytes from 10.1.7.2 port 9

Nesse cenário existem 2 remetentes por essa razão ambos enviaram seus bytes assim que a simulação começou, ou seja, ambos remetentes e destinatário iniciaram suas operações quando a simulação atingiu 1 segundo de execução. Assim como no cenário anterior, aqui o protocolo UDP se depara com um pacote maior que 1472 portanto realiza a mesma ação de dividir em pacotes menores. No momento 1.09234 da simulação (pode ser encontrado na Tabela 4) os 1024 bytes enviados pelo nó 1 retornam de sua ida até o nó 6. Como é possível notar na Figura 9, o nó 3 é utilizado a todo momento, o que causa um congestionamento na rede e uma diminuição da velocidade de entrega dos bytes.

PCAPS:

3-0:reading from file sim3_udp-3-0.pcap, link-type PPP (PPP)
-2:-59:-59.055406 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.085875 IP 10.1.2.2.49153 > 10.1.7.2.discard: UDP, bad length 20480 > 1472
-2:-59:-59.055406 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024
-2:-59:-59.109343 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.132812 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.156281 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.179750 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.203218 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.226687 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.250156 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.273625 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.297093 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.320562 IP 10.1.2.2 > 10.1.7.2: udp

-2:-59:-59.344031 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.367500 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.387343 IP 10.1.2.2 > 10.1.7.2: udp
-2:-59:-59.387343 IP 10.1.7.2.discard > 10.1.2.2.49153: UDP, bad length 20480 > 1472
-2:-59:-59.410812 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.434281 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.457750 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.481218 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.504687 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.528156 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.551625 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.575093 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.598562 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.622031 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.645500 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.668968 IP 10.1.7.2 > 10.1.2.2: udp
-2:-59:-59.692437 IP 10.1.7.2 > 10.1.2.2: udp

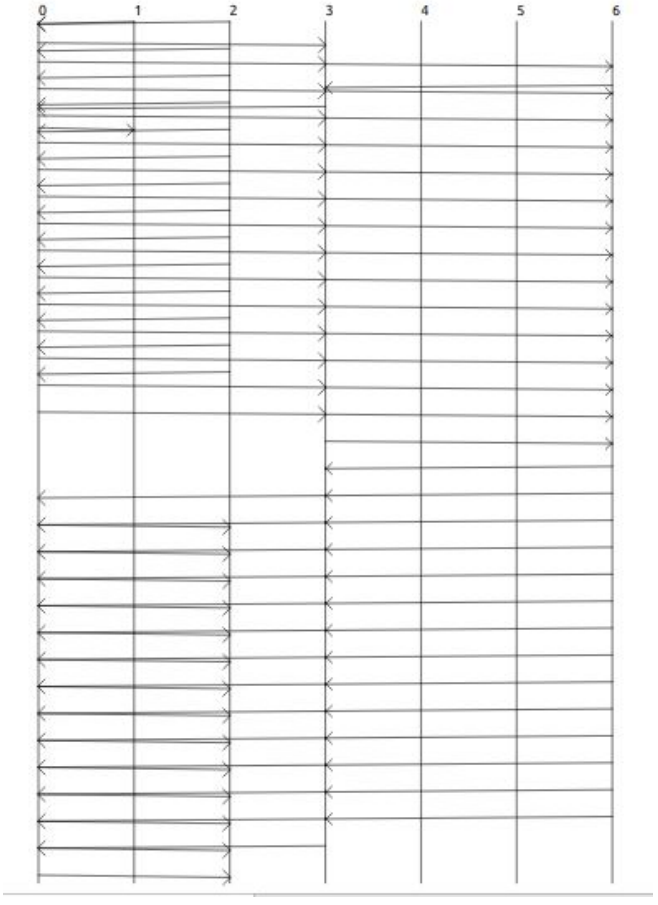

Figura 9 - representação do pacote viajando pela rede UDP (gerado pelo NetAnim).

De Id	Para Id	Tempo
1	0	1
2	0	1
0	3	1.01847

2	0	1.02347
0	3	1.03494
3	6	1.03694
2	0	1.04694
6	3	1.05541
0	3	1.05841
3	6	1.06041
2	0	1.07041
3	0	1.07387
0	3	1.08187
3	6	1.08387
0	1	1.09234
2	0	1.09387
0	3	1.10534
...	...	...
0	2	1.67297
6	3	1.69244
3	0	1.69444
0	2	1.69644
3	0	1.71791
0	2	1.71991
0	2	1.74337

Tabela 4 - Nós e tempo da Figura 9.

3.2 TCP:

PCAPS:

m3_tcp-6-0.pcap (remetente 10.1.1.2:49153) -> nó 1
reading from file sim3_tcp-6-0.pcap, link-type PPP (PPP)
-2:-59:-59.006278 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [S], seq 0, win 65535, options [TS val 1000 ecr 0,wscale 2,sackOK,eol], length 0
-2:-59:-59.006278 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [S.], seq 0, ack 1, win 65535, options [TS val 1006 ecr 1000,wscale 2,sackOK,eol], length 0
-2:-59:-59.006371 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [S], seq 0, win 65535, options [TS val 1000 ecr 0,wscale 2,sackOK,eol], length 0
-2:-59:-59.006371 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [S.], seq 0, ack 1, win 65535, options [TS val 1006 ecr 1000,wscale 2,sackOK,eol], length 0
-2:-59:-59.018815 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 0

-2:-59:-59.018908 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 0
-2:-59:-59.021475 IP 10.1.1.2.49153 > 10.1.7.2.discard: Flags [.], seq 1:537, ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 536
-2:-59:-59.021475 IP 10.1.7.2.discard > 10.1.1.2.49153: Flags [.], ack 537, win 32768, options [TS val 1021 ecr 1012,eol], length 0
-2:-59:-59.022419 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 1:537, ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 536
[…]
-2:-59:-59.131833 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 19833, win 32768, options [TS val 1131 ecr 1113,eol], length 0
-2:-59:-59.132777 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 19833:20369, ack 1, win 32768, options [TS val 1123 ecr 1116,eol], length 536
-2:-59:-59.133043 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [F.], seq 20369:20481, ack 1, win 32768, options [TS val 1123 ecr 1116,eol], length 112
-2:-59:-59.133043 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 20482, win 32768, options [TS val 1133 ecr 1123,eol], length 0
-2:-59:-59.133129 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [F.], seq 1, ack 20482, win 32768, options [TS val 1133 ecr 1123,eol], length 0
-2:-59:-59.145647 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], ack 2, win 32768, options [TS val 1139 ecr 1133,eol], length 0

m3_tcp-2-0.pcap (remetente 10.1.2.2.49153) -> nó 2
reading from file sim3_tcp-2-0.pcap, link-type PPP (PPP)
-2:-59:-59.000000 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [S], seq 0, win 65535, options [TS val 1000 ecr 0,wscale 2,sackOK,eol], length 0
-2:-59:-59.012649 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [S.], seq 0, ack 1, win 65535, options [TS val 1006 ecr 1000,wscale 2,sackOK,eol], length 0
-2:-59:-59.012649 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 0
-2:-59:-59.012735 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 1:537, ack 1, win 32768, options [TS val 1012 ecr 1006,eol], length 536
-2:-59:-59.028678 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 537, win 32768, options [TS val 1022 ecr 1012,eol], length 0
-2:-59:-59.028678 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 537:1073, ack 1, win 32768, options [TS val 1028 ecr 1022,eol], length 536
-2:-59:-59.029622 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 1073:1609, ack 1, win 32768, options [TS val 1028 ecr 1022,eol], length 536
-2:-59:-59.044713 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 1609, win 32768, options [TS val 1038 ecr 1028,eol], length 0
-2:-59:-59.044713 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 1609:2145, ack 1, win 32768, options [TS val 1044 ecr 1038,eol], length 536
-2:-59:-59.045657 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 2145:2681, ack 1, win 32768, options [TS val 1044 ecr 1038,eol], length 536
-2:-59:-59.046601 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], seq 2681:3217, ack 1, win 32768, options [TS val 1044 ecr 1038,eol], length 536
[…]

-2:-59:-59.124889 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [F.], seq 20369:20481, ack 1, win 32768, options [TS val 1123 ecr 1116,eol], length 112
-2:-59:-59.126777 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 15545, win 32768, options [TS val 1120 ecr 1109,eol], length 0
-2:-59:-59.128665 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 16617, win 32768, options [TS val 1122 ecr 1109,eol], length 0
-2:-59:-59.130553 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 17689, win 32768, options [TS val 1124 ecr 1111,eol], length 0
-2:-59:-59.132441 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 18761, win 32768, options [TS val 1126 ecr 1113,eol], length 0
-2:-59:-59.138092 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 19833, win 32768, options [TS val 1131 ecr 1113,eol], length 0
-2:-59:-59.139302 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [.], ack 20482, win 32768, options [TS val 1133 ecr 1123,eol], length 0
-2:-59:-59.139388 IP 10.1.7.2.discard > 10.1.2.2.49153: Flags [F.], seq 1, ack 20482, win 32768, options [TS val 1133 ecr 1123,eol], length 0
-2:-59:-59.139388 IP 10.1.2.2.49153 > 10.1.7.2.discard: Flags [.], ack 2, win 32768, options [TS val 1139 ecr 1133,eol], length 0

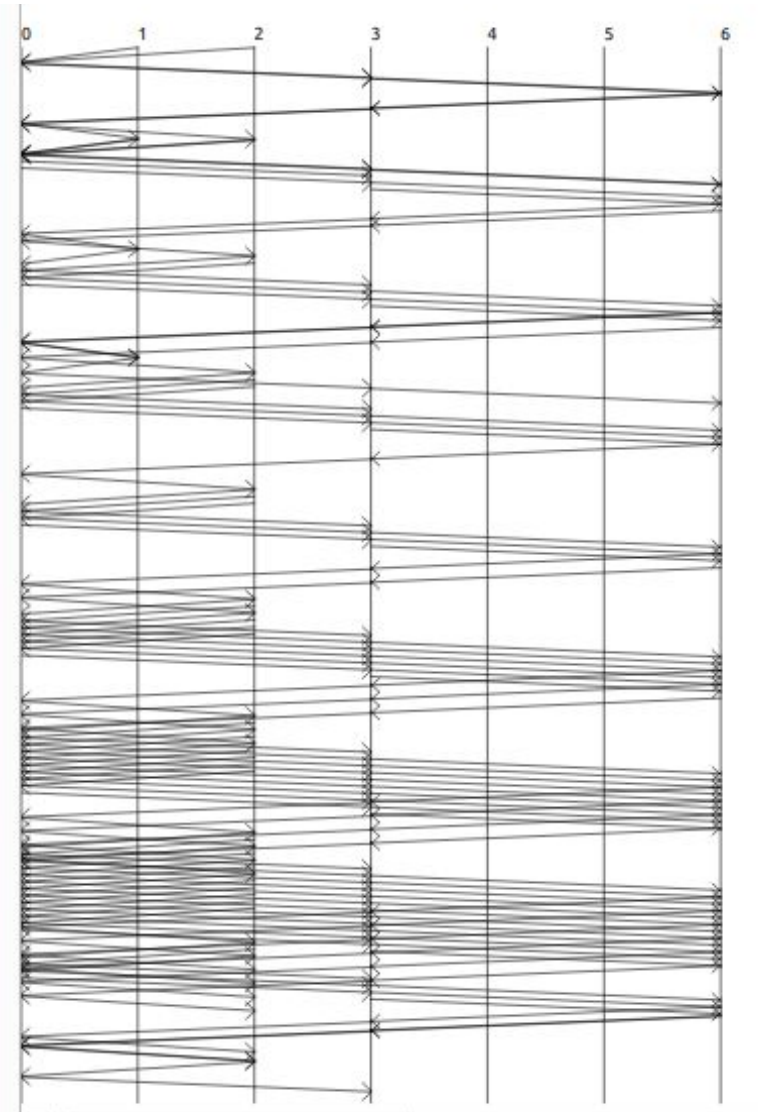

Figura 10 - representação do pacote viajando pela rede TCP (gerado pelo NetAnim).

De Id	Para Id	Tempo
1	0	1
2	0	1
0	3	1.00209
0	3	1.00219
3	6	1.00419
3	6	1.00428
6	3	1.00628
6	3	1.00637
3	0	1.00837
3	0	1.00846
0	1	1.01046
0	2	1.01056
1	0	1.01256
1	0	1.01264
2	0	1.01265
2	0	1.01274
0	3	1.01464
0	3	1.01474
0	3	1.01559
0	3	1.01653
3	6	1.01673
3	6	1.01682
3	6	1.01853
3	6	1.01948
6	3	1.02148
6	3	1.02242
3	0	1.02356
3	0	1.02451
0	1	1.02565
0	2	1.02659
1	0	1.02773
2	0	1.02868
2	0	1.02962
0	3	1.0306
0	3	1.03162
0	3	1.03257
3	6	1.03347
3	6	1.03457
3	6	1.03551

6	3	1.03634
6	3	1.03642
3	0	1.03842
6	3	1.03845
3	0	1.03851
0	1	1.04051
3	0	1.04054
0	**1**	**1.0406**
0	2	1.04263
...	...	...
6	3	1.13313
3	0	1.13392
3	0	1.13513
3	0	1.13522
0	2	1.13601
0	2	1.13722
0	2	1.1373
2	0	1.13939
0	3	1.14148

Tabela 6 - Nós e tempo da Figura 10

Com dois nós remetentes na rede P2P atuando enviando pacotes de dados para um mesmo nó da rede não houve nenhum comportamento diferente pelo protocolo TCP, conseguindo enviar os dados sem causar congestionamento ou problemas na janela de envio do TCP. Foram enviados os bytes do nó 1 até o segundo 1.0406, em seguida, foram tratados os dados apenas do nó 2. Os pacotes continuavam sendo criados com 536 bytes, o que permitia um fluxo constante na rede, impedindo que o congestionamento ocorresse.

Conclusão: TCP foi mais rápido neste cenário devido ao congestionamento gerado pelo consumo completo da taxa de transmissão de pacotes do protocolo UDP.

4. Cenário 4:

Comparação Rede Mista e Rede P2P sob o protocolo UDP: Comparar o tráfego de dados das duas redes apenas para o Protocolo UDP.

- Delay de 2ms na rede P2P
- Taxa de Transferência de dados de 5 Mbps na rede P2P
- Delay de 2ms na rede Broadcast
- Taxa de Transferência de dados de 5 Mbps na rede Broadcast
- Remetentes enviam 1 pacote de tamanho de 1024 bytes com um intervalo de 1 segundo

4.1 UDP sem broadcast (sim1_udpNoBroadcast):

Log do terminal:

At time 1s client sent 1024 bytes to 10.1.7.2 port 9
At time 1.01106s server received 1024 bytes from 10.1.1.2 port 49153
At time 1.01106s server sent 1024 bytes to 10.1.1.2 port 49153
At time 1.02212s client received 1024 bytes from 10.1.7.2 port 9

O cliente, nó 1 da rede P2P (IP 10.1.1.2), enviou 1024 bytes endereçados ao IP 10.1.7.2, nó 6 da rede P2P, a partir da porta 49153.
Os bytes passaram pelo nó 0 (IP 10.1.1.1), seguiram até o nó 3 (IP 10.1.3.2) depois foram até o nó 6 (IP 10.1.7.2) onde seriam recebidos na porta 9 porém não há necessidade de passar esses bytes para outras camadas portanto eles são descartados (discard) e apenas retornados fazendo o mesmo caminho dos bytes de volta até o cliente, nó 1 (IP 10.1.1.2).

PCAPS

0-0:reading from file sim1_udp-0-0.pcap, link-type PPP (PPP)
-2:-59:-59.003686 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.018432 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024
.
.
.
6-0:reading from file sim1_udp-6-0.pcap, link-type PPP (PPP)
-2:-59:-59.011059 IP 10.1.1.2.49153 > 10.1.7.2.discard: UDP, length 1024
-2:-59:-59.011059 IP 10.1.7.2.discard > 10.1.1.2.49153: UDP, length 1024

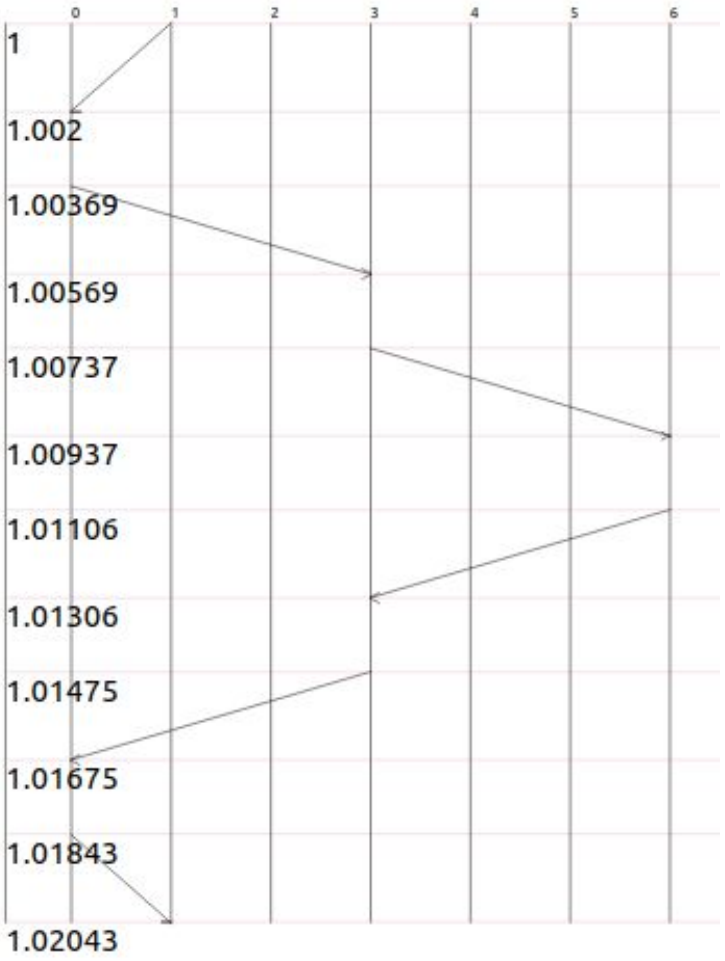

Figura 11 - representação do pacote viajando pela rede (gerado pelo NetAnim)

4.2 UDP com broadcast:
Log do terminal:
At time 1s client sent 1024 bytes to 10.200.1.4 port 9
At time 1.01829s server received 1024 bytes from 10.1.1.2 port 49153
At time 1.01829s server sent 1024 bytes to 10.1.1.2 port 49153
At time 1.04058s client received 1024 bytes from 10.200.1.4 port 9

O cliente, nó 1 da rede P2P, enviou 1024 bytes endereçados ao IP 10.200.1.4, nó 4 da rede broadcast (nó 6 na contagem total de nós).
Os bytes passaram pelo nó 0 (IP 10.1.1.1), seguiram até o nó 3 da rede P2P (IP 10.1.3.2).
O nó 3 também é o primeiro nó (nó 0) da rede broadcast (IP 10.200.1.0) e portanto consegue mandar os bytes até o destinatário nó 4 (IP 10.200.1.4) juntamente com todos os nós conectados nessa rede broadcast. Porém apenas o nó destinatário o recebe.
Depois de chegar até o nó 4, os 1024 bytes são retornados pelo mesmo caminho até o cliente que o enviou.

PCAPS:
0-0(P2P):reading from file sim5_udp_P2P-0-0.pcap, link-type PPP (PPP)

-2:-59:-59.003686 IP 10.1.1.2.49153 > 10.200.1.4.discard: UDP, length 1024
-2:-59:-59.036896 IP 10.200.1.4.discard > 10.1.1.2.49153: UDP, length 1024
.
.
.
3-0(P2P):reading from file sim5_udp_P2P-3-0.pcap, link-type PPP (PPP)
-2:-59:-59.007372 IP 10.1.1.2.49153 > 10.200.1.4.discard: UDP, length 1024
-2:-59:-59.033210 IP 10.200.1.4.discard > 10.1.1.2.49153: UDP, length 1024

arquivo sim5_udp_Csma-4-0:
reading from file sim5_udp_Csma-4-0.pcap, link-type EN10MB (Ethernet)
-2:-59:-59.012475 ARP, Request who-has 10.200.1.4 (Broadcast) tell 10.200.1.1, length 50
-2:-59:-59.014578 ARP, Reply 10.200.1.4 is-at 00:00:00:00:00:0c (oui Ethernet), length 50
-2:-59:-59.018291 IP 10.1.1.2.49153 > 10.200.1.4.discard: UDP, length 1024
-2:-59:-59.027393 ARP, Request who-has 10.200.1.1 (Broadcast) tell 10.200.1.4, length 50
-2:-59:-59.029497 ARP, Reply 10.200.1.1 is-at 00:00:00:00:00:09 (oui Ethernet), length 50
-2:-59:-59.033210 IP 10.200.1.4.discard > 10.1.1.2.49153: UDP, length 1024

- O nó 0 da rede broadcast (IP 10.200.1.0), que também é o nó 3 da rede P2P (IP 10.1.3.2), perguntou pelo nó 4 da rede broadcast (IP 10.200.1.4), que é justamente o destinatário.
- Obteve uma resposta.
- O nó cliente, n1 (IP 10.1.1.2) da rede P2P, enviou ao nó 3 (IP 10.1.3.2), passando pelo nó 0 (IP 10.1.1.1), 1024 bytes que foram repassados até o nó 4 da rede broadcast (IP 10.200.1.4), que é o destinatário, na porta 9. Porém o destinatário deve apenas retornar os mesmos bytes por isso o .discard no lugar da porta de entrada.
- Então o nó 4 da rede broadcast (IP 10.200.1.4) perguntou pelo nó 0 da rede broadcast (IP 10.200.1.0), que é o link entre as redes.
- Obteve uma resposta.
- O servidor, nó 4 broadcast (IP 10.200.1.4), enviou os 1024 bytes de volta para o cliente, nó 1 (IP 10.1.1.2), na porta 49153.

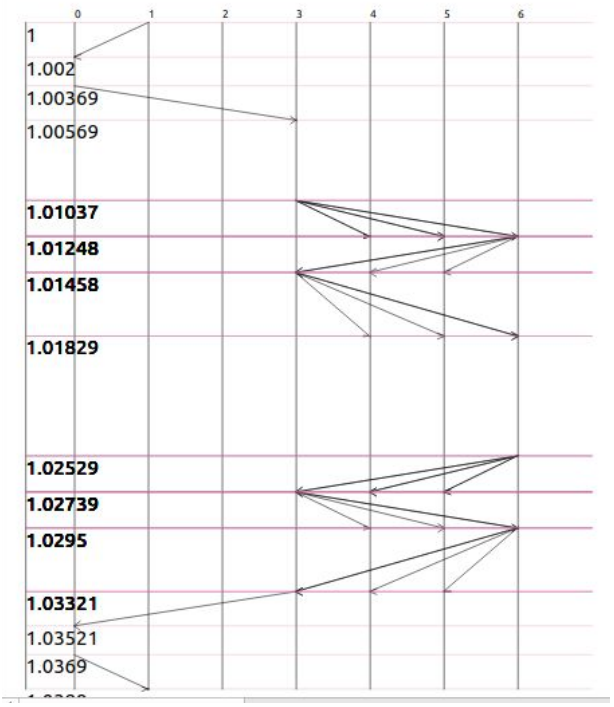

Figura 12 - representação do pacote viajando pela rede (gerado pelo NetAnim)

Concluímos que uma rede UDP sem broadcast, usando apenas conexões p2p, é mais rápida do que uma rede parcialmente implementada com broadcast, pelo menos quando não há muitos nós entre servidor e cliente fazendo que a rota seja ineficiente indo um por um. Há uma considerável diferença de tempo (3ms) entre quando o pacote sai do nó 3 para ir até o nó 6 nas duas versões, a rede com broadcast demora mais provavelmente porque essa é parte onde o broadcast começa e é preciso trocar para esse modelo de rede.

É importante notar que nossa primeira pretensão nesse cenário era simular os protocolos TCP e UDP em Broadcast, mas simular o TCP em broadcast acaba sendo algo muito estranho pois o feedback que o TCP precisa para ajustar sua janela não ocorre em broadcast (os nós que “escutam” poderiam retornar algum segmento ao remetente, mas isso causaria um cálculo errado do buffer de envio dos dados, que ainda ficaria diferente do buffer do destinatário). Vale lembrar que nas nossas simulações, o retorno que recebemos do UDP ocorre para identificarmos, essencialmente, o tempo no caminho de ida e volta de um pacote em uma rede, ou seja, é um retorno que nós definimos, mas que não é essencial pro protocolo UDP (no protocolo UDP o destinatário recebe e o remetente não tem essa garantia de recebimento, que somente os ACKs do protocolo TCP podem oferecer, ou algum protocolo específico com o mesmo objetivo).

6 -Observações

Nos apoiamos sobre os exemplos fornecidos pela ferramenta ns-3 principalmente para estudo sobre a ferramenta e nas duas referências de canais do youtube exibidas nas referências.

Vale observar que a rede broadcast do cenário 4 retornou um tipo de ping 3 vezes entre os nós 6 e 3 antes de retornar os bytes esperados (vide Figura 12) e nosso grupo não foi capaz de identificar o que esse ping representa.

7- Referências

AL-DHIEF, F. T. et al. **Performance Comparison between TCP and UDP Protocols in Different Simulation Scenarios**. 2018. International Journal of Engineering & Technology, 7 (4.36) (2018) 172-176. Disponível: https://www.sciencepubco.com/index.php/ijet/article/view/23739/11888. Acesso em: 26/10/2020.

CLOUDSHARK. **The TCP Timestamp Option.** Disponível em: https://cloudshark.io/articles/tcp-timestamp-option/. Acesso em: 26/10/2020

DEYAH, W; BAHYA, W. S. **Balancing between TCP and UDP to Improve Network Performance.** 2017. Journal of Engineering and Applied Sciences 12 (Special Issue 6). Disponível em: https://www.researchgate.net/publication/334442231_Balancing_between_TCP_and_UDP_to_Improve_Network_Performance. Acesso em: 26/10/2020.

GEANT. **TCP Selective Acknowledgements (SACK)**. 2019. Disponível em: https://wiki.geant.org/display/public/EK/SelectiveAcknowledgements. Acesso em: 26/10/2020

GIGATUX. **Examining tcpdump Output**. Disponível em: http://books.gigatux.nl/mirror/snortids/0596006616/snortids-CHP-2-SECT-6.html. Acesso em: 26/10/2020.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet, Uma abordagem top-down**". 6ª edição, 2014.

NSAM. **Building Topologies** 2020. Disponível em: https://www.nsnam.org/doxygen/dynamic-global-routing_8cc_source.html. Acesso em: 26/10/2020.

NSAM. **dynamic-global-routing.cc.** 2020. Disponível em: https://www.nsnam.org/docs/tutorial/html/building-topologies.html. Acesso em: 26/10/2020.

NSAM. **ns3::UdpClientHelper Class Reference.** 2020.. Disponível em: https://www.nsnam.org/doxygen/dynamic-global-routing_8cc_source.html. Acesso em: 26/10/2020.

STACK OVERFLOW. **TCP compatibility: Why is TCP not compatible with packet broadcast and multicasting actions?** 2011. Disponível em: https://stackoverflow.com/questions/4823739/tcp-compatibility-why-is-tcp-not-compatible-with-packet-broadcast-and-multicast. Acesso em: 26/10/2020.

YOUTUBE. **Canal Adil Alsuhaim**. Disponível em: https://www.youtube.com/c/AdilAlsuhaim. Acesso em: 26/10/2020

YOUTUBE. **NS 3 - Simulação de uma inter-rede P2P e CSMA**. 2020. Canal Edivaldo Pastori Valentini. Disponível em: https://youtu.be/kicshIuptGg. Acesso em: 26/10/2020