

Relatório Trabalho Prático nº2

João Pimentel (a80874)

Rodolfo Silva (a81716)

Pedro Gonçalves (a82313)

Novembro 2018

Universidade do Minho
Redes de Computadores
Grupo 64

Conteúdo

1	Questões e Respostas	3
1.1	Parte I	3
1.2	Parte II	6
2	Conclusões	9
3	Anexos	10

1 Questões e Respostas

1.1 Parte I

1. Prepare uma topologia CORE para verificar o comportamento do traceroute. Ligue um host(pc) h1 a um router r2; o router r2 a um router r3, que por sua vez, se liga a um host(servidor) s4. (Note que pode não existir conectividade IP imediata entre h1 e s4 até que o routing estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado

- (a) Active o *wireshark* ou o *tcdump* no pc h1. Numa *shell* de h1, execute o comando *traceroute -I* para o endereço IP do *host* s4.

Ver figura 1 da secção dos Anexos

- (b) Registe e analise o tráfego ICMP enviado por h1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

O Source IP vai enviando pings com TTL crescentes até conseguir obter resposta do IP de destino.

Ver figura 2 da secção dos Anexos para o registo de algumas das diferentes tentativas para obter resposta.

- (c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino s4? Verifique na prática que a sua resposta está correta.

O TTL deverá ter valor 3, pois é o número de *hops* necessários para chegar de *h1* até *s4*.

Como podemos ver na figura 2 a primeira resposta existe quando TTL = 3.

- (d) Qual o valor médio do tempo de ida-e-volta (Round-Trip Time) obtido?

Pedido	▼	Resposta	▼	Diferença	▼		Média
2,606373314		2,606403394		3,008E-05			4,87478E-05
2,606419772		2,606449014		2,9242E-05			
2,607732969		2,607791163		5,8194E-05			
2,607859295		2,607919765		6,047E-05			
2,608007503		2,608073256		6,5753E-05			

Ver Figura 3 em anexos.

2. Pretende-se agora usar o traceroute na sua máquina nativa, e gerar datagramas IP de diferentes tamanhos.

Com base no tráfego capturado, identifique os pedidos ICMP *Echo Request* e o conjunto de mensagens devolvidas em resposta a esses pedidos.

Selecione a primeira mensagem ICMP capturada (referente a (i) tamanho por defeito) e centre a análise no nível protocolar IP (expanda o *tab* correspondente na janela de detalhe do *wireshark*). Através da análise do cabeçalho IP diga:

- (a) Qual o endereço IP da interface ativa do seu computador?

O endereço IP da interface ativa foi 192.168.2.186, uma vez que é o ponto de partida do *request* para o servidor, como mostra a figura.

No.	Time	Source	Destination	Protocol
32	3.762557505	192.168.2.1	192.168.2.186	ICMP
33	3.762594940	192.168.2.1	192.168.2.186	ICMP
34	3.762603530	192.168.2.1	192.168.2.186	ICMP
16	3.761951629	192.168.2.186	193.136.9.240	ICMP
17	3.761969841	192.168.2.186	193.136.9.240	ICMP
18	3.761991778	192.168.2.186	193.136.9.240	ICMP
19	3.762001420	192.168.2.186	193.136.9.240	ICMP
20	3.762008716	192.168.2.186	193.136.9.240	ICMP
21	3.762016009	192.168.2.186	193.136.9.240	ICMP
22	3.762024616	192.168.2.186	193.136.9.240	ICMP
23	3.762032150	192.168.2.186	193.136.9.240	ICMP

- (b) Qual é o valor do campo protocolo? O que identifica?

O valor do campo protocolo é ICMP(1), significando que está a ser transmitido tráfego *Internet Control Message Protocol*. Ver Figura 4 dos Anexos.

- (c) Quantos *bytes* tem o cabeçalho IP(v4)? Quantos *bytes* tem o campo de dados(*payload*) do datagram? Como se calcula o tamanho do *payload*?

O cabeçalho do IPv4 possui 20 bytes (Figura 6). Sabendo que o payload é calculado subtraindo o tamanho do cabeçalho ao comprimento total do datagrama, como o tamanho total da mensagem é 60 bytes (Figura 5), tem-se que o payload do datagrama possui 40 bytes.

- (d) O datagram IP foi fragmentado? Justifique.

Não, pois a Flag de Fragmentação apresentava o valor 0 e não possui offset de fragmentação. Ver Figura 5.

- (e) Ordene os pacotes capturados de acordo com o endereço IP fonte(e.g., selecionando o cabeçalho da coluna *Source*), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote.

Variam os campos *TTL*, *Header Checksum* e o identificador, como mostra a figura.

▶ Ethernet II, Src: AsustekC_10:f3:b1 (74:d0:2b:10:f3:b1), Dst: Vmw... ▼ Internet Protocol Version 4, Src: 192.168.2.186, Dst: 193.136.9.240 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x646a (25706) Flags: 0x0000 Time to live: 5 Protocol: ICMP (1) Header checksum: 0xc27c [validation disabled] [Header checksum status: Unverified] Source: 192.168.2.186 Destination: 193.136.9.240 ▶ Internet Control Message Protocol	▶ Ethernet II, Src: AsustekC_10:f3:b1 (74:d0:2b:10:f3:b1), Dst: Vmw... ▼ Internet Protocol Version 4, Src: 192.168.2.186, Dst: 193.136.9.240 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x645f (25695) Flags: 0x0000 Time to live: 2 Protocol: ICMP (1) Header checksum: 0xc587 [validation disabled] [Header checksum status: Unverified] Source: 192.168.2.186 Destination: 193.136.9.240 ▶ Internet Control Message Protocol
--	--

- (f) **Observa algum padrão nos valores do campo de Identificação do datagram IP e TTL?**

TTL é incrementada na tentativa de chegar ao destino e identificador é incrementado numa unidade por cada datagrama, consoante o momento de criação.

- (g) **Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL *exceeded* enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL *exceeded* enviados ao seu *host*? Porquê?**

O valor de *TTL* nas respostas *ICMP TTL exceeded* é de 64, permanecendo constante, pois é o *TTL* estabelecido pelo fabricante do *router* para este tipo de respostas. No momento em que o número de *hops* chega a zero, é enviada uma mensagem a avisar do acontecimento à *source*, sempre com um *TTL* alto de modo a chegar ao destino.

3. Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego *depois do tamanho de pacote ter sido definido para 35XX bytes*

- (a) **Localize a primeira mensagem ICMP. Porque é que houve necessidade de fragmentar o pacote inicial?**

Foi necessário fragmentar o pacote a enviar pois o tamanho do pacote a enviar é maior do que o tamanho que o *payload* suporta.

- (b) **Imprima o primeiro fragmento do datagrama IP segmentado. Que informação no cabeçalho indica que o datagrama foi fragmentado? Que informação no cabeçalho IP indica que se trata do primeiro fragmento? Qual é o tamanho deste datagrama IP?**

A primeira mensagem fragmentada é caracterizada por ter a *flag* de *More Fragments* a 1 e não possuir *offset* de fragmentação. O seu tamanho é 1500 bytes. Ver Figura 6 nos Anexos.

- (c) **Imprima o segundo fragmento do datagrama IP original. Que informação do cabeçalho IP indica que não se trata do 1º fragmento? Há mais fragmentos? O que nos permite afirmar isso?**

Não se trata do primeiro fragmento do datagrama pois possui *Fragment offset* maior que zero. Existem mais fragmentos, pois a *Flag More Fragments* tem o valor 1. Ver Figura 7 nos Anexos.

- (d) **Quantos fragmentos foram criados a partir do datagrama original? Como se detecta o último fragmento correspondente ao datagram original?**

Foram criados 3 fragmentos a partir do datagrama original (*IPv4 fragment #27(1480) #28(1480) #29(584)*). É possível detetar o último fragmento pois este possui *Fragment Offset* e tem a *Flag More Fragment* a zero. Ver Figura 8 nos Anexos.

- (e) **Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.**

Os campos que são alterados no cabeçalho *IP* entre os diferentes fragmentos são a *Flag More Fragment* e o valor de *Fragment Offset*. O primeiro datagrama possui *More Fragments*, mas o seu *Fragment Offset* tem valor zero. Já o último fragmento não tem *More Fragments*, mas o seu *offset* é o mais elevado. Os intermédios são caracterizados por possuírem *More Fragments* e *Fragment Offset* sucessivamente mais elevado, consoante a sua posição como fragmento do datagrama.

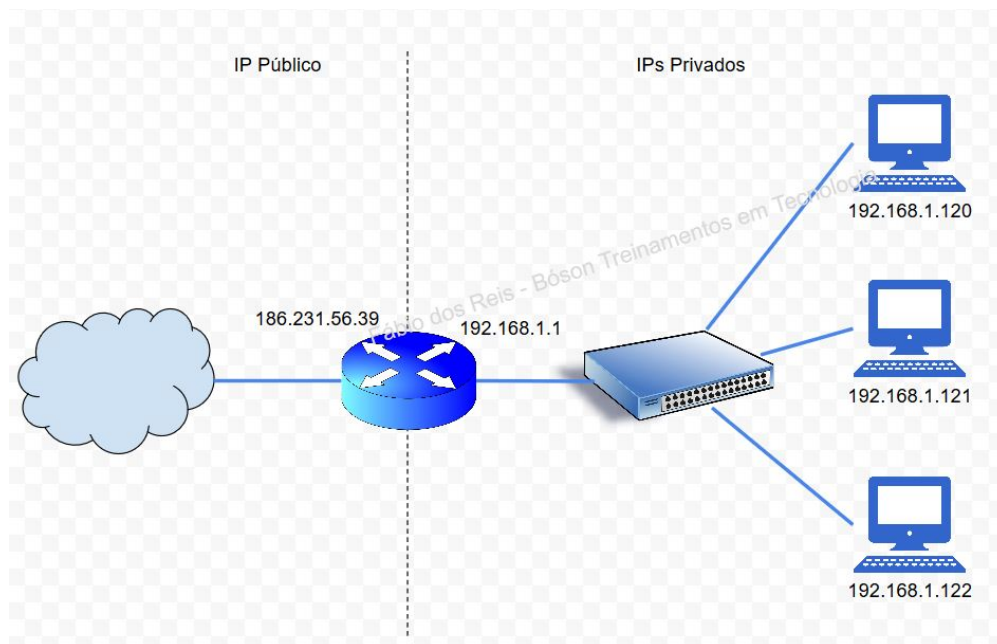
1.2 Parte II

4. Considere que a organização MIEI-RC é constituída por três departamentos(A, B e C) e cada departamento possui um *router* de acesso à sua rede local. Estes *routers* de acesso(Ra, Rb e Rc) estão interligados entre si por ligações Ethernet a 1Gbps, formando um anel. Por sua vez, existe um servidor(S1) na rede do departamento C e, pelo menos, três *laptops* por departamento, interligados ao *router* respectivo através de um comutador(*switch*). S1 tem uma ligação a 1Gbps e os *laptops* ligações a 100Mbps. Considere apenas a existência de um comutador por departamento.

A conectividade IP externa da organização é assegurada através de um router de acesso Rext conectado a Rc por uma ligação ponto-a-ponto a 10Gbps.

Construa uma topologia CORE que reflita a rede local da empresa. Para facilitar a visualização pode ocultar o endereçamento IPv6.

1. Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos da topologia.
 - (a) Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustre de forma clara a topologia definida e o endereçamento usado.
Ver Figura 9 e Figura 10 em Anexos.
 - (b) **Tratam-se de endereços públicos ou privados? Porquê?**
Privados, pois estamos a falar de uma rede local. Além disso, os endereços públicos existem quando falamos da *Internet*, sendo atribuídos pela IANA (*Internet Assigned Numbers Authority*), como mostra a figura.



- (c) **Porque razão não é atribuído um endereço IP aos *switches*?**
Um *Switch* não possui endereço *IP*, pois trabalha no nível de rede 2, não sendo necessário o acesso à parte *IP* dos pacotes *ethernet*, operando apenas sobre o seu *MAC address*.

- (d) Usando o comando *ping* certifique-se que existe conectividade entre os *laptops* dos vários departamentos e o servidor do departamento C(basta certificar-se da conectividade de um *laptop* por departamento).

Sabendo que a existência de conectividade implica a existência de um caminho de ida e volta, as Figuras 11, 12 e 13 demonstram que tal acontece.

- (e) Verifique se existe conectividade IP do router de acesso Rext para o servidor S1.

Existe ligação, como mostra a Figura 14 nos Anexos.

2. Para o *router* e um *laptop* do departamento A:

- (a) Execute o comando `netstat -rn` por forma a poder consultar a tabela de encaminhamento unicast (IPv4). Inclua no seu relatório as tabelas de encaminhamento obtidas; interprete as várias entradas de cada tabela. Se necessário, consulte o manual respetivo (`man netstat`).

Ver Figura 15 em Anexos.

Tenha-se em consideração o seguinte *Flag Description*:

U Up—Route is valid

G Gateway—Route is to a gateway router rather than to a directly connected network or host

H Host name—Route is to a host rather than to a network, where the destination address is a complete address

R Reject—Set by ARP when an entry expires (for example, the IP address could not be resolved into a MAC address)

D Dynamic—Route added by a route redirect or RIP (if routed is enabled)

M Modified—Route modified by a route redirect

C Cloning—A new route is cloned from this entry when it is used

L Link—Link-level information, such as the Ethernet MAC address, is present

S Static—Route added with the route command

A tabela mostra as ligações existentes no sistema, sendo que retratam o caminho a direccionar (*gateway*) o datagrama quando este possui um determinado destino. As que possuem a *Flag U* indicam que a rota está disponível. As que possuem um *G* representam ligações por um *gateway*. Além disso, são apresentadas as máscaras de rede de cada destino. O parâmetro *MSS* é o tamanho máximo do segmento, ou seja, o maior datagrama que o *kernel* construirá para transmitir na rota em questão. A *Window* é a quantidade máxima de dados que o sistema vai aceitar num único *burst* do *host* remoto. *Irtt* indica o *round trip time* inicial.

Por fim, o campo *Iface* indica a interface de rede que a rota usará, sendo *eth0*, para o caso de ser dentro da própria rede e *ethN* (*N* != 0) para o uso de *gateways*.

- (b) Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema).

Sabendo que se deve utilizar roteamento dinâmico quando existe mais do que uma rota possível para o mesmo ponto, está a ser utilizado encaminhamento dinâmico no anel dos *routers*. Já a nível interno dos Departamentos, o escalonamento é estático entre os *PCs* e o *router* do mesmo.

- (c) Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou *default*) deve ser retirada definitivamente da tabela de encaminhamento do servidor S1 localizado no departamento C. Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor. Justifique.

Apagar o *default* implica que tudo o que não está definido é perdido, ou seja toda a conectividade que não esteja dentro da interface de rede do Departamento C é perdida.

- (d) **Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S1, por forma a contornar a restrição imposta na alínea (c). Utilize para o efeito o comando `route add` e registre os comandos que usou.**

Relembrando que conectividade é possuir um caminho de ida e volta e, sendo pedido que se re-estabeleça a conectividade com os departamentos A e B, são usadas as interfaces de rede de cada departamento como destino.

Assim, foram efetuados os comandos seguintes:

```
route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.5.1.  
route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.5.1.
```

- (e) **Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível, utilizando para o efeito o comando `ping`. Registe a nova tabela de encaminhamento do servidor.**

Ver Figuras 16, 17 e 18 para confirmar acessibilidade com o servidor.

A nova Tabela é visível na Figura 16, no topo.

5. **Considere a topologia definida anteriormente. Assuma que o endereçamento entre os routers se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.**

1. **Considere que dispõe apenas do endereço de rede IP 172.xx.48.0/20, em que XX é o decimal correspondendo ao seu número de grupo (PLXX). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo a rede de acesso e core inalteradas) e atribua endereços às interfaces dos vários sistemas envolvidos. Deve justificar as opções usadas**

Usando 4 bits, de forma a ser possível gerar 16 sub-redes (Desde 172.64.48.0/24 a 172.64.63.0/24), sendo que 2 delas serão reservadas para endereçamento (172.64.48.0/24) e *broadcast* (172.64.63.0/24), tem-se que a máscara passa de /20 para /24. De notar que existem 14 sub-redes possíveis de atribuir. Assim, associando as três primeiras sub-redes aos departamentos, tem-se:

Departamento A 172.64.49/24 (SR1)

Departamento B 172.64.50/24 (SR2)

Departamento C 172.64.51/24 (SR3)

2. **Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Justifique.**

Máscara de rede = /24

Hosts por Departamento = $2^8 - 2 = 254$ (dois endereços reservados por Departamento)

3. **Garanta e verifique que conectividade IP entre as várias redes locais da organização MIEI-RC é mantida. Explique como procedeu.**

Utilizando valores entre 1 e 9 para representar *Routers*, 10 a 19 para *servidores* e os restantes para *hosts*.

Para garantir a conectividade entre as várias redes locais, teve que se garantir a existência de um caminho de ida e volta entre cada departamento com os outros dois e consigo mesmo. Assim, foi necessário executar vários *pings* de X para Y e, sucessivamente, de Y para X. Vejam-se as Figuras 19, 20 e 21, que exemplificam um pouco do processo.

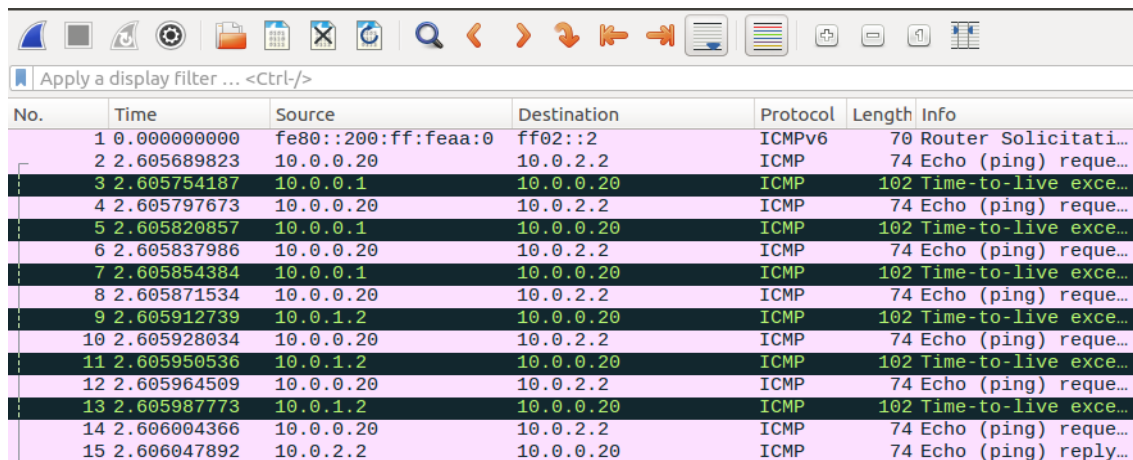
2 Conclusões

O desenvolvimento deste projeto permitiu um aumento no conhecimento relativamente a redes *IPv4*, bem como a *sub-netting*.

Sabendo que um bom conhecimento do funcionamento de uma rede à qual a nossa máquina está ligada permite resolver problemas com alguma facilidade, este trabalho veio comprovar isso mesmo. Além disso, permitiu a perceção de como um datagrama é direccionado até atingir o seu destino e as restrições que a falta de rotas pode causar no que toca à receção dos mesmos.

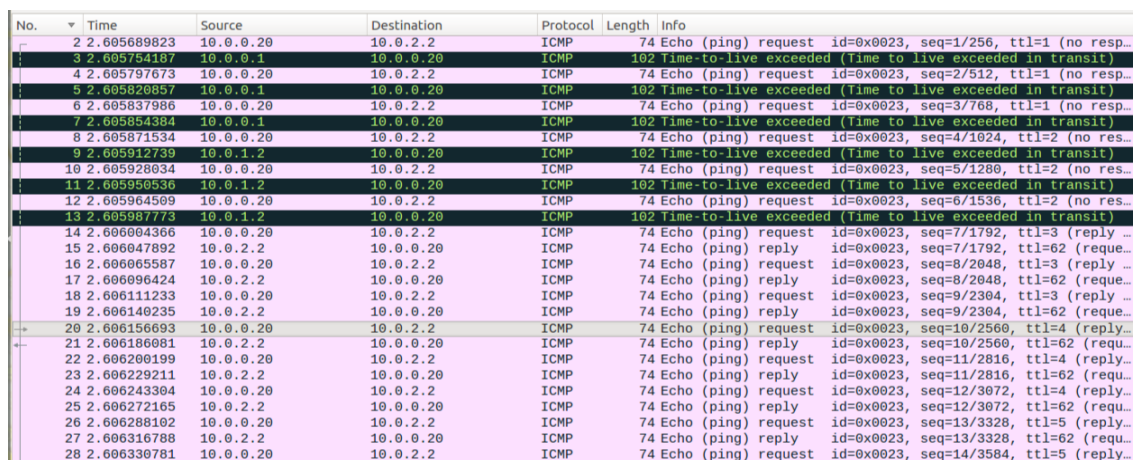
Em suma, este projeto permitiu solidificar a matéria aprendida nas aulas teóricas, tendo sido bastante proveitoso para a possível realização de qualquer trabalho futuro relativo a temas próximos do mesmo.

3 Anexos



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::200:ff:feaa:0	ff02::2	ICMPv6	70	Router Solicitati...
2	2.605689823	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
3	2.605754187	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exce...
4	2.605797673	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
5	2.605820857	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exce...
6	2.605837986	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
7	2.605854384	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exce...
8	2.605871534	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
9	2.605912739	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exce...
10	2.605928034	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
11	2.605950536	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exce...
12	2.605964509	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
13	2.605987773	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exce...
14	2.606004366	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) reque...
15	2.606047892	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply...

Figura 1 - Execução do comando *traceroute -I*



No.	Time	Source	Destination	Protocol	Length	Info
2	2.605689823	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=1/256, ttl=1 (no resp...
3	2.605754187	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
4	2.605797673	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=2/512, ttl=1 (no resp...
5	2.605820857	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
6	2.605837986	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=3/768, ttl=1 (no resp...
7	2.605854384	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
8	2.605871534	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=4/1024, ttl=1 (no res...
9	2.605912739	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10	2.605928034	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=5/1280, ttl=2 (no res...
11	2.605950536	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
12	2.605964509	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=6/1536, ttl=2 (no res...
13	2.605987773	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
14	2.606004366	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=7/1792, ttl=3 (reply ...
15	2.606047892	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=7/1792, ttl=62 (reque...
16	2.606065587	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=8/2048, ttl=3 (reply ...
17	2.606096424	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=8/2048, ttl=62 (reque...
18	2.606111233	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=9/2304, ttl=3 (reply ...
19	2.606140235	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=9/2304, ttl=62 (reque...
20	2.606156693	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=10/2560, ttl=4 (reply...
21	2.606186081	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=10/2560, ttl=62 (reque...
22	2.606206199	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=11/2816, ttl=4 (reply...
23	2.606229211	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=11/2816, ttl=62 (reque...
24	2.606243364	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=12/3072, ttl=4 (reply...
25	2.606272165	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=12/3072, ttl=62 (reque...
26	2.606288102	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=13/3328, ttl=5 (reply...
27	2.606316788	10.0.2.2	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0023, seq=13/3328, ttl=62 (reque...
28	2.606330781	10.0.0.20	10.0.2.2	ICMP	74	Echo (ping) request id=0x0023, seq=14/3584, ttl=5 (reply...

Figura 2 -

2.606373314
2.606403394
2.606419772
2.606449014
2.607732969
2.607791163
2.607859295
2.607919765
2.608007503
2.608073256

Figura 3 - Tempos de resposta ao ping efetuado por *h1* até *s4*

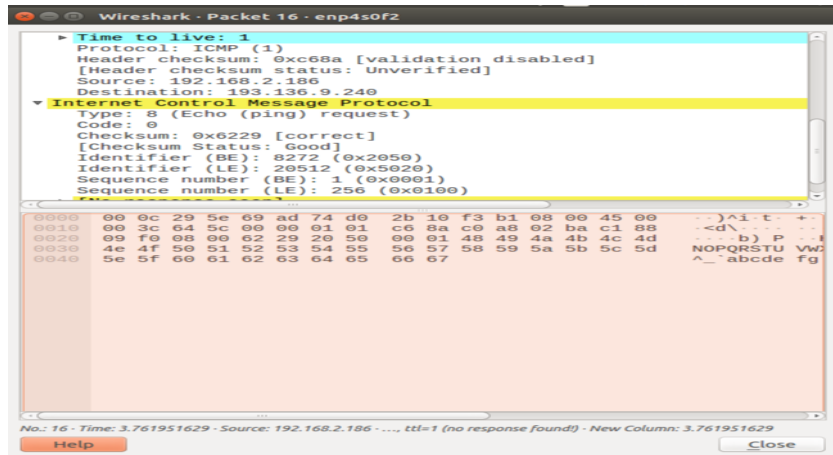


Figura 4

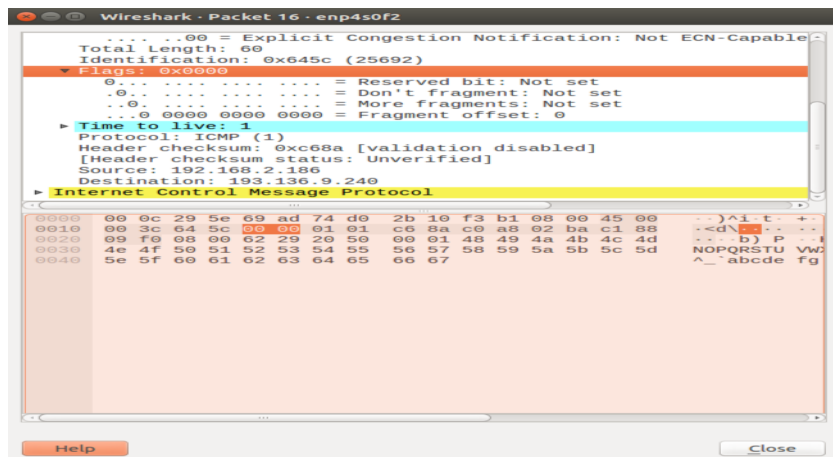


Figura 5 - Flag de Fragmentação

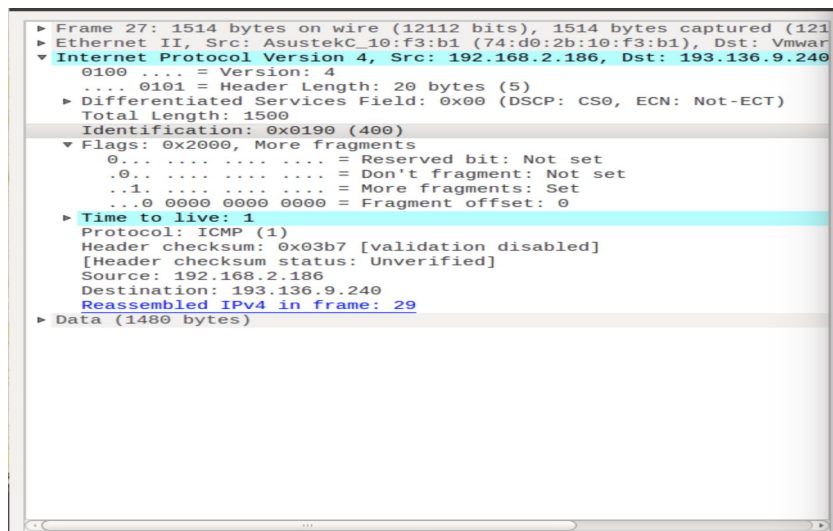


Figura 6

```

▶ Frame 28: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
▶ Ethernet II, Src: AsustekC_10:f3:b1 (74:d0:2b:10:f3:b1), Dst: Vmwar_08:00:27:00:00:00
▼ Internet Protocol Version 4, Src: 192.168.2.186, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x0190 (400)
  ▼ Flags: 0x20b9, More fragments
    0... .. = Reserved bit: Not set
    .0... .. = Don't fragment: Not set
    ..1... .. = More fragments: Set
    ...0 0000 1011 1001 = Fragment offset: 185
  ▶ Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x02fe [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.2.186
  Destination: 193.136.9.240
  Reassembled IPv4 in frame: 29
▶ Data (1480 bytes)

```

Figura 7

```

▶ Frame 29: 618 bytes on wire (4944 bits), 618 bytes captured (4944 bits) on interface 0
▶ Ethernet II, Src: AsustekC_10:f3:b1 (74:d0:2b:10:f3:b1), Dst: Vmware_5e:09:ad (00:0c:29:5e:09:ad)
▼ Internet Protocol Version 4, Src: 192.168.2.186, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 604
    Identification: 0x0190 (400)
  ▼ Flags: 0x0172
    0... .. = Reserved bit: Not set
    .0... .. = Don't fragment: Not set
    ..0... .. = More fragments: Not set
    ...0 0001 0111 0010 = Fragment offset: 370
  ▶ Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x25c5 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.2.186
  Destination: 193.136.9.240
  ▶ [3 IPv4 Fragments (3544 bytes): #27(1480), #28(1480), #29(584)]
▶ Internet Control Message Protocol

```

Figura 8

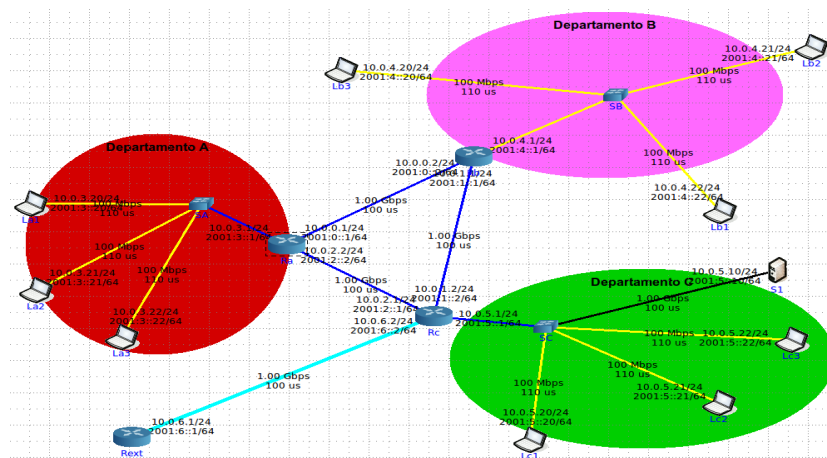


Figura 9 - Topologia Core Exercício 1 Parte 2

Nome	Ip	Máscara de rede
La1	10.0.3.20/24	255.255.255.0
La2	10.0.3.21/24	255.255.255.0
La3	10.0.3.22/24	255.255.255.0
Lb1	10.0.4.22/24	255.255.255.0
Lb2	10.0.4.21/24	255.255.255.0
Lb3	10.0.4.20/24	255.255.255.0
Lc1	10.0.5.20/24	255.255.255.0
Lc2	10.0.5.21/24	255.255.255.0
Lc3	10.0.5.22/24	255.255.255.0
S1	10.0.5.10/24	255.255.255.0
Ra	10.0.3.1/24	255.255.255.0
Ra->b	10.0.0.1/24	255.255.255.0
Ra->c	10.0.2.2/24	255.255.255.0
Rb	10.0.4.1/24	255.255.255.0
Rb->a	10.0.0.2/24	255.255.255.0
Rb->c	10.0.1.1/24	255.255.255.0
Rc	10.0.5.1/24	255.255.255.0
Rc->a	10.0.2.1/24	255.255.255.0
Rc->b	10.0.1.2/24	255.255.255.0
Rc->ext	10.0.6.2/24	255.255.255.0
Rext->c	10.0.0.1/24	255.255.255.0

Figura 10

```

root@La1:/tmp/pycore.37671/La1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.625 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.618 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.620 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.591 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.622 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=62 time=0.623 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=62 time=0.618 ms
64 bytes from 10.0.5.10: icmp_seq=8 ttl=62 time=0.617 ms
64 bytes from 10.0.5.10: icmp_seq=9 ttl=62 time=0.618 ms

```

```

root@S1:/tmp/pycore.34031/S1.conf# ping 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.
64 bytes from 10.0.3.20: icmp_seq=1 ttl=62 time=0.957 ms
64 bytes from 10.0.3.20: icmp_seq=2 ttl=62 time=0.622 ms
64 bytes from 10.0.3.20: icmp_seq=3 ttl=62 time=0.622 ms
64 bytes from 10.0.3.20: icmp_seq=4 ttl=62 time=0.618 ms
64 bytes from 10.0.3.20: icmp_seq=5 ttl=62 time=0.617 ms
64 bytes from 10.0.3.20: icmp_seq=6 ttl=62 time=0.617 ms
64 bytes from 10.0.3.20: icmp_seq=7 ttl=62 time=0.581 ms
64 bytes from 10.0.3.20: icmp_seq=8 ttl=62 time=0.620 ms
64 bytes from 10.0.3.20: icmp_seq=9 ttl=62 time=0.614 ms

```

Figura 11 - Ping entre Departamento A e o servidor S1 e vice-versa

```

root@Lb2:/tmp/pycore.37671/Lb2.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.601 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.616 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.591 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.537 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.616 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=62 time=0.549 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=62 time=0.587 ms
64 bytes from 10.0.5.10: icmp_seq=8 ttl=62 time=0.623 ms

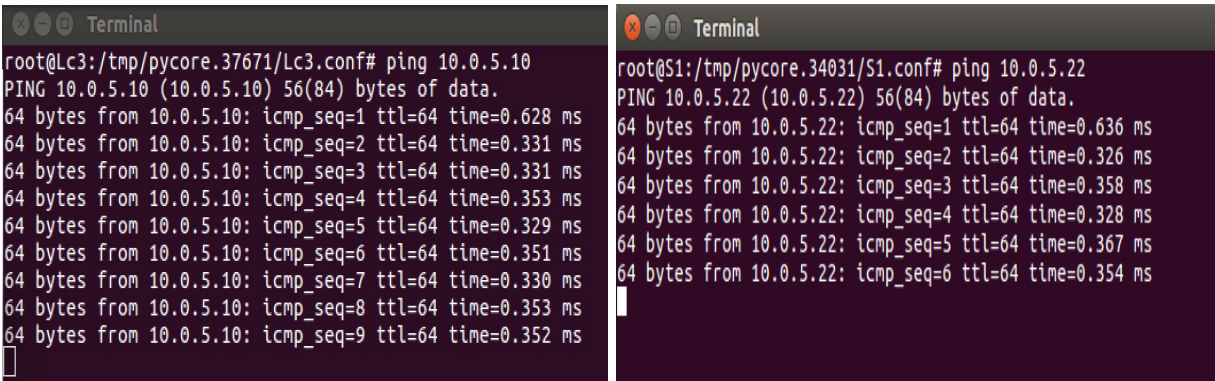
```

```

root@S1:/tmp/pycore.34031/S1.conf# ping 10.0.4.22
PING 10.0.4.22 (10.0.4.22) 56(84) bytes of data.
64 bytes from 10.0.4.22: icmp_seq=1 ttl=62 time=0.799 ms
64 bytes from 10.0.4.22: icmp_seq=2 ttl=62 time=0.623 ms
64 bytes from 10.0.4.22: icmp_seq=3 ttl=62 time=0.622 ms
64 bytes from 10.0.4.22: icmp_seq=4 ttl=62 time=0.620 ms
64 bytes from 10.0.4.22: icmp_seq=5 ttl=62 time=0.624 ms
64 bytes from 10.0.4.22: icmp_seq=6 ttl=62 time=0.667 ms
64 bytes from 10.0.4.22: icmp_seq=7 ttl=62 time=0.626 ms
64 bytes from 10.0.4.22: icmp_seq=8 ttl=62 time=0.621 ms

```

Figura 12 - Ping entre Departamento B e o servidor S1 e vice-versa

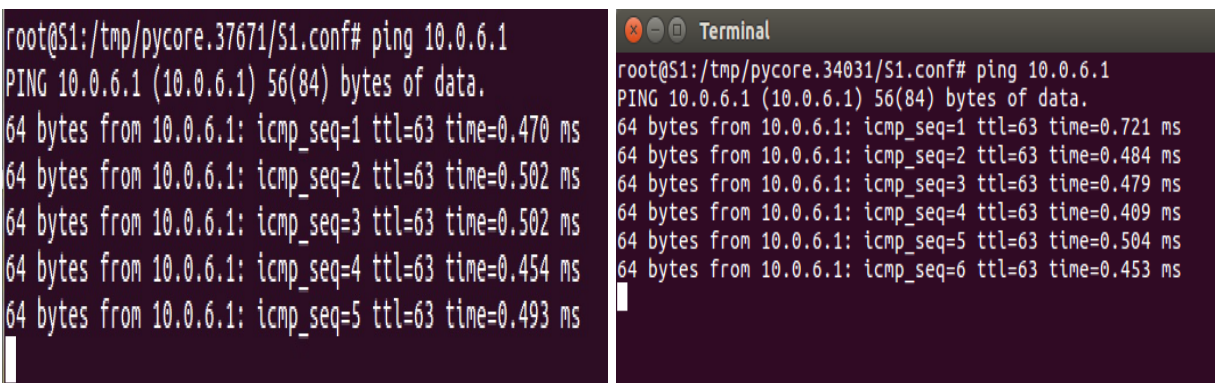


The image shows two terminal windows side-by-side. The left window is titled 'Terminal' and shows a ping command from 'root@Lc3:/tmp/pycore.37671/Lc3.conf#' to '10.0.5.10'. It displays 9 successful ping results with varying times between 0.328 ms and 0.628 ms. The right window is also titled 'Terminal' and shows a ping command from 'root@S1:/tmp/pycore.34031/S1.conf#' to '10.0.5.22'. It displays 6 successful ping results with times between 0.326 ms and 0.636 ms.

```
root@Lc3:/tmp/pycore.37671/Lc3.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=64 time=0.628 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.331 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=0.331 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.353 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.329 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=64 time=0.351 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=64 time=0.330 ms
64 bytes from 10.0.5.10: icmp_seq=8 ttl=64 time=0.353 ms
64 bytes from 10.0.5.10: icmp_seq=9 ttl=64 time=0.352 ms

root@S1:/tmp/pycore.34031/S1.conf# ping 10.0.5.22
PING 10.0.5.22 (10.0.5.22) 56(84) bytes of data.
64 bytes from 10.0.5.22: icmp_seq=1 ttl=64 time=0.636 ms
64 bytes from 10.0.5.22: icmp_seq=2 ttl=64 time=0.326 ms
64 bytes from 10.0.5.22: icmp_seq=3 ttl=64 time=0.358 ms
64 bytes from 10.0.5.22: icmp_seq=4 ttl=64 time=0.328 ms
64 bytes from 10.0.5.22: icmp_seq=5 ttl=64 time=0.367 ms
64 bytes from 10.0.5.22: icmp_seq=6 ttl=64 time=0.354 ms
```

Figura 13 - Ping entre Departamento C e o servidor S1 e vice-versa

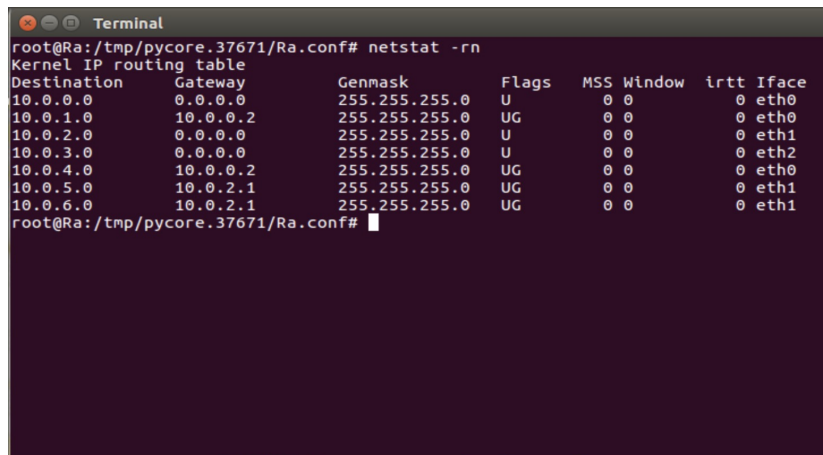


The image shows two terminal windows side-by-side. The left window is titled 'Terminal' and shows a ping command from 'root@S1:/tmp/pycore.37671/S1.conf#' to '10.0.6.1'. It displays 5 successful ping results with times between 0.470 ms and 0.502 ms. The right window is also titled 'Terminal' and shows a ping command from 'root@S1:/tmp/pycore.34031/S1.conf#' to '10.0.6.1'. It displays 6 successful ping results with times between 0.479 ms and 0.721 ms.

```
root@S1:/tmp/pycore.37671/S1.conf# ping 10.0.6.1
PING 10.0.6.1 (10.0.6.1) 56(84) bytes of data.
64 bytes from 10.0.6.1: icmp_seq=1 ttl=63 time=0.470 ms
64 bytes from 10.0.6.1: icmp_seq=2 ttl=63 time=0.502 ms
64 bytes from 10.0.6.1: icmp_seq=3 ttl=63 time=0.502 ms
64 bytes from 10.0.6.1: icmp_seq=4 ttl=63 time=0.454 ms
64 bytes from 10.0.6.1: icmp_seq=5 ttl=63 time=0.493 ms

root@S1:/tmp/pycore.34031/S1.conf# ping 10.0.6.1
PING 10.0.6.1 (10.0.6.1) 56(84) bytes of data.
64 bytes from 10.0.6.1: icmp_seq=1 ttl=63 time=0.721 ms
64 bytes from 10.0.6.1: icmp_seq=2 ttl=63 time=0.484 ms
64 bytes from 10.0.6.1: icmp_seq=3 ttl=63 time=0.479 ms
64 bytes from 10.0.6.1: icmp_seq=4 ttl=63 time=0.409 ms
64 bytes from 10.0.6.1: icmp_seq=5 ttl=63 time=0.504 ms
64 bytes from 10.0.6.1: icmp_seq=6 ttl=63 time=0.453 ms
```

Figura 14 - Ping entre Router de Acesso e o servidor S1 e vice-versa



The image shows a terminal window titled 'Terminal' with the command 'netstat -rn' executed. The output displays the kernel IP routing table with columns for Destination, Gateway, Genmask, Flags, MSS, Window, irtt, and Iface. The routing table shows entries for various destinations, including 10.0.0.0, 10.0.1.0, 10.0.2.0, 10.0.3.0, 10.0.4.0, 10.0.5.0, and 10.0.6.0, all pointing to different gateways and interfaces.

```
root@Ra:/tmp/pycore.37671/Ra.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.1.0 10.0.0.2 255.255.255.0 UG 0 0 0 eth0
10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.4.0 10.0.0.2 255.255.255.0 UG 0 0 0 eth0
10.0.5.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth1
10.0.6.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth1
```

Figura 15 - Netsat -rn

```
Terminal
-i1.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.5.1
root@S1:/tmp/pycore.44023/S1.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.5.1
root@S1:/tmp/pycore.44023/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.3.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
10.0.4.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
10.0.5.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1:/tmp/pycore.44023/S1.conf# ping 10.0.3.22
PING 10.0.3.22 (10.0.3.22) 56(84) bytes of data:
64 bytes from 10.0.3.22: icmp_seq=1 ttl=62 time=0.793 ms
64 bytes from 10.0.3.22: icmp_seq=2 ttl=62 time=0.553 ms
64 bytes from 10.0.3.22: icmp_seq=3 ttl=62 time=0.583 ms
64 bytes from 10.0.3.22: icmp_seq=4 ttl=62 time=0.622 ms
64 bytes from 10.0.3.22: icmp_seq=5 ttl=62 time=0.630 ms
64 bytes from 10.0.3.22: icmp_seq=6 ttl=62 time=0.608 ms
64 bytes from 10.0.3.22: icmp_seq=7 ttl=62 time=0.616 ms
64 bytes from 10.0.3.22: icmp_seq=8 ttl=62 time=0.617 ms
^C
--- 10.0.3.22 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7170ms
rtt min/avg/max/mdev = 0.553/0.628/0.793/0.073 ms
root@S1:/tmp/pycore.44023/S1.conf# ping 10.0.4.22
PING 10.0.4.22 (10.0.4.22) 56(84) bytes of data:
64 bytes from 10.0.4.22: icmp_seq=1 ttl=62 time=0.839 ms
64 bytes from 10.0.4.22: icmp_seq=2 ttl=62 time=0.650 ms
64 bytes from 10.0.4.22: icmp_seq=3 ttl=62 time=0.663 ms
64 bytes from 10.0.4.22: icmp_seq=4 ttl=62 time=0.674 ms
64 bytes from 10.0.4.22: icmp_seq=5 ttl=62 time=0.619 ms
64 bytes from 10.0.4.22: icmp_seq=6 ttl=62 time=0.477 ms
64 bytes from 10.0.4.22: icmp_seq=7 ttl=62 time=0.500 ms
64 bytes from 10.0.4.22: icmp_seq=8 ttl=62 time=0.539 ms
64 bytes from 10.0.4.22: icmp_seq=9 ttl=62 time=0.586 ms
64 bytes from 10.0.4.22: icmp_seq=10 ttl=62 time=0.684 ms
64 bytes from 10.0.4.22: icmp_seq=11 ttl=62 time=0.583 ms
64 bytes from 10.0.4.22: icmp_seq=12 ttl=62 time=0.662 ms
^C
--- 10.0.4.22 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11260ms
rtt min/avg/max/mdev = 0.477/0.630/0.839/0.089 ms
root@S1:/tmp/pycore.44023/S1.conf# ping 10.0.6.1
connect: Network is unreachable
root@S1:/tmp/pycore.44023/S1.conf#
```

Ping: Server S1-LaptopA3

Ping: Server S1-LaptopB1

Ping: Server S1-RouterExt

Figura 16

```
Terminal
root@Lb1:/tmp/pycore.44023/Lb1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.626 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.594 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.619 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.621 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.618 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=62 time=0.616 ms
^C
```

Ping: LaptopB1-ServerS1
depois de routeAdd

Figura 17

```
Terminal
root@La1:/tmp/pycore.44023/La1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.838 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.617 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.609 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.621 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.622 ms
^C
```

Ping: LaptopA1-ServerS1
depois de routeAdd

Figura 18

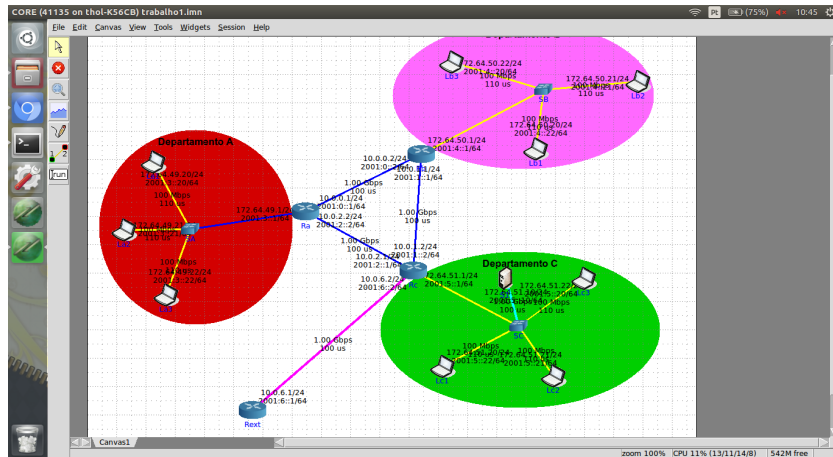


Figura 19

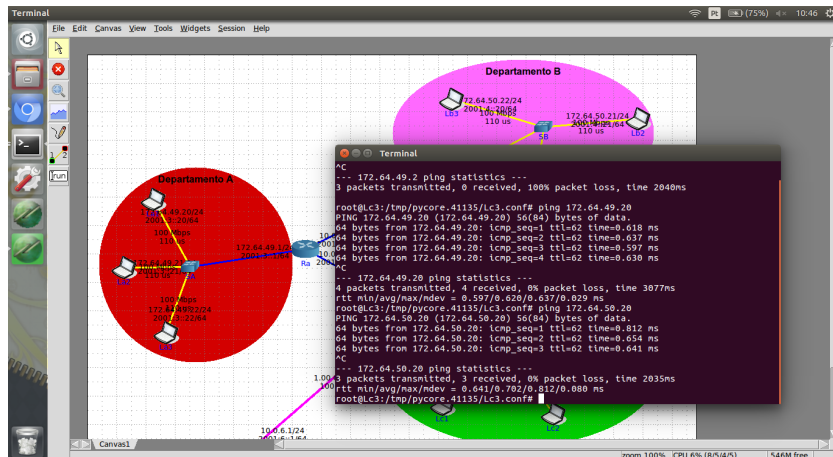


Figura 20

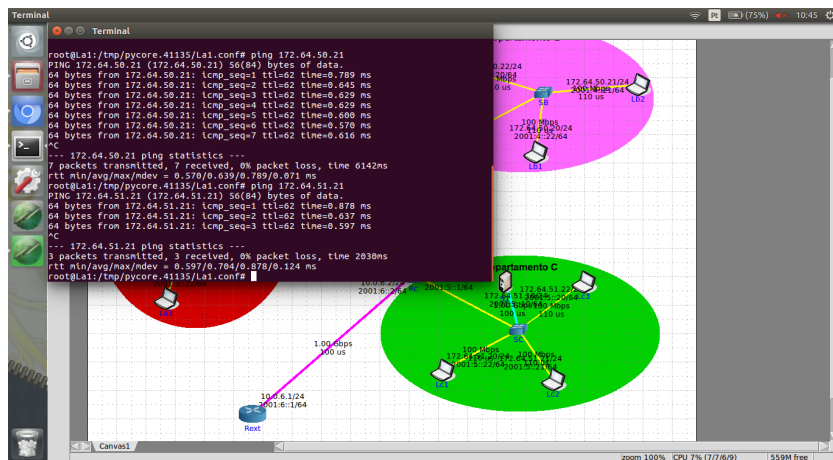


Figura 21