2010

ANÁLISE DE TRÁFEGO TCP/IP

Paulo Marcelo

paulo1410@hotmail.com

13/11/2010

MEMOREX ANÁLISE DE TRÁFEGO

Continuando a série dos ***memorex*** para concurso de TI, disponibilizo o material sobre análise de tráfego que havia prometido desde o ano passado na lista *timaster*. Vez por outra aparecem alguns concurseiros de TI solicitando materiais para o estudo sobre esse tema visando as provas do CESPE e a resposta era sempre a mesma: não existem livros nem tutoriais específicos sobre o assunto, além de alguns tutoriais voltados para a ferramenta X ou Y.

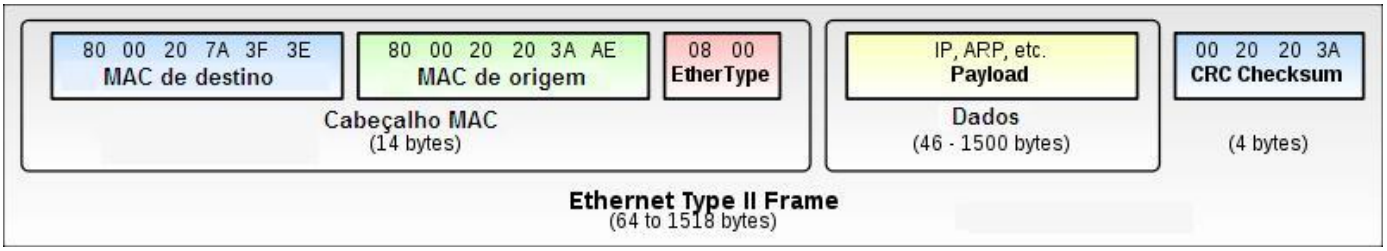
Acredito que este tenha sido o trabalho mais desafiador por se tratar de um tema que exige uma maior concentração bem como um conhecimento mais aprofundado no assunto. Durante minhas pesquisas pude observar diversos *sites* e *blogs* comentando algumas questões de forma superficial e muitas vezes errada, isso me fez levar mais tempo para elaboração do material, pois não gostaria de ser mais um a disponibilizar um tutorial cheio de furos.

Diversos exercícios foram comentados pelo Gustavo (p3r1t0f3d3r4l) na lista da *timaster* restando a mim, nestes casos, apenas a organização e formatação de suas contribuições (créditos mantidos). Como sempre, os memorex são free! Quem tiver dúvidas ou desejar contribuir com sugestões meu e-mail encontra-se no rodapé. Bons Estudos a todos!

ANÁLISE DE TRÁFEGO TCP/IP

Antes de falarmos em análise de tráfego devemos primeiramente entender os conceitos básicos que são envolvidos nesse tipo de questão. É preciso que se identifique claramente qual o tipo de informação estão sendo proposto na avaliação: estamos querendo avaliar pacotes da camada 3 ou segmentos da camada 4? Tais dados estão sendo trafegados em que tipo de enlace? Ethernet? Token ring?

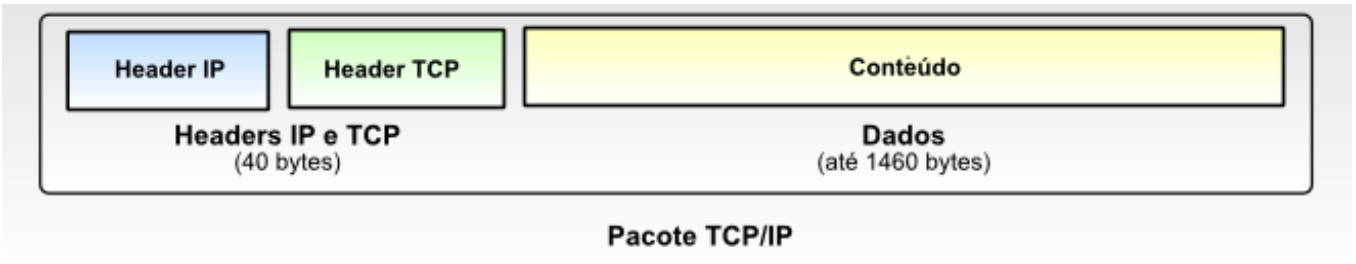
Os frames **Ethernet** são "envelopes" para os pacotes TCP/IP. O aplicativo (um navegador, um servidor web, ou qualquer outro aplicativo transmitindo dados pela rede) envia os dados ao sistema operacional, que divide a informação em pacotes TCP/IP e os envia à placa de rede. As placas de rede (que não entendem o protocolo TCP/IP) tratam os pacotes como um fluxo de dados qualquer e adicionam mais uma camada de endereçamento, desta vez baseada nos endereços MAC dos dispositivos da rede, gerando o frame Ethernet que é finalmente transmitido. Ao chegar do outro lado, o "envelope" é removido e o pacote TCP/IP é entregue.



OBS: Não mostrado na figura o preâmbulo de 8 bytes.

Voltando nossa atenção para o campo “conteúdo” vemos que a capacitdade máxima suportada por esse padrão será de 1500 bytes. Esse valor é conhecido como MTU (*Maximum Transmission Unit*). Informações das camadas superiores serão transportados dentro do campo de dados (ou conteúdo) obedecendo tal limitação do padrão ethernet.

Devido essa limitação do protocolo ethernet, as camadas superiores constumam separar o tamanho máximo por pacote para encaixar neste valor padrão. A aplicação, portanto, envia para a camada hierarquimante inferior dados de 1500 bytes, correto? Não! Tenha sempre em mente que além dos dados da aplicação, também conhecido como payload que é a carga útil de informação a ser transmidida, também devem ser acrescidos os overhead (informações extras) pelas camadas de tranporte, rede e enlace. Os overheads trazem dados de controle essenciais para que os pacotes alcancem seu destino, sendo descartados na ponta de destino.

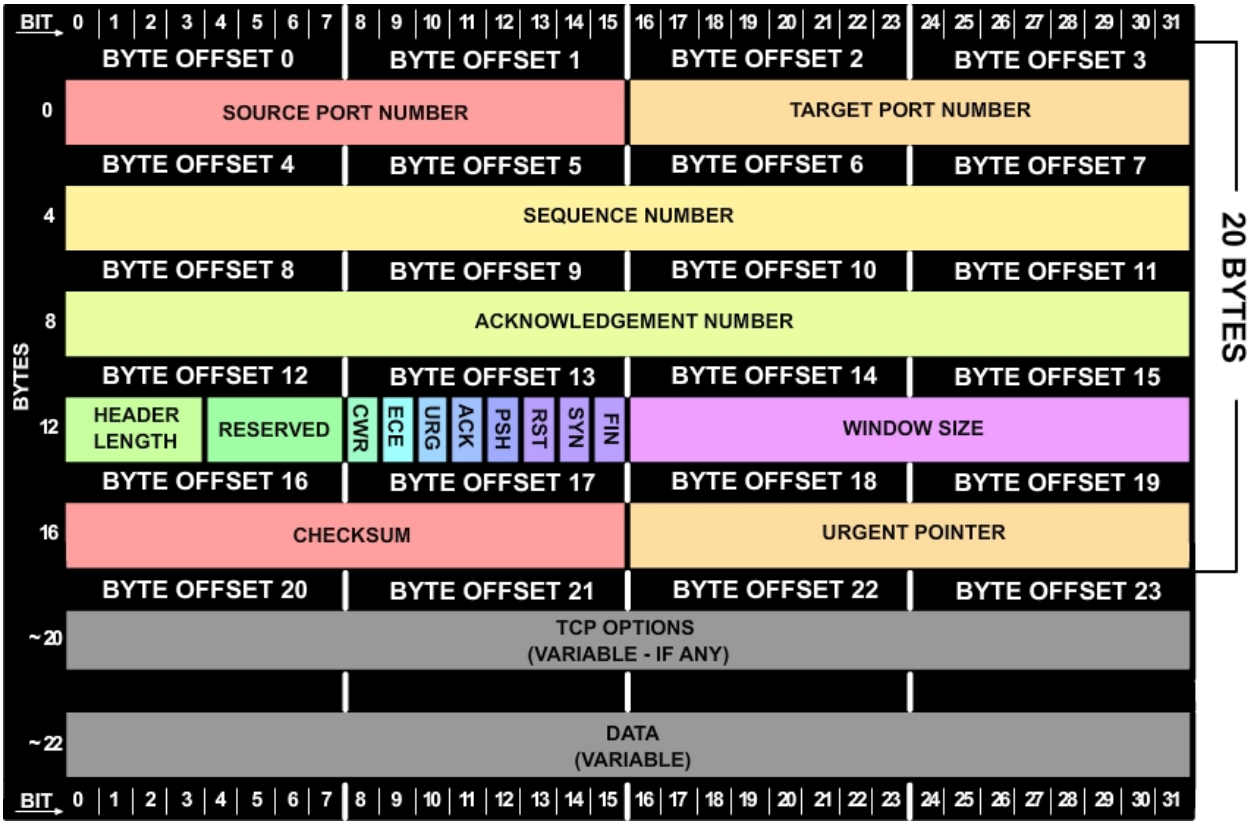


Dentro do pacote TCP/IP temos os headers (cabeçalhos), que contém o endereço IP de origem, endereço IP de destino, porta de origem, porta de destino, códigos de verificações, número do pacote, campo para inclusão de opções e vários outros. Geralmente, teremos 20 bytes para os headers do protocolo TCP e mais 20 bytes para os *headers* do protocolo IP, totalizando 40 bytes de headers por pacote. Por este motivo, a aplicação deverá entregar, no máximo, 1460 bytes de dados/payload que serão transmitidos em um pacote de 1500 bytes (lembre-se estamos falando do padrão ethernet).

Os *headers* do protocolo IP incluem o endereço IP de origem e de destino, enquanto os *headers* do TCP incluem a porta de origem e de destino. Resumidamente, podemos afirmar que o IP se encarrega da entrega dos pacotes (através da localização do endereço IP), enquanto o TCP se encarrega da verificação de erros, numeração de portas e etc.

Dentro da rede local ethernet, teremos um total de 1518 bytes transmitidos para cada pacote TCP/IP de 1500 bytes, incluído agora os 14 bytes do cabeçalho ethernet e os 4 bytes de CRC (checagem de erro). Se considerarmos também os 8 bytes iniciais que formam o preâmbulo, presente apenas durante o processo de sincronização, este número sobe para 1526 bytes. Considerando que cada pacote contém apenas 1460 bytes de carga útil (informação da camada de aplicação), teremos 66 bytes de *overhead* (dados de controle), que corresponde a quase 5% do volume de dados transmitidos por quadro!

O CABEÇALHO TCP



Sequence number: normalmente especifica o número assinalado para o primeiro byte de dados na mensagem corrente. Na fase de estabelecimento de uma conexão, pode ser usado como uma identificação da transmissão.

Acknowledgment number: contém o número seqüencial do próximo byte de dados que o dispositivo de origem espera receber.

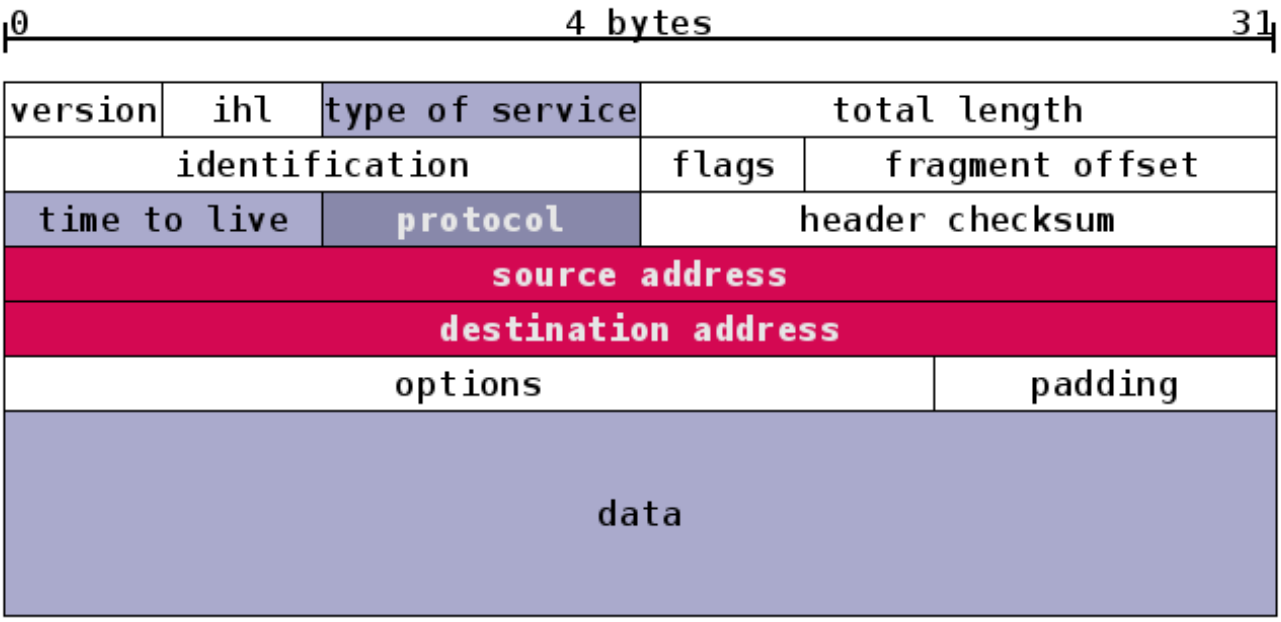
Reserved: reservado para uso futuro.

Flags: usado para uma variedade de informações de controle, como SYN e ACK para estabelecer conexão e FIN para terminar.

URG: Indica que o campo Urgent Pointer possui informações válidas.
ACK: Signfica que o campo ACK possui informações válidas, ou seja, o transmissor está reconhecendo o recebimento de uma informação anterior e está esperando mais informações.
PSH (Push): Indica que os dados recebidos devem passados imediatamente para a aplicação.
RST (Reset): Serve tanto para reinicializar uma conexão que tenha ficado confusa por falhas em um dos lados, tanto para indicar a rejeição de um pacote transmitido.
SYN: Significa que o campo SYN possui informações válidas, ou seja, o pacote possui dados transmitidos, e espera que o receptor as reconheça.
FIN: Indica que o transmissor não tem mais dados para enviar. Nesse caso, ele pode continuar recebendo pacotes, e os reconhecendo, até o emissor encerrar a sua transmissão.

Window: especifica o tamanho da parte de memória (buffer) disponível para os dados a receber.
Checksum: verificação da integridade do cabeçalho.
Urgent pointer: possui 16 bits (2 bytes) e, quando o **flag URG** está ativado, indica o offset do primeiro byte dos dados considerados urgentes, e que devem ser passados para aplicação antes dos demais.
Options: Podem ter até 40 bytes e como o nome diz, são campos opcionais, e podem implementar algumas funções adicionais como *Selective Acknowledgement* (SACK) (RFC 2018), *Window Scale* (RFC 1323) e *Timestamp TCP*.
Data: contém cabeçalho e dados da camada superior, isto é, a de aplicação. O seu comprimento é variável, podendo ser bem mais que os 32 bits indicados na tabela.

O CABEÇALHO IP



Version: indica a versão do IP em uso.
IHL (IP header length): indica o comprimento do pacote em palavras de 32 bits.
Type of service: especifica como o protocolo da camada acima gostaria de manipular o pacote e define vários níveis de importância.
Total length: indica o comprimento total em bytes do pacote, incluindo cabeçalho e dados.
Identification: contém um inteiro que identifica o pacote atual. Usado para auxiliar a recomposição de pacotes fragmentados.

Flags: campo de 3 bits para controlar fragmentação. O bit menos significativo indica se o pacote pode ser fragmentado. O central indica se o pacote atual é o último de uma série de pacotes fragmentados. O terceiro bit não é usado.
Fragment offset: indica a posição do fragmento de dados em relação ao início dos dados no pacote original o que permite a sua recomposição pelo processo no destino.
Time to live: um contador que é gradualmente decrementado até zero, quando o pacote é descartado. Isso evita um *loop* infinito de pacotes no sistema.
Protocol: define qual protocolo na camada superior receberá o pacote após o processamento do IP.
Header checksum: verificação da integridade do cabeçalho.
Source address: endereço da origem.
Destination address: endereço do destino.
Options: define várias opções do IP, como segurança.
Data: contém o pacote da camada acima. O comprimento pode ser bem maior que os 32 bits representados.

ANÁLISE DE TRÁFEGO TCP/UDP

1 IP 10.1.1.1.1047 > 10.1.1.2.25: S 1808759712:1808759712(0) win 65535 <mss 1460,nop,nop,sackOK>
2 IP 10.1.1.2.25>10.1.1.1.1047: S 730816141:730816141(0)ack 1808759713 win 65535 <mss 1460,nop,nop,sackOK>
3 IP 10.1.1.1.1047 > 10.1.1.2.25: . ack 1 win 65535
4 IP 10.1.1.2.25 > 10.1.1.1.1047: P 1:19(18) ack 1 win 65535
5 IP 10.1.1.1.1047 > 10.1.1.2.25: . ack 19 win 65517
6 IP 10.1.1.1.1047 > 10.1.1.2.25: P 1:14(13) ack 19 win 65517
7 IP 10.1.1.2.25 > 10.1.1.1.1047: P 19:43(24) ack 14 win 65535
8 IP 10.1.1.1.1047 > 10.1.1.2.25: . ack 43 win 65493
9 IP 10.1.1.1.1047 > 10.1.1.2.25: P 14:27(13) ack 43 win 65493
10 IP 10.1.1.2.25 > 10.1.1.1.1047: . ack 27 win 65535
11 IP 10.1.1.2.25 > 10.1.1.1.1047: P 43:63(20) ack 27 win 65535
12 IP 10.1.1.1.1047 > 10.1.1.2.25: . ack 63 win 65473
13 IP 10.1.1.1.1047 > 10.1.1.2.25: P 27:33(6) ack 63 win 65473
14 IP 10.1.1.2.25 > 10.1.1.1.1047: P 63:91(28) ack 33 win 65535
15 IP 10.1.1.1.1047 > 10.1.1.2.25: F 33:33(0) ack 91 win 65445
16 IP 10.1.1.2.25 > 10.1.1.1.1047: F 91:91(0) ack 33 win 65535
17 IP 10.1.1.1.1047 > 10.1.1.2.25: . ack 92 win 65445
18 IP 10.1.1.2.25 > 10.1.1.1.1047: F 91:91(0) ack 34 win 65535

Com base nessas informações, assinale a **única** opção correta.
a) Trata-se de uma conexão TCP de um cliente para um servidor de HTTP.
b) Essa conexão corresponde a uma transferência de grande volume de informações, com dados sendo enviados, e seus recebimentos, confirmados nos segmentos de 4 a 13.
c) O estabelecimento da conexão se dá nos segmentos 1 e 2.
d) Os segmentos 14, 15, 16, 17 e 18 encerram a conexão.
e) O segmento 18 corresponde à confirmação do recebimento do segmento 15, e o 17 reconhece o segmento 16.

COMENTÁRIOS

a. Trata-se de uma conexão com um servidor SMTP (conexão na porta 25)
b. Transferências de alto volume tendem a encher a janela de recepção (o que não é o caso)
c. Conexão estabelecida nas linhas 1, 2 e 3
d. Segmento 14 não participa do encerramento da conexão
e. Segmento 18 confirma o 15 e o 16 confirma o 17.

Resposta correta: Letra "E" A seguir vamos fazer um análise linha-a-linha:

==========

INÍCIO DO ESTABELECIMENTO DA CONEXÃO

==========

1 CLIENTE > SERVIDOR_SMTP: S 1808759712:1808759712(0) win 65535 <mss 1460,nop,nop,sackOK>

Cliente contata o servidor e pede para fazer conexão, para isso envia seu número de sequencia inicial e não transmite nenhum byte de informação. Informa ainda o tamanho da janela de recepção (**win 65535)**. Nos campos opcionais (entre < >) Outra informação passada é o tamanho máximo de segmento da rede onde está plugado (que tem tudo para ser ethernet) e diz que a técnica de recepção empregada na janela deslizante será o ack-seletivo, ou seja, só serão reenviados os bytes que não tiverem sua recepção confirmada.

2 SERVIDOR_SMTP>10.1.1.1.1047: S 730816141:730816141(0)ack 1808759713 win 65535 <mss 1460,nop,nop,sackOK>

Servidor responde ao cliente que também quer estabelecer a conexão, para isso envia seu número de sequencia inicial e não transmite nenhum byte de informação. Confirma a recepção do número de sequência inicial do cliente informando qual byte está apto a receber (número de seq. recebido no segmento anterior + 1). O servidor é esperto, mandou num único pacote a solicitação de conexão e confirmação do segmento anterior. A esta técnica dá-se o nome de *piggybacking*. Outra informação passada é o tamanho máximo de segmento da rede onde está plugado (que tem tudo para também ser ethernet) e diz que a técnica de recepção empregada na janela deslizante será o ack-seletivo.

3 CLIENTE > SERVIDOR_SMTP: . ack 1

Cliente confirma o pedido de conexão do servidor e está estabelecida a conexão (este processo é conhecido como *3-way handshake* conforme ilustrado na imagem). A partir desse ponto, os números de seqüencias iniciais são substituídos por uma contagem mais compreensiva pelo softwares que usam o padrão TCPDUMP. Tomei a liberdade de excluir os tamanhos de janela a partir desse ponto para facilitar o entendimento.

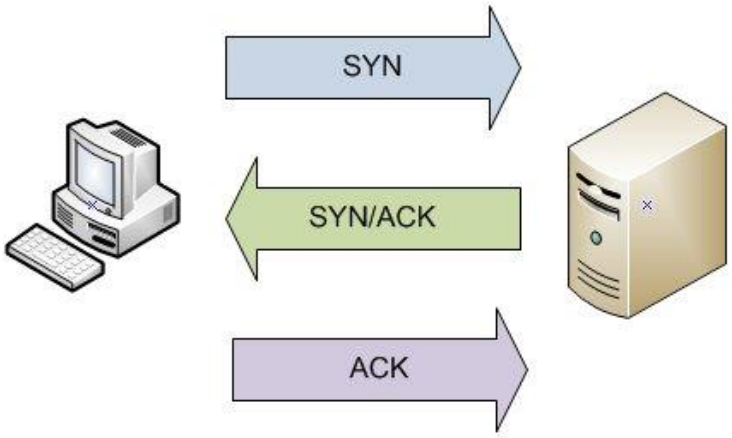


Figura : Estabelecimento da conexão TCP: *3-way handshake*

==========

FIM DO ESTABELECIMENTO DA CONEXÃO

==========

==========

TROCA DE DADOS

==========

4 SERVIDOR_SMTP > cliente: P 1:19(18) ack 1

Servidor envia 18 bytes de informação para o cliente (contagem inicia no 1 e finaliza no 19, que não é enviado) e através do ack 1 informa que espera receber o byte 1 do cliente. O Flag PUSH está setado, ou seja, a informação não esperará o buffer completar para enviar a informação, ela será enviada de imediato.

5 CLIENTE > SERVIDOR_SMTP: . ack 19

Cliente responde que espera receber a partir do byte 19 do servidor, indicando que até o byte 18 está tudo confirmado.

6 CLIENTE > SERVIDOR_SMTP: P 1:14(13) ack 19

Cliente envia 13 bytes para o servidor e confirma que espera receber a partir do byte 19.

7 SERVIDOR_SMTP > cliente: P 19:43(24) ack 14

Servidor envia 24 bytes para o servidor e confirma que espera receber a partir do byte 14.

Esse chove-não-molha se repetira até a **linha 14**, razão pela qual vou pular a descrição destas linhas

8 CLIENTE > SERVIDOR_SMTP: . ack 43
9 CLIENTE > SERVIDOR_SMTP: P 14:27(13) ack 43
10 SERVIDOR_SMTP > cliente: . ack 27
11 SERVIDOR_SMTP > cliente: P 43:63(20) ack 27
12 CLIENTE > SERVIDOR_SMTP: . ack 63
13 CLIENTE > SERVIDOR_SMTP: P 27:33(6) ack 63
14 SERVIDOR_SMTP > cliente: P 63:91(28) ack 33

==========

FIM DA TROCA DE DADOS E INÍCIO DO ENCERRAMENTO DA CONEXÃO

==========

15 CLIENTE > SERVIDOR_SMTP: F 33:33(0) ack 91

Cliente comunica ao servidor que pretende encerrar a comunicação (flag FIN setada), não envia nenhum dado e confirma a recepção do byte 90 (transmitido na linha 14), haja vista indicar que espera receber o byte 91. (Mesmo quando nenhuma informação é enviada, uma numeração de ack é consumida, observe que o próximo ack do cliente deverá ser 92)

16 SERVIDOR_SMTP > cliente: F 91:91(0) ack 33

Servidor comunica ao cliente que deseja encerrar a conexão e confirma a recepção do byte 32 (linha 13), uma vez que espera receber o byte 33.

OBS: Pelo que tudo indica, as máquinas solicitaram o encerramento da conexão quase que simultaneamente, e pode ser que as respostas cheguem fora da sequencia de solicitação, uma vez que podem seguir caminhos

diversos com situações de congestionamento diversas. Nenhuma das máquinas recebeu ainda a confirmação de que sua solicitação foi recebida pela contraparte, e ambas esperam por uma resposta. Se nenhum dado chegar num intervalo equivalente a duas vezes o *round-trip-time* a conexão será terminada pois a extremidade que está na espera interpreta que o outro lado não tem nada para mandar.
OBS 2: Para melhor entendimento cronológico do trecho a seguir, inverti a sequencia das próximas linhas.

18 SERVIDOR_SMTP > cliente: F 91:91(0) ack 34
Servidor acusa recebimento do byte 33 (enviado na linha 15) e informa que pretende encerrar a conexão. A conexão do lado cliente para servidor está desfeita.

17 **CLIENTE > SERVIDOR_SMTP: . ack 92**
Cliente confirma a recepção da solicitação de encerramento (da linha 16) encerrando o outro nodo de conexão (servidor para cliente).

Contribuição: [Gustavo/p3r1t0f3d3r4l]

==========
FIM DO ENCERRAMENTO DA CONEXÃO
==========

MPOG

1. 0.055429 IP (tos 0x0, ttl 128, id 2442, offset 0, flags [DF], proto: TCP (6), length: 48) 10.1.1.1.2373 > 10.1.1.2.7777: S, cksum 0x9764 (correct), 160520737:160520737(0) win 64240 <mss1460,nop, nop, sackOK>
2. 0.055990 IP (tos 0x0, ttl 128, id 2691, offset 0, flags [DF], proto: TCP (6), length: 48) 10.1.1.2.7777 > 10.1.1.1.2373: S, cksum 0xb8a6 (correct), 3778458614:3778458614(0) ack 160520738 win 17520 <mss1460,nop,nop,sackOK>
3. 0.056088 IP (tos 0x0, ttl 128, id 2443, offset 0, flags [DF], proto: TCP (6), length: 40) 10.1.1.1.2373 > 10.1.1.2.7777: ., cksum 0x2eea (correct), ack 1 win 64240
4. 0.095338 IP (tos 0x0, ttl 128, id 2450, offset 0, flags [DF], proto: TCP (6), length: 1064) 10.1.1.1.2373 > 10.1.1.2.7777: P, cksum 0x24b7 (correct), 1:1025(1024) ack 1 win 64240
5. 0.095444 IP (tos 0x0, ttl 128, id 2451, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.1.1.1.2373 > 10.1.1.2.7777: P, cksum 0xc78f (correct), 1025:2485(1460) ack 1 win 64240
6. 0.098918 IP (tos 0x0, ttl 128, id 2698, offset 0, flags [DF], proto: TCP (6), length: 40) 10.1.1.2.7777 > 10.1.1.1.2373: ., cksum 0xdbb6 (correct), ack 2485 win 17520
7. 0.099035 IP (tos 0x0, ttl 128, id 2452, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.1.1.1.2373 > 10.1.1.2.7777: P, cksum 0x970f (correct), 2485:3945(1460) ack 1 win 64240
8. 0.099073 IP (tos 0x0, ttl 128, id 2453, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.1.1.1.2373 > 10.1.1.2.7777: P, cksum 0x1825 (correct), 3945:5405(1460) ack 1 win 64240
9. 0.099109 IP (tos 0x0, ttl 128, id 2454, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.1.1.1.2373 > 10.1.1.2.7777: P, cksum 0x738f (correct), 5405:6865(1460) ack 1 win 64240
10. 0.103041 IP (tos 0x0, ttl 128, id 2705, offset 0, flags [DF], proto: TCP (6), length: 40) 10.1.1.2.7777 > 10.1.1.1.2373: ., cksum 0xd04e (correct), ack 5405 win 17520

Considerando o trecho de captura de tráfego de rede apresentado acima, julgue os próximos itens.

1. A captura em apreço ilustra uma conexão TCP com todas as suas fases, com tráfego interativo.
2. Assumindo que a captura apresentada adira ao modelo cliente-servidor, o cliente seria o host 10.1.1.1 e servidor, o host 10.1.1.2.
3. Segundo a captura em questão, ocorrem retransmissões de pacotes.

COMENTÁRIOS

1. Uma conexão TCP, via de regra, é caracterizada pelo estabelecimento, transmissão de dados e encerramento da mesma. Neste caso temos o estabelecimento nos itens 1, 2 e 3 onde o bit "S" (SYN) está presente no 1, "S" e "ack" no 2 e "ack" no 3. Este conjunto representa o hand-shake de 3 vias, estabelecendo com sucesso a conexão. Já o encerramento da conexão não aparece, pois ele se caracteriza pelo encerramento através dos bit "F" (FIN) e ack em ambas direções, ou seja duas vezes. Existe ainda um encerramento abrupto que se caracteriza pelo bit "R" (RST) somente em um sentido pois este "derruba" a conexão sem esperar o outro lado. Esta forma de encerramento também está ausente na questão.
2. Numa arquitetura cliente servidor, o servidor adota uma postura passiva e o cliente uma postura ativa. Observe que a iniciativa do estabelecimento da conexão parte de 10.1.1.1, indicando que este é o cliente. Outra característica que indica o cliente e o servidor, entretanto, não existente no exemplo, é o uso de portas baixas (< 1024) no servidor (*well known ports*) e portas altas (> 1024) usadas no cliente.
3. De acordo com a análise dos pacotes, observa-se que o campo ID (identificação) do cabeçalho IP não se repete. Dessa forma, não há retransmissão dos pacotes.

Comentários: [Gustavo/p3r1t0f3d3r4l]

GAB: E C E.

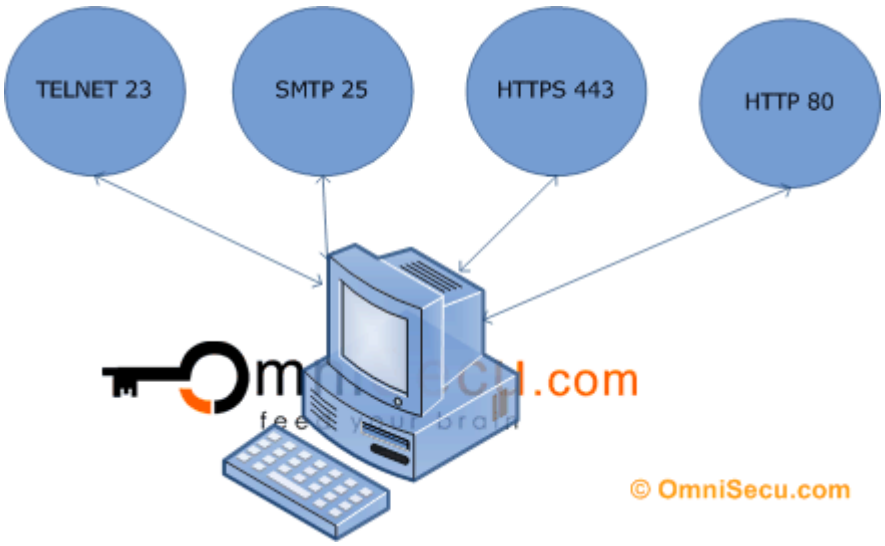


Figura : Exemplo de *well known ports.*

TRT17 2009

1 IP 10.0.1.18.1047 > 10.0.1.200.110: S 59712:59712(0) win 65535 <mss 1460,nop,nop,sackOK>
2 IP 10.0.1.200.110 > 10.0.1.18.1047: S 16141:16141(0) ack 59713 win 65535 <mss 1460,nop,nop,sackOK>
3 IP 10.0.1.18.1047 > 10.0.1.200.110: . ack 1 win 65535
4 IP 10.0.1.200.110 > 10.0.1.18.1047: P 1:19(18) ack 1 win 65535
5 IP 10.0.1.18.1047 > 10.0.1.200.110: . ack 19 win 65517
6 IP 10.0.1.18.1047 > 10.0.1.200.110: P 1:14(13) ack 19 win 65517
7 IP 10.0.1.200.110 > 10.0.1.18.1047: P 19:43(24) ack 14 win 65535
8 IP 10.0.1.18.1047 > 10.0.1.200.110: . ack 43 win 65493
9 IP 10.0.1.18.1047 > 10.0.1.200.110: P 14:27(13) ack 43 win 65493
10 IP 10.0.1.200.110 > 10.0.1.18.1047: . ack 27 win 65535
11 IP 10.0.1.200.110 > 10.0.1.18.1047: P 43:63(20) ack 27 win 65535
12 IP 10.0.1.18.1047 > 10.0.1.200.110: . ack 63 win 65473
13 IP 10.0.1.18.1047 > 10.0.1.200.110: P 27:33(6) ack 63 win 65473
14 IP 10.0.1.200.110 > 10.0.1.18.1047: P 63:91(28) ack 33 win 65535
15 IP 10.0.1.18.1047 > 10.0.1.200.110: F 33:33(0) ack 91 win 65445
16 IP 10.0.1.200.110 > 10.0.1.18.1047: F 91:91(0) ack 33 win 65535
17 IP 10.0.1.18.1047 > 10.0.1.200.110: . ack 92 win 65445
18 IP 10.0.1.200.110 > 10.0.1.18.1047: F 91:91(0) ack 34 win 65535

Considerando o trecho de captura de tráfego acima, julgue os itens subsequentes.

1. Na captura mostrada, os segmentos com o flag PUSH setado transferem, no payload de dados, o tamanho do MSS.
2. Os hosts estão na mesma rede se utilizarem uma máscara de rede de 24 bits.
3. A captura apresenta uma aplicação que utiliza o protocolo UDP.
4. O primeiro pacote tem origem no cliente.
5. A captura apresenta informações referentes às camadas de enlace, rede, transporte e aplicação.
6. A aplicação utilizada libera a conexão no décimo quinto datagrama.
7. Em nenhum dos segmentos a janela deslizante foi totalmente ocupada.
8. O fluxo de dados da aplicação se inicia no primeiro datagrama.
9. O estabelecimento da conexão ocorre nos três primeiros segmentos.
10. O encerramento da conexão ocorre entre os cinco últimos segmentos.
11. A captura apresenta uma transferência de dados em volume.
12. Durante o trecho de captura mostrado ocorreu retransmissão de informação.

COMENTÁRIOS

1 O MSS é transferido no campo OPÇÕES do cabeçalho TCP, nunca no payload. Esses parâmetros são negociados no estabelecimento da conexão e, ainda no trecho apresentado, nenhum segmento com PUSH setado está carregando dados com o tamanho máximo do MSS.

2 Certa. Máscara de 24 bits: 255.255.255.0
Com esta máscara, para estarem na mesma rede, dois IPs devem possuir os mesmos valores para os 3 primeiros octetos, que é o que acontece: 10.0.1.X
10.0.1.18.1047
10.0.1.200.110

Obs: Os números 1047 e 110 são as portas dos hosts.
3. Falso. A aplicação utilizada é o POP (porta 110) que só utiliza TCP. Alerto que se a porta usada fosse 53 (DNS) e o trecho de tráfego mostrasse ainda APENAS TCP, a questão estaria CORRETA. Observe que a questão não pergunta se existe TRÁFEGO UDP, mas sim se a APLICAÇÃO usa protocolo UDP. No segundo caso (53), DNS usa tanto TCP quanto UDP, independentemente do que vier mostrado na análise de tráfego

4. Correto. O servidor adota postura passiva e o cliente toma a iniciativa. O cliente usa portas altas para iniciar a conexão, o servidor "escuta" em portas baixas (<1024). UDP segue o mesmo raciocínio, entretanto, não existe estabelecimento de conexão. Exemplos: DNS, TFTP, NFS, WHOIS, BOOTP

5. Falso. Camada de aplicação: MSS (usado pela aplicação e não considera o tamanho do cabeçalho de transporte); Camada de transporte: Portas; camada de rede: IP; camada de enlace: INEXISTENTE

6. Correta. A **APLICAÇÃO** utilizada libera a conexão no trecho 15. Daí em diante, quem cuida é o Sistema Operacional. A aplicação já tem terminado quando a conexão fica nos estados FIN_WAIT. [Alex]

7. Correto. O tamanho máximo do win (65535). O menor valor atingindo no trecho 5 foi 65445. Mesmo que em algum momento o valor win fosse igual a zero, não poderíamos afirmar que a janela foi totalmente ocupada, pois "Win=0" é um pedido do receptor para que o emissor suspenda o envio até "segunda ordem", significa "sem janela". Um exemplo disso é quando o receptor está envolvido em outras tarefas mais prioritárias e quer dedicar todo seu poder de processamento às mesmas, deixando a recepção dos pacotes para depois. Existem exceções onde o transmissor não respeitará tal ordem: 1. Quando o segmento a ser enviado estiver com o flag URGENTE setado (dados fora da faixa) 2. Quando o transmissor envia um byte pedindo o próximo byte esperado e tamanho da janela.

8. Errada. Os 3 primeiros segmentos/linhas correspondem ao handshake (inicialização da conexão). O fluxo de dados se inicia no quarto segmento/linha. Observe a nomenclatura "datagrama" (usada geralmente para fluxo UDP), em vez de "segmento" (normalmente para o fluxo TCP). A maioria das bancas não se apega à nomenclatura correta da PDU, logo os termos datagrama, pacote, segmento são usados indistintamente.

9. Certa. O handshake é feito em 3 passos: 1 - cliente envia flag SYN (S); 2 - servidor envia flags SYN + ACK; 3 - cliente envia ACK apenas.

10 Certa. Essa é uma pegadinha de português. O encerramento, na verdade, ocorre nos 4 últimos segmentos. Então está correto dizer que também ocorre entre os 5 últimos, uma vez que abrange os 4 últimos.

11. Errada. Se fosse transferência em volume, ou seja, grande quantidade de informações, a quantidade de bytes transmitidos tenderia a ser igual ao tamanho do MSS (que é negociado no estabelecimento da conexão), neste caso 1460.

12. Errada. A linhas 18 que poderia causar confusão, é confirmação (ack) da linha 15 (finalização).

GABARITO

1	2	3	4	5	6	7	8	9	10	11	12
E	C	E	C	E	C	C	E	C	C	E	E

SERPRO 2008

```
1  0.036806 IP (tos 0x0, ttl 128, id 26676, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x13e3 (correct), ack 1308900895 win 65535 <nop,nop,sack 1 {1461:46721.
2  0.148342 IP (tos 0x0, ttl 52, id 49837, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x8c0d (correct), 1:1461(1460) ack 0 win 5840.
3  0.148675 IP (tos 0x0, ttl 128, id 26677, offset 0, flags [DF], proto: TCP (6), length: 40) 10.10.100.101.1010
   10.10.10.20.2020: ., cksum 0x67f5 (correct), ack 46721 win 65535.
4  0.148708 IP (tos 0x0, ttl 52, id 49838, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0xd9f5 (correct), 49641:51101(1460) ack 0 win 5840.
5  0.148740 IP (tos 0x0, ttl 128, id 26678, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x9011 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:51101}.
6  0.149045 IP (tos 0x0, ttl 52, id 49839, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0xaa9a (correct), 51101:52561(1460) ack 0 win 5840.
7  0.149091 IP (tos 0x0, ttl 128, id 26679, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x8a5d (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:52561}>.
8  0.170421 IP (tos 0x0, ttl 52, id 49840, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x8d79 (correct), 52561:54021(1460) ack 0 win 5840.
9  0.170489 IP (tos 0x0, ttl 128, id 26680, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10. 10.10.10.20.2020: ., cksum 0x84a9 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:54021}>
10 0.174039 IP (tos 0x0, ttl 52, id 49841, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x38da (correct), 54021:55481(1460) ack 0 win 5840.
11 0.174056 IP (tos 0x0, ttl 128, id 26681, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x7ef5 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:55481}>.
12 0.174766 IP (tos 0x0, ttl 52, id 49842, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0xa178 (correct), 55481:56941(1460) ack 0 win 5840.
13 0.174783 IP (tos 0x0, ttl 128, id 26682, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x7941 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:56941}>.
14 0.175877 IP (tos 0x0, ttl 52, id 49843, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x4c94 (correct), 56941:58401(1460) ack 0 win 5840
15 0.175893 IP (tos 0x0, ttl 128, id 26683, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x738d (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:58401}>;
16 0.176227 IP (tos 0x0, ttl 52, id 49844, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x35ba (correct), 58401:59861(1460) ack 0 win 5840
17 0.176242 IP (tos 0x0, ttl 128, id 26684, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x6dd9 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:59861}>
18 0.244960 IP (tos 0x0, ttl 52, id 49845, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0xe681 (correct), 59861:61321(1460) ack 0 win 5840
19 0.245007 IP (tos 0x0, ttl 128, id 26685, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x6825 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:61321}>
20 0.255050 IP (tos 0x0, ttl 52, id 49846, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x801a (correct), 61321:62781(1460) ack 0 win 5840
21 0.255068 IP (tos 0x0, ttl 128, id 26686, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x6271 (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:62781}>
22 0.373902 IP (tos 0x0, ttl 52, id 49847, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: P, cksum 0xdc94 (correct), 62781:64241(1460) ack 0 win 5840
23 0.373971 IP (tos 0x0, ttl 128, id 26687, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010
   > 10.10.10.20.2020: ., cksum 0x5cbd (correct), ack 46721 win 65535 <nop,nop,sack 1 {49641:64241}>
24 0.374334 IP (tos 0x0, ttl 52, id 49848, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0xa881 (correct), 46721:48181(1460) ack 0 win 5840
25 0.374669 IP (tos 0x0, ttl 52, id 49849, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020
   > 10.10.100.101.1010: ., cksum 0x89bd (correct), 64241:65701(1460) ack 0 win 5840
```

Com referência ao trecho de captura de tráfego acima apresentado, julgue os itens a seguir.

1 A captura deve ter sido realizada no host 10.10.10.20 ou no segmento em que ele se encontrava.
2 A chegada fora de ordem de alguns segmentos deve-se às condições de transmissão nos enlaces de rede entre os hosts e não à transmissão decorrente de perda de segmento
3 Há três segmentos retransmitidos no trecho de captura em questão.
4 A opção sack nos segmentos de 13 a 20 refere-se a mais de um segmento perdido.
5 Nem todos os segmentos perdidos foram retransmitidos.

COMENTÁRIOS

Uma das questões mais difíceis sobre análise de tráfego, por isso, muita atenção no acompanhamento da resolução. Primeiramente devemos observar que apenas o host 10.10.100.1010 (cliente) estará recebendo informações do host 10.10.10.2020 (servidor) durante toda captura. Sua resposta ao servidor consiste apenas em enviar as confirmações (ack).

1 0.036806 IP (tos 0x0, ttl 128, id 26676, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x13e3 (correct), ack 1308900895 win 65535 <nop,nop,**sack 1 {1461:46721}>.**
Logo no primeiro trecho vemos a presença de 1 sack informando uma confirmação seletiva dos pacotes 1461 a 46721, porém aguarda pacote(s) anterior(es), ou seja houve perda de pacote durante a transmissão.

2 0.148342 IP (tos 0x0, ttl 52, id 49837, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x8c0d (correct), **1:1461**(1460) ack 0 win 5840.
No trecho 2 ocorre a retransmissão pelo servidor (10.10.10.20) do pacote perdido {1:1461}. Aqui já podemos concluir que cada segmento terá um tamanho de 1460, ou seja, provavelmente o enlace de dados trata-se de um ethernet. Esta informação será importante para calcularmos a quantidade de segmentos retransmitidos que aparece em uma das perguntas.

3 0.148675 IP (tos 0x0, ttl 128, id 26677, offset 0, flags [DF], proto: TCP (6), length: 40) 10.10.100.101.1010 10.10.10.20.2020: ., cksum 0x67f5 (correct), **ack 46721** win 65535.
No trecho 3 o receptor (cliente) reconhece a transmissão normal (ack) até o pacote 46720 dizendo que espera receber o próximo pacote 46721. Vale ressaltar que os trechos {1461:46721} já haviam sido confirmados pelo cliente no trecho 1, porém através de um reconhecimento seletivo (sack) dizendo ao transmissor: "ok servidor, estou te confirmando esses dados, mas você ainda me deve segmento(s) anterior(es) a estes (neste exemplo o trecho 1:1461)". Quando o servidor recebe o reconhecimento normal (ack) sem nenhum reconhecimento do tipo sack, significa que o cliente recebeu todos os pacotes que estavam em atraso e até o momento não há mais pendências.

4 0.148708 IP (tos 0x0, ttl 52, id 49838, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0xd9f5 (correct), **49641:51101**(1460) ack 0 win 5840.
Continuidade da transmissão {49641:51101}, respeitando o tamanho máximo do MTU do meio (1460). Perceba que 49641 não é a continuação do segmento esperado (46721), ou seja, este segmento se perdeu no caminho. **Agora, neste trecho tiramos nossa conclusão da primeira questão. Por que não vemos os dados do segmento {46721:49460} na captura do tráfego? Resposta: Porque o *sniffer* (analisador do tráfego) se encontrava do lado do cliente (10.10.100.101)! O pacote se perde antes de chegar ao receptor por isso não aparece na captura. Portanto primeira questão FALSA.**

5 0.148740 IP (tos 0x0, ttl 128, id 26678, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x9011 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:51101}.**
O receptor responde que confirma o recebimento dos dados de forma seletiva (ack) dos trechos {49641:51101}, porém o reconhecimento normal parou no segmento 46721 (ack). Com o sack é possível que o receptor confirme blocos entregues fora da ordem esperada sejam eles por atraso ou porque foram perdidos. Só que nesta questão não existiram PACOTES (camada 3) fora de ordem. Observe os números "ID" sendo transmitidos de forma ordenados e sequenciais para concluir que não há problemas com as condições de transmissão nos enlaces de rede. **Então respondendo a segunda pergunta da questão: a chegada fora de ordem deve-se sim a perda de segmentos e não há problemas com as condições dos enlaces de redes.**

6 0.149045 IP (tos 0x0, ttl 52, id 49839, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0xaa9a (correct), **51101:52561**(1460) ack 0 win 5840.
O servidor continuar o envio de pacotes sem se preocupar com os pacotes que foram perdidos, ou seja, ele não os retransmite por enquanto.

7 0.149091 IP (tos 0x0, ttl 128, id 26679, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x8a5d (correct), **ack 46721** win 65535 <nop,nop,sack 1 {**49641:52561**}>.
O cliente continua confirmando o recebimento dos dados seletivamente aguardando a retransmissão dos dados perdido**.** Quantos pacotes foram perdidos até aqui? Vejamos: do trecho 4 temos que a confirmação seletiva começou a partir do 49641, quando o esperado era 46721. Fazendo o cálculo: 49641 – 46721 = 2920, que pelo tamanho do MTU concluímos se tratar de 2 segmentos de 1460! Fechando o cálculo: 2 segmentos perdidos no trecho 4 somado com 1 segmento perdido no trecho 1 totaliza **3 segmentos perdidos até aqui!**

8 0.170421 IP (tos 0x0, ttl 52, id 49840, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x8d79 (correct), **52561:54021(1460)** ack 0 win 5840.
O trecho 8 ao 23 segue um mesmo padrão. Nele não ocorrem novas perdas nem retransmissões, apenas envio de dados seqüencialmente com confirmações seletivamente (sack)

9 0.170489 IP (tos 0x0, ttl 128, id 26680, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10. 10.10.10.20.2020: ., cksum 0x84a9 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:54021}**>
10 0.174039 IP (tos 0x0, ttl 52, id 49841, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x38da (correct), **54021:55481(1460)** ack 0 win 5840.
11 0.174056 IP (tos 0x0, ttl 128, id 26681, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x7ef5 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:55481}>.**
12 0.174766 IP (tos 0x0, ttl 52, id 49842, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0xa178 (correct), **55481:56941(1460)** ack 0 win 5840.
13 0.174783 IP (tos 0x0, ttl 128, id 26682, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x7941 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:56941}**>.
14 0.175877 IP (tos 0x0, ttl 52, id 49843, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x4c94 (correct), **56941:58401(1460)** ack 0 win 5840
15 0.175893 IP (tos 0x0, ttl 128, id 26683, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x738d (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:58401}>;**
16 0.176227 IP (tos 0x0, ttl 52, id 49844, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x35ba (correct), 58401:59861(1460) ack 0 win 5840
17 0.176242 IP (tos 0x0, ttl 128, id 26684, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x6dd9 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:59861}>**
18 0.244960 IP (tos 0x0, ttl 52, id 49845, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0xe681 (correct), 59861:61321(1460) ack 0 win 5840
19 0.245007 IP (tos 0x0, ttl 128, id 26685, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x6825 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:61321}>**
20 0.255050 IP (tos 0x0, ttl 52, id 49846, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x801a (correct), 61321:62781(1460) ack 0 win 5840
Respondendo a **quarta questão:** a opção sack **entre os trechos 13 a 20** (como também do trecho 4 ao trecho 23) refere-se a dois segmentos perdidos de 1460 {46721:49641}. Questão correta.

21 0.255068 IP (tos 0x0, ttl 128, id 26686, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x6271 (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:62781}>**
22 0.373902 IP (tos 0x0, ttl 52, id 49847, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: P, cksum 0xdc94 (correct), 62781:64241(1460) ack 0 win 5840
23 0.373971 IP (tos 0x0, ttl 128, id 26687, offset 0, flags [DF], proto: TCP (6), length: 52) 10.10.100.101.1010 > 10.10.10.20.2020: ., cksum 0x5cbd (correct), **ack 46721** win 65535 <nop,nop,**sack 1 {49641:64241}>**
24 0.374334 IP (tos 0x0, ttl 52, id 49848, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0xa881 (correct), **46721:48181(1460)** ack 0 win 5840
A **terceira questão** quer saber quantos segmentos foram **RETRANSMITIDOS** e não sobre a quantidade de segmentos **PERDIDOS**. Já sabemos que a quantidade de segmentos **PERDIDOS** foram 3: o segmento perdido no trecho 1 **que foi logo retransmitido no trecho 2** e agora, observando o trecho 24, há uma segunda

retransmissão de apenas **um dos dois segmentos que foram perdidos no trecho 4** {46721:48181}. Perceba que **não houve retransmissão do trecho {48181:49641), apenas o primeiro trecho {46721:48181} foi retransmitido.** Portanto resposta da **terceira questão: apenas dois segmentos foram retransmitidos durante captura e não 3 como afirma a questão. Esta análise também responde a quinta e ultima questão: nem todos os segmentos perdidos foram retransmitidos.**

25 0.374669 IP (tos 0x0, ttl 52, id 49849, offset 0, flags [DF], proto: TCP (6), length: 1500) 10.10.10.20.2020 > 10.10.100.101.1010: ., cksum 0x89bd (correct), **64241:65701(1460)** ack 0 win 5840
Continuação da transmissão do ultimo segmento do trecho 23. Não tem nenhuma relevância para nossa análise. Para fecharmos o entendimento: **três (3) segmentos foram perdidos, mas apenas dois (2) foram retransmitidos na captura em questão.**

OBS: Na questão 1, uma outra forma de identificar qual o lado que se encontra o analisador de tráfego (*sniffer*) é através da observação do campo TTL. Quando este valor estiver fora do padrão do sistema operacional (64, 128, 255) significa que o pacote atravessou nós na rede que acarretou seu decremento. Mais adiante veremos questões envolvendo o TTL.

Comentários: [Paulo Marcelo/paulo1410]

GAB: 1E – 2E – 3E – 4C – 5C

TCU 2009

1 771929 IP (tos 0x10, ttl 64, id 46018, offset 0, flags [DF], proto: TCP (6), length: 60) 192.168.1.1.5463 > 192.168.8.8.3421: S, cksum 0x1db2 (correct), 0:0(0) win 5840 <mss 1460,sackOK,timestamp 2538826 0,nop,wscale 6>

2. 1.994556 IP (tos 0x0, ttl 50, id 20037, offset 0, flags [DF], proto: TCP (6), length: 44) 192.168.8.8.3421 > 192.168.1.1.5463: S, cksum 0x9e62 (correct), 0:0(0) ack 1160318601
win 5840 <mss 1460>

3. 1.994605 IP (tos 0x10, ttl 64, id 46019, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb61f (correct), 1:1(0) ack 1 win 5840

4. 4.909380 IP (tos 0x10, ttl 64, id 46020, offset 0, flags [DF], proto: TCP (6), length: 47) 192.168.1.1.5463 > 192.168.8.8.3421: P, cksum 0xa89d (correct), 1:8(7) ack 1 win 5840

5. 5.220509 IP (tos 0x0, ttl 50, id 20038, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xb618 (correct), 1:1(0) ack 8 win 5840

6. 5.220591 IP (tos 0x0, ttl 50, id 20041, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: **F**, cksum 0xae04 (correct), 2068:2068(0) ack 8 win 5840

7. 5.220607 IP (tos 0x10, ttl 64, id 46021, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb618 (correct), 8:8(0) ack 1 win 5840

8. 5.223374 IP (tos 0x0, ttl 50, id 20040, offset 0, flags [DF], proto: TCP (6), length: 647) 192.168.8.8.3421 > 192.168.1.1.5463: P, cksum 0xe4c5 (correct), 1461:2068(607) ack 8 win 5840

9. 5.223381 IP (tos 0x10, ttl 64, id 46022, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb618 (correct), 8:8(0) ack 1 win 5840

10. 5.229617 IP (tos 0x0, ttl 50, id 20039, offset 0, flags [DF], proto: TCP (6), length: 1500) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xbf1b (correct), 1:1461(1460) ack 8 win 5840

11. 5.229632 IP (tos 0x10, ttl 64, id 46023, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xa29c (correct), 8:8(0) ack 2069 win 8760

12. 5.231280 IP (tos 0x10, ttl 64, id 46024, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: **F**, cksum 0xa29b (correct), 8:8(0) ack 2069 win 8760

13. 5.452312 IP (tos 0x0, ttl 50, id 20042, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xae03 (correct), 2069:2069(0) ack 9 win 5840

1 A captura em apreço apresenta uma conexão TCP completa, com todos os segmentos envolvidos no estabelecimento e encerramento da conexão.

2 Há elementos característicos de tráfego interativo, em que a janela deslizante não é completamente preenchida. Entretanto, há segmentos com o tamanho máximo possível.

3 Na captura em questão, a recepção de segmentos fora de ordem deve-se à ocorrência de retransmissão por perda de pacotes.

COMENTÁRIOS

1 771929 IP (tos 0x10, ttl 64, id 46018, offset 0, flags [DF], proto: TCP (6), length: 60) 192.168.1.1.5463 > 192.168.8.8.3421: S, cksum 0x1db2 (correct), 0:0(0) win 5840 <mss 1460,sackOK,timestamp 2538826 0,nop,wscale 6>

2. 1.994556 IP (tos 0x0, ttl 50, id 20037, offset 0, flags [DF], proto: TCP (6), length: 44) 192.168.8.8.3421 > 192.168.1.1.5463: S, cksum 0x9e62 (correct), 0:0(0) ack 1160318601
win 5840 <mss 1460>

3. 1.994605 IP (tos 0x10, ttl 64, id 46019, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb61f (correct), 1:1(0) ack 1 win 5840

4. 4.909380 IP (tos 0x10, ttl 64, id 46020, offset 0, flags [DF], proto: TCP (6), length: 47) 192.168.1.1.5463 > 192.168.8.8.3421: P, cksum 0xa89d (correct), 1:8(7) ack 1 win 5840

5. 5.220509 IP (tos 0x0, ttl 50, id 20038, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xb618 (correct), 1:1(0) ack 8 win 5840

6. 5.220591 IP (tos 0x0, ttl 50, id 20041, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: F, cksum 0xae04 (correct), 2068:2068(0) ack 8 win 5840

7. 5.220607 IP (tos 0x10, ttl 64, id 46021, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb618 (correct), 8:8(0) ack 1 win 5840

8. 5.223374 IP (tos 0x0, ttl 50, id 20040, offset 0, flags [DF], proto: TCP (6), length: 647) 192.168.8.8.3421 > 192.168.1.1.5463: P, cksum 0xe4c5 (correct), 1461:2068(607) ack 8 win 5840

9. 5.223381 IP (tos 0x10, ttl 64, id 46022, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xb618 (correct), 8:8(0) ack 1 win 5840

10. 5.229617 IP (tos 0x0, ttl 50, id 20039, offset 0, flags [DF], proto: TCP (6), length: 1500) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xbf1b (correct), 1:1461(1460) ack 8 win 5840

11. 5.229632 IP (tos 0x10, ttl 64, id 46023, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: ., cksum 0xa29c (correct), 8:8(0) ack 2069 win 8760

12. 5.231280 IP (tos 0x10, ttl 64, id 46024, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.1.1.5463 > 192.168.8.8.3421: F, cksum 0xa29b (correct), 8:8(0) ack 2069 win 8760

13. 5.452312 IP (tos 0x0, ttl 50, id 20042, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.8.8.3421 > 192.168.1.1.5463: ., cksum 0xae03 (correct), 2069:2069(0) ack 9 win 5840

1. Correto. O peguinha consistiu em embaralhar o segundo flag FIN para que o candidato imaginasse que a conexão não havia sido encerrada em sua totalidade. A chegada fora de ordem do FLAG F (destacado em vermelho) aparece no trecho 6.

2. Correto. No tráfego interativo, os dados a serem transmitidos geralmente não atingem a limitação da rede, pois a fonte geradora dos dados geralmente produz menor quantidade de informações do que a rede pode transmitir. Assim, são praticamente imperceptíveis problemas de desempenho ocasionados nesta situação. Problemas de desempenho geralmente são notados em uma situação onde o tráfego acontece em rajadas, ou seja, possui momentos de baixos e altos volumes de dados a serem transmitidos. Isso porque a quantidade de informações a serem transmitidas supera a capacidade de transmissão da rede, o que utiliza assim o conceito de buferização, onde na maioria do tempo há informações a espera para serem transmitidas.

3. Errado. Não houve perda de pacotes e, portanto, não houve retransmissão. Apenas pacotes que chegaram fora de ordem (observe que não há id repetidos).

INMETRO 2009

0.544624 00:e0:7d:ab:b8:1e > 00:0c:29:46:25:33, ethertype IPv4 (0x0800), length 66: (tos 0x0, ttl 64, id 10105, offset 0, flags [DF], proto: TCP (6), length: 52) 192.168.72.4.22 > 192.168.72.1.1821: P, cksum 0xf520 (correct), 610292262:610292274(12) ack 1867580146 win 6432
0.543601 00:0c:29:46:25:33 > 00:00:21:12:c7:da, ethertype IPv4 (0x0800), length 66: (tos 0x0, ttl 64, id 10105, offset 0, flags [DF], proto: TCP (6), length: 52) 192.168.72.4.22 > 192.168.72.1.1821: P, cksum 0xf520 (correct), 0:12(12) ack 1 win 6432
0.699554 00:00:21:12:c7:da > 00:0c:29:46:25:33, ethertype IPv4 (0x0800), length 60: (tos 0x0, ttl 128, id 36787, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.72.1.1821 > 192.168.72.4.22: ., cksum 0x6b48 (correct), ack 12 win 17197
0.666090 00:0c:29:46:25:33 > 00:e0:7d:ab:b8:1e, ethertype IPv4 (0x0800), length 60: (tos 0x0, ttl 128, id 36787, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.72.1.1821 > 192.168.72.4.22: ., cksum 0x6b48 (correct), ack 12 win 17197
2.791857 00:00:21:12:c7:da > 00:0c:29:46:25:33, ethertype IPv4 (0x0800), length 454: (tos 0x0, ttl 128, id 36790, offset 0, flags [DF], proto: TCP (6), length: 440) 192.168.72.1.1821 > 192.168.72.4.22: P 1:401(400) ack 12 win 17197
2.768640 00:0c:29:46:25:33 > 00:e0:7d:ab:b8:1e, ethertype IPv4 (0x0800), length 454: (tos 0x0, ttl 128, id 36790, offset 0, flags [DF], proto: TCP (6), length: 440) 192.168.72.1.1821 > 192.168.72.4.22: P 1:401(400) ack 12 win 17197
2.822118 00:e0:7d:ab:b8:1e > 00:0c:29:46:25:33, ethertype IPv4 (0x0800), length 60: (tos 0x0, ttl 64, id 10106, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.72.4.22 > 192.168.72.1.1821: ., cksum 0x8f95 (correct), ack 401 win 7504
2.793644 00:0c:29:46:25:33 > 00:00:21:12:c7:da, ethertype IPv4 (0x0800), length 60: (tos 0x0, ttl 64, id 10106, offset 0, flags [DF], proto: TCP (6), length: 40) 192.168.72.4.22 > 192.168.72.1.1821: ., cksum 0x8f95 (correct), ack 401 win 7504

Com base no trecho de captura de tráfego apresentado acima, julgue os seguintes itens.
1 Metade dos segmentos são retransmissões.
2 Apesar de não ter ocorrido no tráfego apresentado, a fragmentação seria nele permitida.
3 Assumindo o uso de portas convencionais, é correto afirmar que a captura envolve o serviço SSH.
4 Os segmentos de estabelecimento de conexão não estão presentes na captura mostrada.
5 A conexão exibe tráfego TCP interativo, sem que a janela deslizante seja usada totalmente.

COMENTÁRIOS

1. [E] Perceba que há ID com números repetidos, porém os endereços de enlace (MAC address) são diferentes, ou seja, o mesmo pacote trafegando em trechos diferentes (provavelmente há um roteador entre as sub-redes).
2. [E] o bit [DF] presente em todos os segmentos informa que a fragmentação não é permitida.
3. [C] O IP 192.168.72.4 utiliza a porta 22 que convencionalmente caracteriza um serviço SSH.
4. [C] Ausencia do flag SYN.
5. [C] Pequeno volume de dados caracteriza tráfego interativo.

Fragmentação IP

Teoricamente o tamanho do pacote IP pode variar até o limite máximo de 64 KB dependendo da carga útil e das opções incluídas. Todavia, no caminho até seu destino um pacote pode atravessar diferentes tipos de redes. As redes são heterogêneas admitindo diferentes tamanhos de quadros (frames). Então, na prática, o tamanho máximo do pacote IP também é definido em função do hardware de rede por onde o pacote irá transitar. O tamanho máximo que um quadro pode ter é chamado de MTU - Maximum Transmission Unit - Unidade Máxima de Transmissão. Cada padrão de rede irá possuir um valor de MTU diferente.

Quando um pacote, de certo tamanho, necessita atravessar uma rede que lida somente com pacotes de menor tamanho é necessário dividir a carga do pacote em vários quadros, adequados ao tamanho da MTU dessa rede. O processo de adequar a carga do pacote IP ao tamanho da MTU é chamado de fragmentação de pacotes. Em alguns textos a fragmentação de pacotes é definida como segmentação. O conceito é o mesmo, mas se tratando do protocolo IP, prefira o termo fragmentação.

A princípio, um pacote é encaminhado e entregue com o mesmo tamanho que foi gerado na origem (fragmentação local). Mas, como a rota até o destino é uma escolha do roteador, um pacote pode seguir por uma rede que necessite de mais fragmentação (fragmentação na Inter-rede). A fragmentação que ocorre na inter-rede é invisível para o módulo IP do host que enviou o pacote. Caso um pacote seja fragmentado, transmitido e remontado entre dois roteadores o módulo IP não será informado disto.

Assim, havendo necessidade de fragmentação na inter-rede os roteadores poderão fazê-la, já que atuam na camada de rede do modelo TCP/IP. Neste caso, os roteadores ficam obrigados a remontar os pacotes antes de entregá-los ao destino. A fragmentação da inter-rede é chamada de fragmentação transparente.

A fragmentação de pacotes que acontece no módulo IP é chamada de não-transparente. Isso significa que o IP pode enviar tanto fragmentos de pacotes quanto pacotes sem fragmentação. Na prática, isso significa também que cada roteador receberá e transmitirá tanto pacotes completos, adequados ao tamanho da MTU, quanto fragmentos de pacotes ajustados a MTU. Tecnicamente um fragmento é apenas um pacote idêntico aos demais. Na fragmentação não-transparente o protocolo IP é "consciente" de que ocorreu fragmentação no encaminhamento, sendo também responsável pela remontagem dos fragmentos no destino. Para tanto, o protocolo IP fornece alguns campos que garantem que os pacotes fragmentados sejam montados corretamente. Os campos que estão diretamente envolvidos na operação de fragmentação e remontagem de pacotes são: ***Identification (ID)***, ***Fragment Offset (offset)*** e os sinalizadores binários ***Don't Fragment (DF)*** e ***More Fragment (MF)***.

Havendo necessidade de fragmentação um mecanismo de fragmentação será usado para criar os fragmentos. Por exemplo, suponha um pacote que será fragmentado em dois fragmentos. Esse processo ocorre do seguinte modo: basicamente são criados dois novos datagramas, o conteúdo do cabeçalho do datagrama original é copiado para os cabeçalhos dos novos datagramas. Os campos identificação, endereço de origem e destino e o número do protocolo de transporte (para o qual a carga útil será entregue) permanecem inalterados, independente do número de fragmentos.

A carga útil do datagrama original é dividida em blocos de oito octetos (64 bits). Cada fragmento poderá carregar múltiplos blocos de 64 bits, limitados ao tamanho da MTU. Teoricamente um único pacote com 64 KB poderia ser dividido em até 8.192 fragmentos com 64 bits cada.

Entretanto, como os valores de MTU são superiores ao tamanho mínimo do fragmento dificilmente esse número de fragmentos é alcançado.

O último fragmento não necessita ter uma carga útil múltipla de 8 bytes. Por exemplo, suponha um pacote com 113 bytes que será encaminhado em dois fragmentos. O primeiro fragmento terá 56 bytes enquanto o último terá 57.

Este processo ocorrerá da seguinte maneira: a primeira porção dos dados (56 bytes) é inserida no primeiro fragmento, o campo *Total Lenght* é ajustado para o novo tamanho do datagrama, enquanto o sinalizador MF - *More Fragment* - é configurado em 1 então o fragmento é enviado.

A porção restante dos dados (57 bytes) é inserida no segundo fragmento, o campo Total Lenght é ajustado. O sinalizador MF permanece inalterado (zero), indicando que este é o último fragmento do pacote.

O campo *Fragment Offset* do primeiro pacote é configurado como zero. No segundo fragmento o valor de *Fragment Offset* será o valor do *Fragment Offset* do pacote anterior (0) somado ao valor do NFB - *Number of Fragment Block* - Número de Blocos dos Fragmentos (7). Neste exemplo, o valor do NFB é 7, pois a cada fragmento foram enviados sete bytes de carga útil (56 / 8 - exceto no último que serão enviados oito bytes). O NFB é um índice utilizado para remontar o pacote.

A importância da fragmentação, é que ela torna as redes mais homogêneas. Entretanto, fragmentação gera um maior volume de tráfego (overhead) na rede, pois multiplica cabeçalhos e gera um número maior de pacotes para serem tratados por um roteador. O gerenciamento da fragmentação pode ser necessário nos casos em que seja observada uma sobrecarga de carga em relação à largura de banda da rede.

Além disto, fragmentação mal gerenciada pode ser um ponto de vulnerabilidade na segurança da rede. Diversos ataques a segurança das redes utilizam a fragmentação para serem realizados. São exemplos de ataques de fragmentação: o ataque de pequenos fragmentos e o ataque de sobreposição de fragmentos.

TER/MT 2009
Em um enlace de comunicação de dados com MTU (maximum transmission unit) de 1.500 bytes, que conecta um roteador A a um roteador B, o roteador A recebe um datagrama de 6 kilobytes, a ser repassado ao roteador B. Esse enlace utiliza o protocolo IPv4, com cabeçalho padrão de 20 bytes, e permite a fragmentação. Com base nessas informações, é correto afirmar que:

A - o último fragmento recebido pelo roteador B tem o campo de flag do cabeçalho IP ajustado para 1.
B - o primeiro fragmento tem o valor de deslocamento igual ao valor do cabeçalho IP.
C - o segundo fragmento tem deslocamento de 185, desde que o primeiro fragmento tenha sido enviado com o MTU máximo.
D - são necessários quatro fragmentos para transferir os 6 kilobytes do datagrama original.
E- o campo de flag do cabeçalho IP contém zero para todos os fragmentos, exceto o último

Como vimos, o campo Frament Offset cada unidade representa 8 bytes/octetos de acordo com o NFB. Trata-se uma convenção adotada pela RFC 791. Isso indica que o **segundo fragmento** levará a **carga (ou payload)** a partir do pedaço que se inicia no byte 1480 (termino da carga do primeiro fragmento). O primeiro fragmento fica na posição zero. O segundo na posição 185. O terceiro na posição 370. E assim sucessivamente. (1480 / 8 = 185). Vejamos em mais detalhes:

É desejado a transmissão de 6144 bytes que deverá ser fragmentados numa MTU de 1500 bytes:
fragmento 1: 1500bytes
fragmento 2: 1480bytes + 20cabecalho
fragmento 3: 1480bytes + 20cabecalho
fragmento 4: 1480bytes + 20cabecalho
fragmento 5: 204bytes + 20cabecalho
Total = 6144bytes originais transmitidos + 80bytes de overhead (cabeçalhos extras)

Percebam que o primeiro fragmento utilizou o cabeçalho do pacote original (6KB). As partes subsequentes precisaram de cabeçalhos extras que foram inseridos pela camada de rede durante o processo de fragmentação.

Deslocamento = 0ffset
Quando os fragmentos chegarem em seu destino, o campo offset do IP irá organizar a remontagem do pacote descartando **todos os cabeçalhos IP** incluindo o cabeçalho IP do primeiro fragmento, contabilizando apenas as informações do PAYLOAD.
IMPORTANTE: O offset não descarta o cabeçalho ICMP ou UDP que porventura esteja encapsulado na camada 3.
O campo offset tem 13 bits medidos em múltiplos de 8 bytes. Offset=1 significa deslocamento de 8bytes, offset = 2 deslocamento de 16 bytes, offset = 185 deslocamento de 1480 bytes e assim sucessivamente.

Logo, o primeiro fragmento terá offset 0 e o segundo fragmento terá offset 185 . [Gab. C]

TEORIA DA FRAGENTAÇÃO

A MTU ou o tamanho máximo de um datagrama IP para a Ethernet é de 1500 bytes. Se um datagrama for maior que isso e precisar atravessar uma rede Ethernet, ele exigirá a fragmentação por um roteador. Os fragmentos continuam viajando até seu destino, onde o host de destino os remonta. Os fragmentos podem até mesmo tornar-se mais fragmentados, se cruzarem uma MTU menor que o tamanho do fragmento. Embora a fragmentação seja um evento perfeitamente normal, é possível interceptar fragmentos com o intuito de evitar a detecção por roteadores e sistemas de detecção de intrusão que não lidam bem com fragmentação. Que tipo de informação os fragmentos precisam conduzir para que o host de destino os remonte ao estado desfragmentado original? A lista a seguir responde a essa pergunta:

- Cada fragmento precisa estar associado aos outros fragmentos através de um número de identificação de fragmento comum. Esse número é clonado de um campo no cabeçalho IP conhecido como número de identificação IP, que é também chamado de ID de fragmento.

- Cada fragmento precisa conduzir a informação de qual é o seu lugar, ou deslocamento, no pacote desfragmentado original.

- Cada fragmento precisa informar a extensão dos dados transportados no fragmento.

- Finalmente, cada fragmento precisa saber se mais fragmentos seguem o atual. Isso é feito usando-se o flag More Fragments (MF).

Essas informações são inseridas no cabeçalho IP , que é colocado em um datagrama IP seguido de um fragmento encapsulado. Como você aprendeu, todo tráfego TCP/IP precisa estar envolto dentro do IP por ser o protocolo responsável pela distribuição de pacotes.

Visualizando a fragmentação: ver é entender

Esta seção usa a Ethernet como o meio de camada de vínculo para demonstrar o empacotamento de datagramas. A Figura 3.1 descreve a configuração de um datagrama que não está fragmentado. Como já foi

dito, um datagrama viajando na Ethernet possui uma MTU de 1500 bytes. Cada datagrama precisa ter um cabeçalho IP, que normalmente é de 20 bytes, mas pode ser maior, se estiverem incluídas opções de IP.

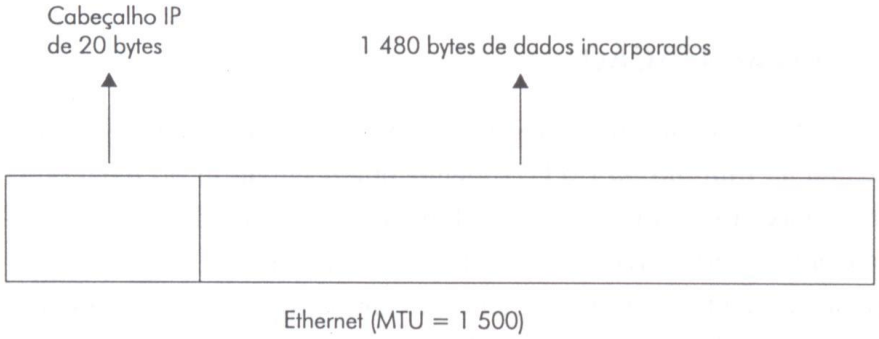


Figura 3.1 Empacotamento de datagrama Ethernet.

Recapitulando rapidamente, lembre-se de que o cabeçalho IP contém informações como números de IP de origem e de destino. Essas informações são consideradas a parte de 'rede' do datagrama IP, pois os roteadores usam as informações encontradas no cabeçalho IP para direcionar o datagrama ao seu destino. Alguns tipos de dados são encapsulados após o cabeçalho IP. Esses dados podem ser um protocolo IP como TCP, UDP ou ICMP. Se esses dados fossem TCP, por exemplo, eles incluiriam um cabeçalho TCP e dados TCP. A Figura 3.2 mostra um datagrama de 4028 bytes. Essa é uma requisição de eco ICMP destinada a uma rede Ethernet com MTU de 1500 bytes. Essa é uma requisição de eco ICMP grande, que não é representativa do tráfego normal, mas é usada para ilustrar, como ocorre na fragmentação. Assim, o datagrama de 4028 bytes terá de ser dividido em fragmentos de 1500 bytes ou menos. Cada um desses pacotes fragmentados de 1500 bytes terá um cabeçalho IP de 20 bytes como o fragmento inicial, deixando 1480 bytes no máximo para os dados de cada fragmento. As seções a seguir examinam o conteúdo de cada um dos três fragmentos individuais.

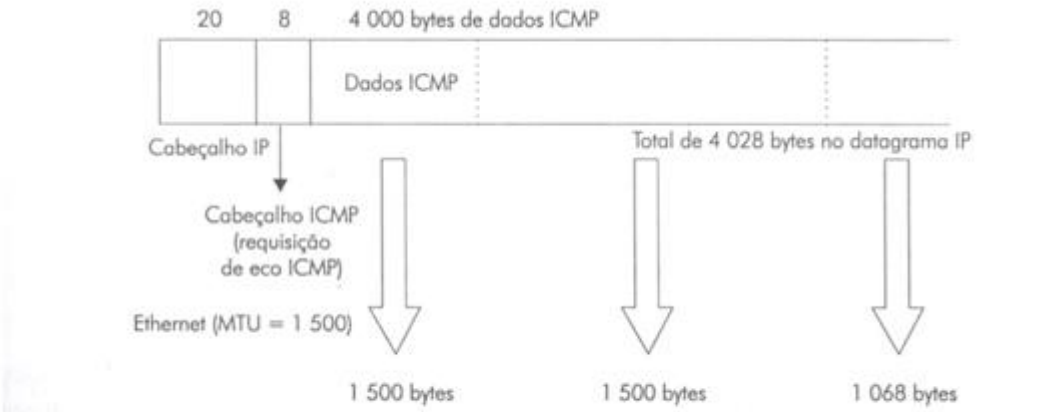


Figura 3.2 Fragmento original de 4 028 bytes, dividido em três fragmentos de 1 500 bytes ou menos.

O ID de fragmento/número de identificação IP

O número de identificação IP é um campo de 16 bits encontrado no cabeçalho IP de todos os datagramas. Esse número identifica unicamente cada datagrama enviado pelo host. Tipicamente, esse valor aumenta de um para cada datagrama enviado por esse host. Quando o datagrama se torna fragmentado, todos os fragmentos criados desse datagrama contêm esse mesmo número de identificação IP ou ID de fragmento. A saída TCPdump, a seguir, mostra um número de identificação IP de 202 para essa saída desfragmentada :

ping.com > 192.168,244.2: icmp: echo request (ttl 240. id 202)

Se esse datagrama fosse se tornar fragmentado no caminho para o seu destino, todos os fragmentos criados do datagrama iriam compartilhar um ID de **fragmento de 202**. Essa saída TCPdump foi gerada usando-se a opção –w. Essa é uma opção que diz para relacionar, no final da saída, o valor do campo Time to Live e o número de identificação ID do fragmento.

Todos a bordo do trem de fragmento

Volte sua atenção para o fragmento inicial no trem de fragmento mostrado na Figura 3.3. O cabeçalho IP 'original' será clonado para incluir o mesmo número de identificação IP no primeiro fragmento e nos fragmentos seguintes.

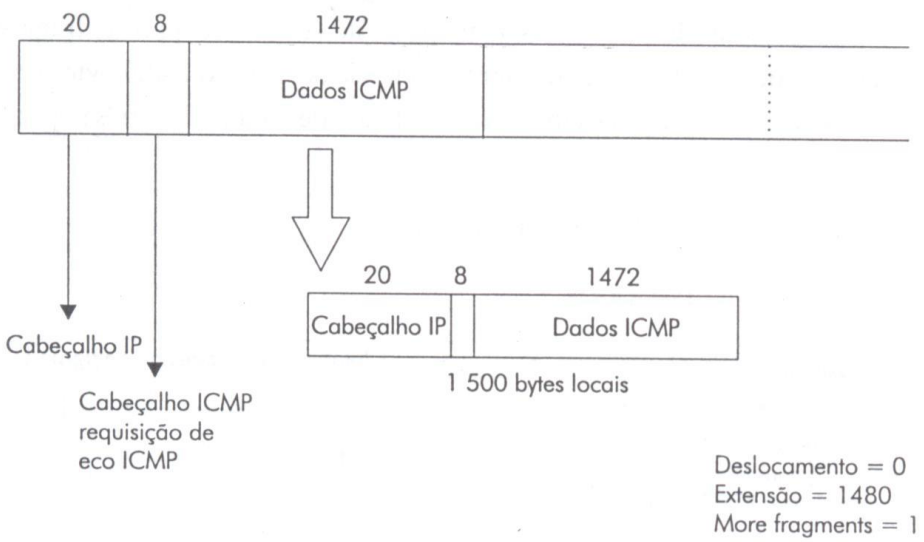


Figura 3.3 O fragmento máquina.

O primeiro fragmento é o único que seguirá com o cabeçalho da mensagem ICMP. Esse cabeçalho não é clonado nos fragmentos associados subseqüentes, e esse conceito de apenas o primeiro fragmento identificar a natureza do trem de fragmento é importante, como você logo verá. O primeiro fragmento possui um deslocamento 0, uma extensão de 1480 bytes, 1472 bytes de dados e 8 bytes de cabeçalho ICMP; e, como seguem mais fragmentos, o flag More Fragments está definido.

A Figura 3.4 explica a configuração do primeiro fragmento no trem de fragmento. Os primeiros 20 bytes dos 1500 bytes são o cabeçalho IP. Os próximos 8 bytes são o cabeçalho ICMP. Lembre-se de que essa foi uma

requisição de eco ICMP que tinha um cabeçalho de 8 bytes em seu pacote original. Os 1472 bytes restantes são para os dados ICMP.

Além dos campos normais conduzidos no cabeçalho IP, como IP e protocolo de origem e destino (neste exemplo de ICMP), existem campos especificamente para a fragmentação. O ID de fragmento com um valor de 21223 é o vínculo comum para todos os fragmentos no trem de fragmento. Existe um campo conhecido como o flag More Fragments (mais fragmentos), que indica que outros fragmentos seguem o atual. Nesse primeiro fragmento, o flag é definido como 1 para indicar que mais fragmentos se seguem. Também, o deslocamento dos dados contidos nesse fragmento em relação aos dados do datagrama inteiro precisa ser armazenado. Para o primeiro registro, o deslocamento é 0. Finalmente, a extensão dos dados conduzidos neste fragmento é armazenada como a extensão de dados - que, neste fragmento é de 1480. Essa extensão é o cabeçalho ICMP de 8 bytes seguido dos primeiros 1472 bytes dos dados ICMP.

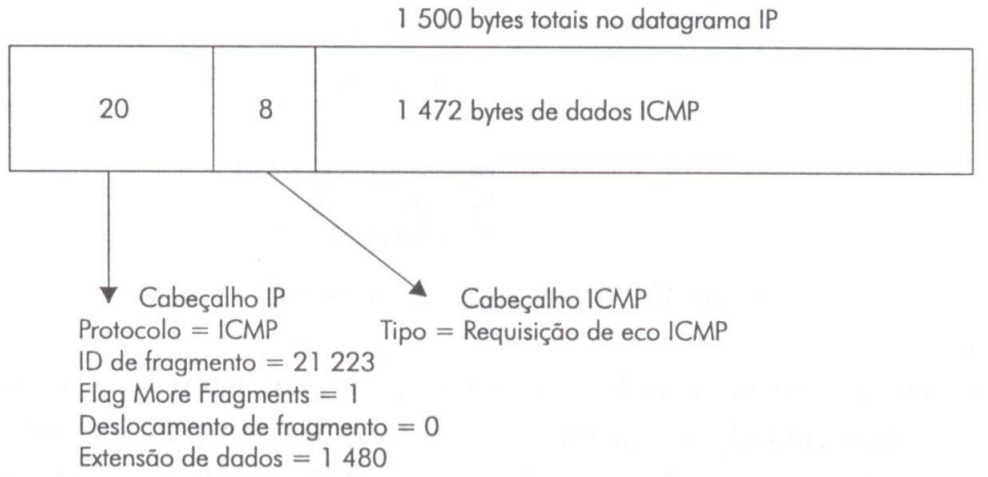


Figura 3.4 Interior do fragmento máquina.

O tratamento vagão-restaurante

Veja a Figura 3.5 para analisar o próximo fragmento no trem de fragmento. Um cabeçalho IP é clonado do cabeçalho 'original' com um ID de fragmento idêntico, e a maioria dos outros dados no cabeçalho IP (como os números de origem e destino) é repetida no novo cabeçalho. Incorporados após esse novo cabeçalho IP, estão 1480 bytes de dados ICMP . Como você pode ver, o segmento fragmento possui um deslocamento de

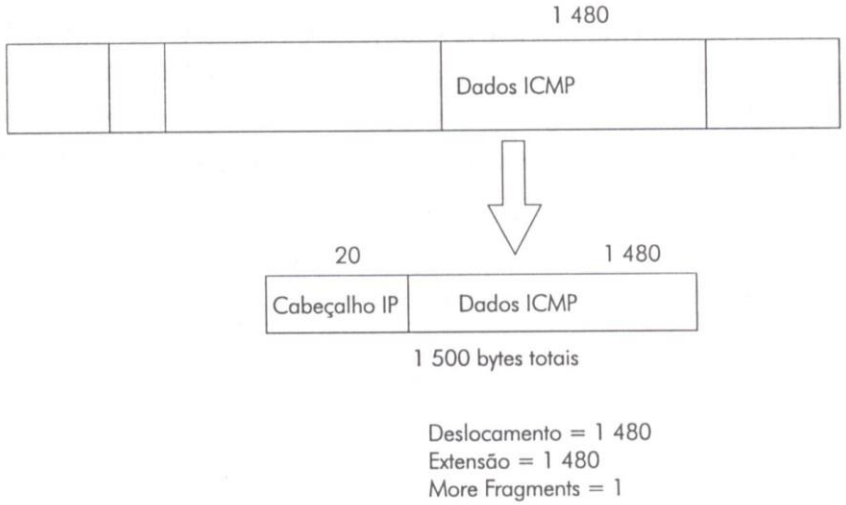


Figura 3.5 O fragmento vagão-restaurante.

1480 e uma extensão de 1480 bytes; e, como mais um fragmento se segue, o flag More Fragments está definido.

Continuando com a fragmentação na Figura 3.6, você pode examinar o datagrama IP conduzindo o segundo fragmento. Como em todos os fragmentos desse trem de fragmento, esse requer um cabeçalho IP de 20 bytes. Novamente, o protocolo no cabeçalho indica ICMP. O ID de fragmento permanece 21233 e o flag MF está ativado, porque outro fragmento se segue. O deslocamento é 1480 bytes na parte dos dados da mensagem ICMP original. O fragmento anterior ocupou os primeiros 1480 bytes. Esse fragmento também possui 1480 bytes de extensão e é composto inteiramente de bytes de dados ICMP.

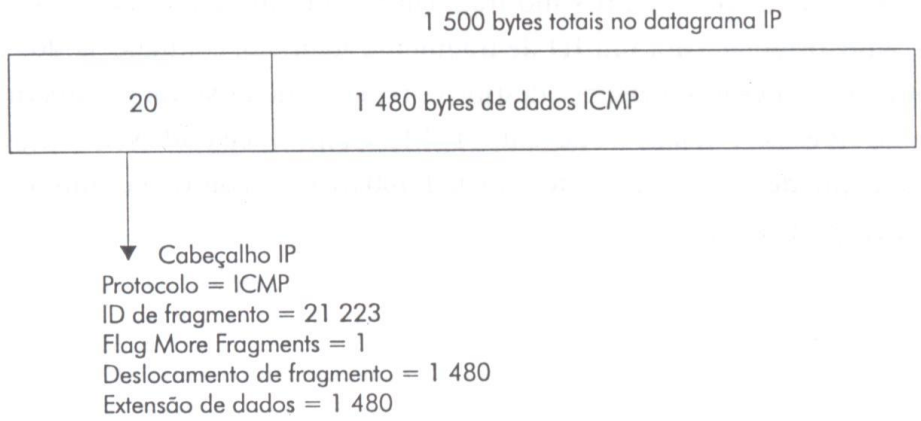
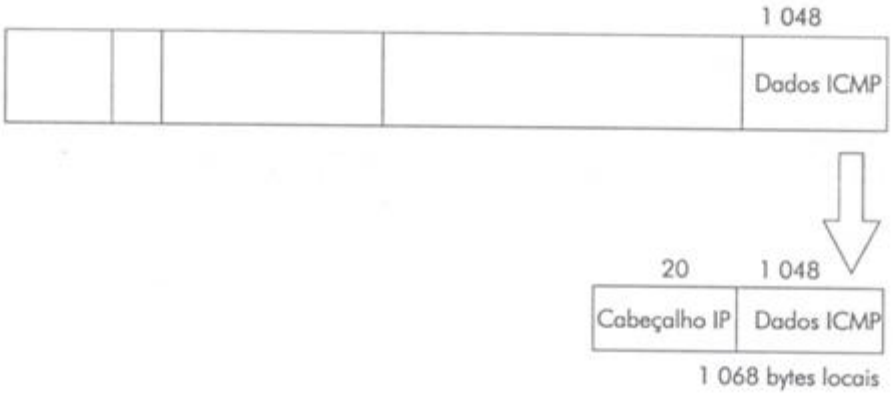


Figura 3.6 Interior do fragmento vagão-restaurante.

Vale a pena repetir que o cabeçalho ICMP segue apenas com o primeiro fragmento, ou seja, não é clonado com o restante dos dados ICMP. Isso significa que, se esse fragmento fosse examinado isoladamente, não poderia saber o tipo de mensagem ICMP - nesse caso, uma requisição de eco ICMP. Esse é um aspecto importante em relação aos dispositivos de filtragem de pacote.

O fragmento do vagão de carga

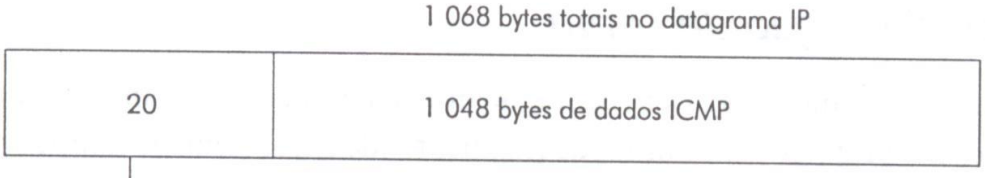
Examine o último fragmento do trem de fragmento na Figura 3.7. Novamente, um cabeçalho IP é clonado do cabeçalho 'original' com um ID de fragmento idêntico, e outros campos são repetidos no novo cabeçalho. Os últimos 1048 bytes de dados ICMP são incorporados neste novo datagrama IP. Observe que o terceiro fragmento possui um deslocamento de 2960 e uma extensão de 1048 bytes; e, como mais nenhum fragmento se segue, o flag MF é zero.



Deslocamento = 2 960
Extensão = 1 048
More fragments = 0

Figura 3.7 O fragmento vagão de carga.

A Figura 3.8 descreve o último fragmento no trem de fragmento. Novamente, 20 bytes são reservados para o cabeçalho IP. Os bytes de dados ICMP restantes são conduzidos na parte de dados desse fragmento. O ID de fragmento é 21 223, e o flag MF não está definido, porque esse é o último fragmento. O deslocamento é 2960 (a soma dos dois fragmentos de 1480 bytes anteriores). Apenas l 048 bytes de dados são conduzidos nesse fragmento formado inteiramente pelos bytes de mensagem ICMP restantes. Esse fragmento, assim como o segundo, não possui cabeçalho ICMP e, portanto, não possui tipo de mensagem ICMP para indicar que essa é uma requisição de eco ICMP.



Cabeçalho IP
Protocolo = ICMP
ID de fragmento = 21 223
Flag More Fragments = 0
Deslocamento de fragmento = 2 960
Extensão de dados = 1 048

Figura 3.8 Interior do fragmento vagão de carga.

Visualizando a fragmentação com o TCPdump

Observe a seguinte saída TCPdump. Como você vê, os três registros diferentes representam os três fragmentos discutidos anteriormente. Isso significa que o host que executa o TCPdump coletou a requisição de eco ICMP depois que a fragmentação ocorreu.

ping.com > myhost,com: icmp: echo request (frag 2l223:l48O@0+)

ping.com > myhost-com-. (frag 21223: l480@148O+)

ping.com > myhost-com: (frag 2l223:l048@2960)

A primeira linha mostra *ping.com* enviando uma requisição de eco ICMP para *myhost.com.* A razão pela qual o TCPdump pode identificar isso como uma requisição de eco ICMP é que o primeiro fragmento contém o cabeçalho ICMP de 8 bytes, o que o identifica como uma requisição de eco ICMP .
Agora, observe a notação da fragmentação no lado direito do registro. A convenção do TCPdump para exibir saída fragmentada é que a palavra ***frag*** aparece seguida do ID de fragmento (21223, neste exemplo), seguida de dois pontos. Segue-se a extensão de dados no fragmento atual, 1480, seguida de um sinal de arroba (@) e, depois, aparece o deslocamento nos dados (0, porque este é o primeiro fragmento). O sinal de adição indica que o flag More Fragments está definido. Esse fragmento sabe qual é a finalidade do tráfego, que esse é o primeiro fragmento, que mais fragmentos se seguem, mas não sabe o que ou quantos seguem.

O segundo registro é um pouco diferente. Repare que não existe um rótulo de requisição de eco ICMP. Isso é porque não existe um cabeçalho ICMP para dizer que tipo de tráfego ICMP é esse. O cabeçalho IP ainda terá o campo Protocol definido em ICMP, mas isso é tudo o que você pode dizer olhando apenas esse fragmento. Observe que a saída TCPdump relaciona o ID de fragmento de 21223, a extensão de dados atual é 1480 e o deslocamento é 1480. O sinal de adição significa que o flag MF está definido. Esse fragmento possui uma afiliação, mas é essencialmente incógnito quanto ao seu propósito - parece o primeiro dia na escola.

A última linha é bastante semelhante à Segunda quanto ao formato. Ela mostra o mesmo ID de fragmento de 21223, possui uma extensão de 1048 e um deslocamento de 2960. Entretanto, nenhum flag More Fragments aparece no último registro, como você poderia esperar. Esse fragmento possui uma afiliação, não possui nenhuma indicação de propósito e quaisquer seguidores.

A fragmentação e os dispositivos de filtragem de pacote

Esta seção aborda a fragmentação e como um dispositivo de filtragem de pacote, como um roteador ou firewall, pode lidar com ela. O problema surge quando um dispositivo tenta bloquear dados fragmentados.

Como apenas o primeiro fragmento de um trem de fragmento conterá qualquer tipo de cabeçalho de protocolo, como TCP, UDP ou ICMP, apenas esse fragmento é impedido de entrar em uma rede protegida por um dispositivo de filtragem de pacote incapaz de examinar o estado de um campo de cabeçalho. O termo ***estado*** aqui significa que parece óbvio que qualquer fragmento compartilhando o número de identificação IP do fragmento bloqueado também deve ser bloqueado. Porém, alguns dispositivos de filtragem de pacote não mantêm essa informação. Esses dispositivos míopes olham cada fragmento como uma entidade individual e não o associam aos pacotes anteriores ou subseqüentes.

Por intuição, se essa não é uma arquitetura particularmente boa, por que ela é usada? Pense no trabalho necessário para manter o estado. Isso significa que cada fragmento precisa ser examinado e armazenado; isso é custoso em termos de tempo, processamento e memória. Posteriormente, os fragmentos precisam ter o acesso concedido ou negado, o que consome ainda mais recursos. E muito mais simples ter uma arquitetura atômica que analise pacote por pacote.

Se um determinado pacote não corresponde ao critério de bloqueio - neste caso, devido à ausência de cabeçalho de protocolo, sua entrada na rede é permitida. Os datagramas TCP ou UDP fragmentados só podem conter suas respectivas informações de cabeçalho no primeiro fragmento. As decisões de bloqueio normalmente são baseadas nas informações de cabeçalho, como as portas de destino TCP ou UDP. Isso

significa que TCP e UDP fragmentados são suscetíveis às mesmas deficiências de um dispositivo de filtragem de pacote sem informações de estado.

Um último fator a ser lembrado é que, como o IP não é um protocolo seguro, é provável que seja perdido o primeiro fragmento que contém as informações de cabeçalho de protocolo. Quando isso ocorre, o dispositivo de filtragem de pacote tem ainda mais trabalho para permitir ou negar tráfego. Na verdade, **quando um dos fragmentos não chega ao seu destino, todos precisam ser enviados novamente.**

O flag Don't Fragment

Em algumas saídas TCPdump apresentadas, você pode ter observado as letras DF entre parênteses. Isso significa que o fag Don't Fragment (não fragmente) está definido. Nenhuma surpresa aqui; como o nome implica, se esse flag estiver definido, não ocorrerá fragmentação no datagrama. Se esse flag estiver definido, e o datagrama cruzar uma rede onde a fragmentação é exigida, o roteador descobre isso, descarta o datagrama e envia uma mensagem de erro ICMP de volta ao host emissor.

A mensagem de erro ICMP contém a MTU da rede que exigiu a fragmentação. Alguns hosts enviam intencionalmente um datagrama inicial através da rede, com o flag DF definido como um meio de descobrir a MTU de uma determinada origem ou destino. Se a mensagem de erro ICMP é retornada com a MTU menor, o host então empacota todos os datagramas designados a esse destino em unidades pequenas o suficiente para evitar a fragmentação. Isso freqüentemente é usado com o TCP por ser um protocolo que exige muito overhead. A fragmentação pode causar ineficiência, pois, se um fragmento é perdido, todos precisam ser reenviados; esse é um motivo de se desejar evitar a fragmentação. Como você pode imaginar, um usuário mal-intencionado também pode usar esse expediente para descobrir a MTU de um segmento da sua rede, a fim de usá-la futuramente para explorações de fragmentação. O usuário poderia interceptar datagramas com diferentes extensões e com o flag DF definido, e observar quando uma mensagem de erro ICMP é recebida. Isso considera que rede-alvo não tenha impedido a mensagem de erro ICMP de ser enviada.

router.ru > mail-mysite.com: icmp: host-ru unreachable - need to frag (mtu 308) (DF)

A saída TCPdump acima mostra uma mensagem ICMP na qual um roteador descobriu que a fragmentação era necessária, mas o flag Don't Fragment estava definido. O estímulo para essa resposta foi que *mail.mysite.com* tentou enviar um datagrama maior que 308 bytes para host.ru com o flag DF def'lnido. router.ru descobre que o datagrama precisa atravessar uma rede menor com uma MTU de 308 bytes e a fragmentação é necessária.

Quando router.ru examina o registro, descobre que o flag DF está definido, e uma mensagem ICMP é enviada de volta a mail.mysite.com, informando-o do problema. Agora, mail.mysite.com precisa empacotar os datagramas para serem menores que a MTU de 308, de modo que a fragmentação não ocorra ou precisa remover o flag DF para que a fragmentação possa ocorrer e, depois, reenviar o datagrama.

TCU2009

1. 0.285156 IP (tos 0x0, ttl 128, id 8313, offset 2944, flags [ ], proto: ICMP (1), length: 84) 10.1.1.100 > 10.1.1.200: icmp
2. 0.285156 IP (tos 0x0, ttl 128, id 8313, offset 1472, flags [+], proto: ICMP (1), length: 1492) 10.1.1.100 > 10.1.1.200: icmp
3. 0.285156 IP (tos 0x0, ttl 128, id 8313, offset 0, flags [+], proto: ICMP (1), length: 1492) 10.1.1.100 > 10.1.1.200: ICMP echo request, id 768, seq 2560, length 1472
4. 0.525390 IP (tos 0x0, ttl 250, id 9564, offset 2960, flags [ ], proto: ICMP (1), length: 68) 10.1.1.200 > 10.1.1.100: icmp
5. 0.546875 IP (tos 0x0, ttl 250, id 9564, offset 0, flags [+], proto: ICMP (1), length: 764) 10.1.1.200 > 10.1.1.100: ICMP echo reply, id 768, seq 2560, length 744
6. 0.570312 IP (tos 0x0, ttl 250, id 9564, offset 744, flags [+], proto: ICMP (1), length: 756) 10.1.1.200 > 10.1.1.100: icmp
7. 0.591796 IP (tos 0x0, ttl 250, id 9564, offset 1480, flags [+], proto: ICMP (1), length: 764) 10.1.1.200 > 10.1.1.100: icmp
8. 0.615234 IP (tos 0x0, ttl 250, id 9564, offset 2224, flags [+], proto: ICMP (1), length: 756) 10.1.1.200 > 10.1.1.100: icmp

1 Se utilizarem a mesma máscara de rede, qualquer que seja, então os hosts envolvidos na captura de tráfego estarão na mesma sub-rede.

2 A chegada fora de ordem dos pacotes de resposta deve-se à retransmissão de alguns deles ao longo do percurso.

3 É consistente com a captura que há quatro nós intermediários entre os hosts nela presentes.

4 É consistente com a captura que o processo de fragmentação tenha sido aplicado mais de uma vez nos pacotes de resposta.

COMENTÁRIOS

1. [E] Até a máscara /24 os hosts estariam na mesma sub-rede, porém a partir da máscara /25 isso não seria verdade. Outra forma de responder o item é enxergar o processo de fragmentação. Como e quando esta ocorre? Em enlaces com diferentes valores de MTU! Se eles não estão no mesmo enlace, devem existir roteadores entre eles, logo, eles não podem estar na mesma (sub) rede!

2 [E] Como checar a chegada fora de ordem? Pelo *offset*! Como identificar retransmissão? Identifique no *payload* a string 'Houston, we have a problem!' Brincadeira a parte, estamos falando de fragmentos IP (ICMP), onde não existe retransmissão! Em caso de perda de algum fragmento, todo pacote precisa ser retransmitido.

3 [C] O campo TTL admite os seguintes valores iniciais: 64,128 ou 255. O echo request usa o valor 128 e o echo reply valor 255. Perceba que a resposta do ping veio com um TTL de 250. Echo reply é a pilha TCP/IP do Sistema Operacional pingado quem dá as cartas. Uma característica destes SOs é o uso de TTLs padrão, geralmente 64 (Linux), 128 (Windows) ou 255 (Unix). Conclusão: houve 5 decrementos do TTL. (255 -250).

4 [C] Perceba os fragmentos 5-8 com tamanhos (*length*) diferentes: 764 e 756.

Curiosidade/falha encontrada:
O p3r1t0f3d3r4l levantou os seguintes questionamentos sobre esta questão do TCU:

1. Existem 5 (e não 4) nós intermediários entre as máquinas comunicantes. Quando o Cespe quer dizer "pelo menos" ele diz, mas aqui não é o caso.

2. A captura deveria ter ocorrido dentro da rede de 10.1.1.100 (não há decremento do TTL), entretanto, se isso fosse verdade absoluta, **não haveria motivos para o fragmento final se apresentar entes dos demais**, uma vez que este traço é característico dos pacotes quando percorrem caminhos distintos. Nesse caso o TTL deveria ser 128 menos alguma coisa.

Na real, acredito que alguém tentou editar um arquivo de captura para não ficar 100 por cento igual e cometeu alguns equívocos. Como a grande maioria tem deficiência na compreensão de capturas a coisa ficou meio esquecida.

[Gustavo/p3r1t0f3d3r4l]

INMETRO2009

0.745185 IP (tos 0x0, ttl 128, id 8591, offset 0, flags [+], proto: ICMP (1), length: 1500) 192.168.0.135 > 10.0.1.1: ICMP echo request, id 1024, seq 18688, length 1480. 0.745209 IP (tos 0x0, ttl 128, id 8591, offset 1480, flags [ ], proto: ICMP (1), length: 548) 192.168.0.135 > 10.0.1.1: icmp 0.784339 IP (tos 0x0, ttl 58, id 7016, offset 0, flags [+], proto: ICMP (1), length: 1500) 10.0.1.1 > 192.168.0.135: ICMP echo reply, id 1024, seq 18688, length 1480 0.789152 IP (tos 0x0, ttl 58, id 7016, offset 1480, flags [ ], proto: ICMP (1), length: 548) 10.0.1.1 > 192.168.0.135: icmp
Considerando o trecho de captura de tráfego acima, julgue os próximos itens. 1 O MTU, como percebido pela camada do protocolo IP, é 1500 bytes, em ambos os hosts. 2 Ocorreu fragmentação com pacotes sendo entregues fora de ordem. 3 O ICMP carrega 2028 bytes, tanto no request como no reply. 4 É consistente com a captura que existam pelo menos cinco nós intermediários entre os dois hosts. 5 Os protocolos presentes oferecem entrega garantida.

COMENTÁRIOS

1 **[C]** Ver campo length dos primeiros fragmentos (flag[+]).
2 **[E]** Ocorreu fragmentação, mas os pacotes foram entregue na ordem correta: flag[+] (mais fragmentos) em primeiro e flag[ ] (ultimo fragmento) no final.
3 **[E]** 1500 + 548 - 20 - 20 – 8 = **2000 bytes** ICMP (lembre-se de excluir os 8 bytes do cabeçalho ICMP). Observe que estaria correto dizer que o **IP** carrega 2008 bytes.
4 **[C]** Campo TTL com dois valores: 64 (Linux), 128 (Windows) O *echo request* usa o valor 128 e o *echo reply* valor 64. Perceba que a resposta do ping veio com um TTL de 58, concluímos, portanto, que houve 6 decrementos do TTL. (64 -58).
5 **[E]** Não existe entrega garantida na camada de rede do protocolo IP.

SERPRO 2008

1 0.285156 IP (tos 0x0, ttl 128, id 3138, offset 0, flags [+], proto: ICMP (1), length: 1492) 10.1.1.1 > 10.2.2.2: ICMP echo request, id 768, seq 2560, length 1472
2 0.285156 IP (tos 0x0, ttl 128, id 3138, offset 1472, flags [+], proto: ICMP (1), length: 1492) 10.1.1.1 > 10.2.2.2: icmp
3 0.285156 IP (tos 0x0, ttl 128, id 3138, offset 2944, flags [ ], proto: ICMP (1), length: 84) 10.1.1.1 > 10.2.2.2: icmp
4 0.525390 IP (tos 0x0, ttl 251, id 9643, offset 2960, flags [ ], proto: ICMP (1), length: 68) 10.2.2.2 > 10.1.1.1: icmp
5 0.546875 IP (tos 0x0, ttl 251, id 9643, offset 0, flags [+], proto: ICMP (1), length: 764) 10.2.2.2 > 10.1.1.1: ICMP echo reply, id 768, seq 2560, length 744
6 0.570312 IP (tos 0x0, ttl 251, id 9643, offset 744, flags [+], proto: ICMP (1), length: 756) 10.2.2.2 > 10.1.1.1: icmp
7 0.591796 IP (tos 0x0, ttl 251, id 9643, offset 1480, flags [+], proto: ICMP (1), length: 764) 10.2.2.2 > 10.1.1.1: icmp
8 0.615234 IP (tos 0x0, ttl 251, id 9643, offset 2224, flags [+], proto: ICMP (1), length: 756) 10.2.2.2 > 10.1.1.1: icmp

Em seguida as afirmações:

71 O trecho de captura corresponde ao tráfego gerado quando o host 10.1.1.1 executa o comando ping com um payload de 3.000 bytes de dados.
72 O pacote 4 é um fragmento inicial.
73 Os fragmentos de resposta possivelmente passaram por percursos diferentes.
74 A recepção fora de ordem dos fragmentos de resposta acarretará retransmissão.
75 No host 10.2.2.2, o MTU, conforme percebido pelo IP, deve ser, no máximo, de 1.507 bytes.

COMENTÁRIOS

71 – Certo

Analisando as linhas: 1,2 e 3, trata-se de um Echo Request, ou seja, um ping do host 10.1.1.1 para o host 10.2.2.2. Até aqui sem mistério

Na linha 1: Dados a serem enviados: 1492 (1472 de dados + 20 cabeçalho) offset = 0

Na linha 2: Recebido 1472 de dados (offset = 1472) e será enviado mais 1472: 1492 (lenght) – 20 bytes do cabeçalho IP.

Na linha 3: Recebido 2944 de dados (offset = 2944) falta enviar mais 64: 84 (lenght) – 20 = 2944 + 64 = 3008 bytes.

Como ele quer o payload e implica dizer "bytes transmitido, excluindo-se seus cabeçalhos", Porém, faltou excluirmos o cabeçalho do ICMP que tem 8 bytes (presente apenas no primeiro segmento). Logo, payload de dados é 3.000 bytes.

72 – Errado

Na prova a numeração do pacote 4 estava ausente, mas isso não anulou a questão, pois é possível saber qual é o quarto pacote, basta ver o *timestamp* que aparece no início dos pacotes. Mas vamos à análise do item. Para ser um fragmento inicial, o pacote deveria ter a flag *Don't Fragment* ligada, o que não ocorre. A ausência da flag indica que tal fragmento é o último e não o primeiro. Resumindo, o único fragmento que não possui a flag DF setada é o último fragmento. Para saber a posição do fragmento pacote completo devemos analisar o offset dele, mas isto não vem ao caso.

73 – Certo

Observe o campo do "offset". Os três primeiros fragmentos é o icmp request (ping) e tem um MTU de 1492 bytes. Já os 4 últimos fragmentos (icmp reply) chegam foram de ordem justamente por percorrerem caminhos distintos do icmp request. Caminhos no plural pois a resposta segue por 2 MTU diferentes: 1 com

tamanho máximo de 764 e outro com 756. Note que nessa misturada de caminhos o fragmento 4 foi o primeiro a chegar, mas ele é um fragmento final.

74 – Errado
Não existe retransmissão de fragmentos IP. Se um único fragmento se perder, toda a informação deverá ser retransmitida.

75 – Certo
Prezados, eu não vejo como este item pode estar correto. No meu entendimento o MTU máximo deveria ser 764. Acredito que a banca cometeu um erro no gabarito.

Observe as afirmativas abaixo, relacionadas a datagramas IPv4.

I - Em situações de congestionamento, os datagramas com o bit DF igual a 1 devem ser descartados.
II - A remontagem de datagramas sempre ocorre no destino.
III - Quando a MTU de uma rede é menor que o campo offset do datagrama a ser trafegado, ocorre a fragmentação.
Está(ão) correta(s) a(s) afirmativa(s)
(A) I, somente.
(B) II, somente.
(C) I e II, somente.
(D) II e III, somente.
(E) I, II e III.

COMENTÁRIOS

I - O bit DF apenas informa que o datagrama não pode ser fragmentado. Nesse caso, o datagrama só é descartado caso haja necessidade de passagem por uma rede cujo MTU seja menor que o pacote. Como, por imposição do bit DF (dont fragment), ele não poderá ser fragmentado, não há outra opção senão o descarte do mesmo. ERRADO

II - Existem 2 tipos de fragmentação: Transparente e não-transparente. No primeiro caso, os pacotes são fragmentados ao entrarem numa rede de MTU menor e remontados ao saírem dela. Nem a origem nem o destino final têm consciência da fragmentação. No **segundo caso, a remontagem só ocorre no destino**, ficando este ciente da fragmentação**. O protocolo IP opera com o segundo caso.** CERTO

III - Se formos preciosos na avaliação dessa assertiva, chegaremos à conclusão que não existe o campo offset, mas sim *fragment offset*. De qualquer forma, este campo apenas indica a posição do fragmento dentro do contexto do pacote original. A fragmentação ocorre quando a MTU é menor do que o pacote e não do campo offset. ERRADO

Comentários: [Gustavo/p3r1t0f3d3r4l]

“Quando se tem uma meta, o que era um obstáculo passa a ser uma das etapas do plano.”
Gerhard Erich Boehme