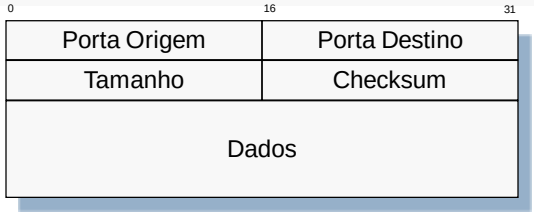
Protocolos de Transporte da Pilha TCP/IP

UDP (User Datagram Protocol)

- O protocolo UDP é bastante simples
  - Orientado a datagrama
  - Não orientado à conexão
  - Não executa controle de fluxo, controle de erro e sequenciamento
  - Não tem reconhecimento dos datagramas (ACK/NACK)
- Devido a sua simplicidade é considerado não confiável

Header UDP



Onde,

Porta Origem e Porta Destino identificam o processo de aplicação que está enviando dados e o processo de aplicação que irá receber os dados.

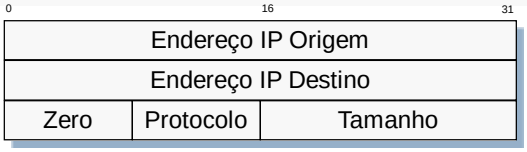
Tamanho é representa o tamanho total do frame UDP

Checksum é calculado usando o header UDP e também a área de dados, e destina-se a verificação de erros de transmissão.

Checksum UDP

- O Checksum no UDP é opcional
  - ⇨Campo de checksum = 0, não executa verificação
  - ⇨Campo de checksum <> 0, executa verificação
- O cálculo do checksum utiliza o header, os dados e também o Pseudo-Header
  - ⇨Este pseudo-header é utilizado para verificação adicional e confirmação de que o datagrama chegou ao destino correto

Pseudo-Header



Onde,

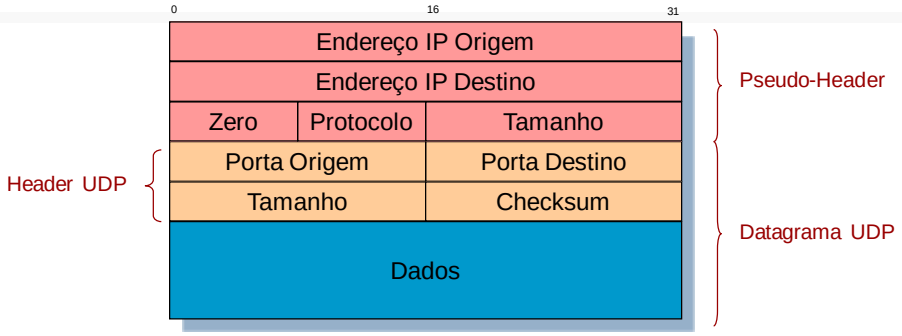
Endereço IP Origem e Endereço IP destino são do nível de rede (protocolo IP) utilizadas para a segunda validação do destino do datagrama.

Zero é um campo com valor zero para complementar a estrutura do pseudo-header.

Protocolo indica qual o protocolo de transporte (TCP ou UDP), pois o pseudo-header é utilizado para os dois protocolos.

Tamanho indica o tamanho do frame de transporte (UDP ou TCP)

Ordem de Header para o Checksum do UDP



Atenção!

O Pseudo-Header não é transmitido junto com o datagrama UDP, ele é utilizado apenas para cálculo do Checksum.

Processamento do Checksum

- Na origem, as informações necessárias são organizadas em blocos de 16 bits para o cálculo do checksum
  - ⇨Caso o cálculo resulte em zero, os 16 bits do checksum serão configurado todos em 1 (valor = 65535)
- Se optar-se por não utilizar checksum, os 16 bits serão configurados todos em 0

Processamento do Checksum

- Se o checksum recebido tem todos os bits em zero, não é necessário calculá-lo (pois não está sendo utilizado)
- Caso contrário, o cálculo do checksum é realizado novamente
  - ⇨Se o cálculo resultar em Zero, o datagrama não contém erros
  - ⇨Se o cálculo resultar diferente de Zero, o datagrama é descartado

Tamanho Máximo do Datagrama

- Teoricamente o tamanho máximo é de 64Kb
  - ⇨Porque no IP o campo tamanho total é de 16 bits
  - ⇨Mas deve-se considerar que no IP estão sendo calculado
    - ✓Tamanho do Header do IP (20 bytes)
    - ✓Datagrama UDP (8 bytes)
  - ⇨Assim, o tamanho máximo é de 65507 bytes

Tamanho Máximo do Datagrama

- Outros fatores podem influenciar
  - ⇨Programas de aplicação podem ser limitados pela interface de programação
  - ⇨Implementação do kernel do TCP/IP
- Truncando Datagramas
  - ⇨Apesar do tamanho máximo, nem todas as aplicações podem estar preparadas para receber um datagrama maior que esperado
    - ✓Truncar ou não? Depende da implementação de cada interface de programação

UDP e ICMP Source Quench

- Mensagens ICMP Source Quench
  - ⇨ Podem ser geradas pelo sistema quando ele recebe dados a uma taxa maior que ele consegue processar
  - ⇨ Não é obrigatória a geração, mesmo que o sistema descarte os datagramas
- "O sentimento corrente é que esta mensagem deve ser considerada obsoleta"
  - ⇨ Porque consome largura de banda e é ineficaz para o controle de congestionamento
    - ✓ Almquist 1993 (RFC???)

UDP e ICMP Source Quench

- Várias sistemas operacionais não geram estas mensagens
- Vários sistemas operacionais não repassam tais mensagens para o protocolo UDP
- Somente o TCP é notificado quando estas mensagens ocorrem!!!

TCP (Transmission Control Protocol)

- Protocolo de transporte considerado confiável
  - ⇨Orientado à conexão
  - ⇨Controle de erros com retransmissão
  - ⇨Controle de fluxo
  - ⇨Sequenciamento
  - ⇨Entrega ordenada
- Orientado a *"byte stream"*

Header TCP

Porta origem			Porta destino
Número de Seqüência			
Acknowlegement			
Tam.	Reser.	Flags	Window
Checksum			Urgent Pointer
Opções (se houver)			
Dados			

Header TCP

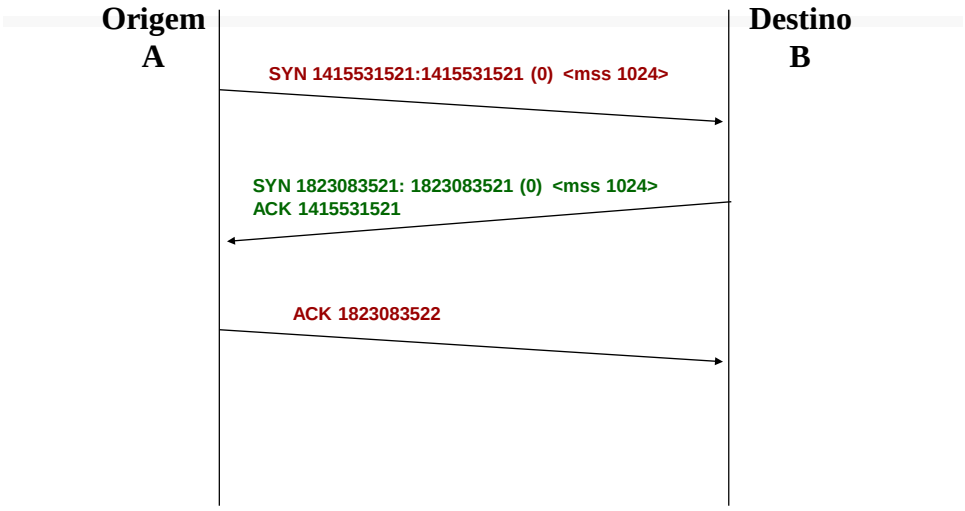
Onde,

Porta Origem e Porta Destino identificam o processo de aplicação que está enviando dados e o processo de aplicação que irá receber os dados.
Número de seqüência identifica os bytes enviados. Na prática ele é a identificação do primeiro byte de dados contido no segmento enviado. Os demais são seqüenciados a partir deste byte.
Acknowledgement identifica os bytes que foram recebidos e tratados sem erro pelo destino, bem como a seqüência do próximo byte esperado
Tamanho é representa o tamanho total do frame TCP
Reservado é um campo ainda não utilizado
FLAGS identifica as flags (syn, fin, psh, rst, ack, urg)
Window identifica o tamanho da janela para o controle de fluxo
Checksum destina-se a verificação de erros de transmissão. É calculado usando o pseudo header, o header TCP e também a área de dados
Urgent Poninter é um ponteiro para dados urgentes, contidos na área de dados.

Controle de Conexão TCP

- Três Fases
  - ⇨Estabelecimento da Conexão
  - ⇨Transmissão de Dados
  - ⇨Encerramento da Conexão
- Flags
  - ⇨SYN – solicitação de conexão
  - ⇨FIN – Finalização da Conexão
  - ⇨RST – Reset da Conexão
  - ⇨ACK – Reconhecimento de recebimento

Estabelecimento da Conexão



Estabeleci mento da Conexão

- Ativo x passivo
  - A origem da solicitação de conexão executa o "*active open*"
  - O destino que recebe a solicitação de conexão executa o "*passive open*"
- Origem e destino enviam seus número de seqüência iniciais para a conexão em curso
  - Este número deve ser alterado ao longo do tempo e ser diferente de conexão para conexão

Inicialização do Número de Seqüência

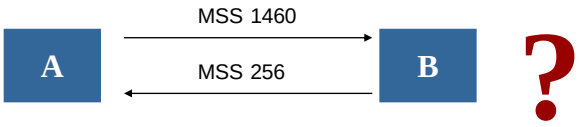
- RFC 793
  - ⇨Número de 32 bits
  - ⇨É incrementado a cada 4 microsegundos
- Como escolher o número inicial?
  - ⇨4.4BSD
    - ✓Quando sistema é inicializado o número de seqüência é 1 (violação da RFC)
    - ✓A variável é incrementada de 64.000 a cada ½ segundo
    - ✓Isso significa que irá retornar a 0 em períodos de 9 horas e ½

MSS (Maximum Segment Size)

- O MSS representa o tamanho do maior bloco de dados que poderá ser enviado para o destino.
- Não é negociável, cada host divulga o seu MSS
  - ⇨Default: 536 bytes (20 bytes IP, 20 bytes TCP, para um total de 576 bytes)
  - ⇨Ethernet: 1460 bytes (20 bytes IP, 20 bytes TCP, para um total de 1500 bytes)

MSS...

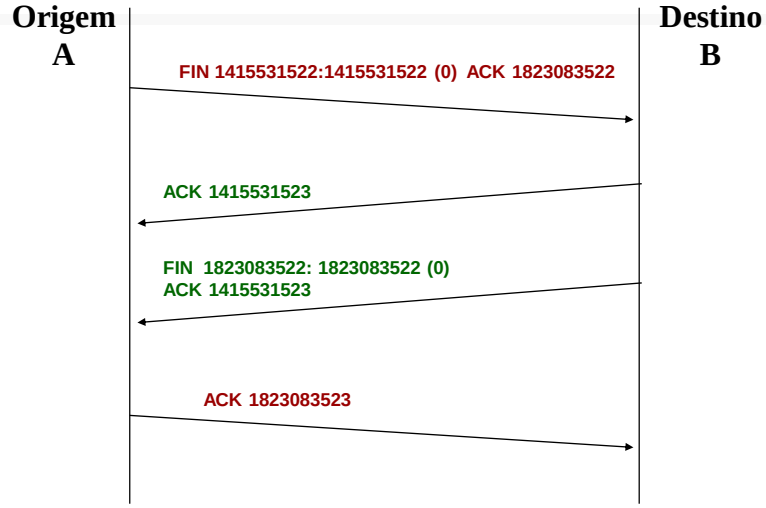
- Em geral, quanto maior o MSS melhor, até que ocorra fragmentação
  - ⇨Quanto maior a quantidade de dados enviados em um único bloco, menor o overhead de headers do TCP e do IP
- Exemplo



Outras Opções TCP

- End of option list (1 byte)
- No operation (NOP) (1 byte)
- Windows scale factor (3 bytes)
- Timestamp (10 bytes)
- MSS (4 bytes)

Encerramento da Conexão



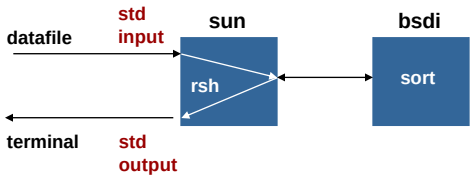
Encerramento da Conexão

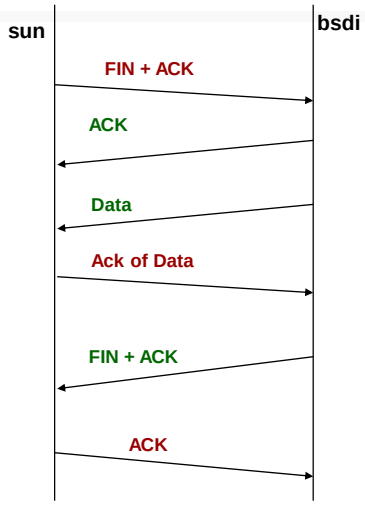
- *Half Close*
  - ⇨Conexões TCP são *full-duplex*, logo cada lado da conexão deve finalizar a conexão de forma independente
  - ⇨Quando um dos lados envolvidos recebe uma solicitação de finalização deve enviar a notificação para a aplicação
    - ✓Uma aplicação após receber o pedido de finalização ainda pode mandar dados

Half Close - Exemplo

Exemplo:

sun% **rsh bsdi < datafile**



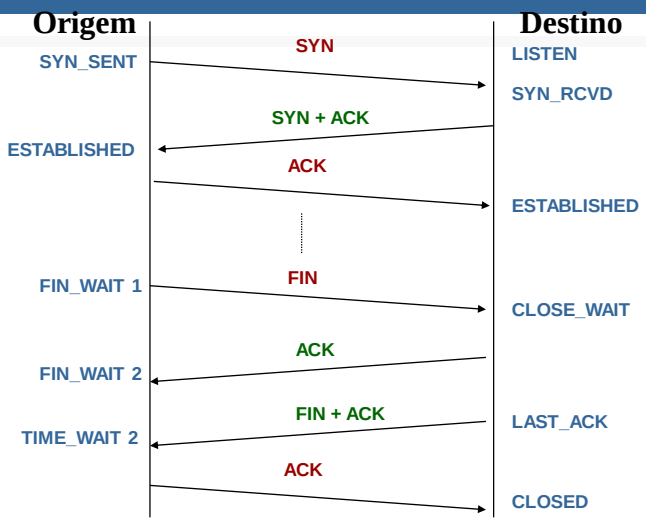


Timeout no Estabelecimento da Conexão

- Trecho de tráfego monitorado (tcpdump)

- Importante: tempo entre cada tentativa vs. tempo máximo exigido na RFC
  - ⇨Tempo: 75 segundos
  - ⇨4.4 BSD: leva 76 segundos
  - ⇨Problema: Timeout

Estados x Mensagens



2MSL Wait State

- O TIME_WAIT é também chamado 2MSL
  - ⇨MSL = Maximum Segment Life
  - ⇨Tempo máximo que um segmento pode existir antes de ser descartado
  - ⇨TTL do IP ???
- RFC 793
  - ⇨“MSL dever ser de 2 minutos”
  - ⇨Mas as implementações variam de entre 30 segundos, 1 minuto e 2 minutos

Utilização do 2MSL

- Quando é executado um *active close* e enviado o ACK final, a conexão deve permanecer no estado TIME_WAIT pelo dobro do tempo especificado no MSL
  - ⇨ Isso irá permitir a retransmissão do ACK final, caso o primeiro seja perdido
  - ⇨ Outra conseqüência é a não liberação do par de sockets utilizados para a conexão
  - ⇨ Algumas implementações restringem também o uso das portas locais durante o 2MSL

Utilização do 2MSL

- Socket option
  - ⇨ SO_REUSEADDR
  - ⇨ Transpor as regras sobre uso do endereço das portas
- Quanto a transmissão de dados
  - ⇨ Qualquer segmento que chegue para uma conexão neste período (2MSL) deverá ser descartado
    - ✓ Conseqüências para clientes e servidores?

Quite Time

- Considere a seguinte situação
  - Ocorre uma falha no host que está no estado 2MSL
  - Ele faz o reboot ainda no período de espera do MSL
  - Ele estabelece uma nova conexão imediatamente, usando o mesmo endereço de porta local e porta remota
- O que poderá ocorrer neste caso?
- Para prevenir, o host deve esperar por MSL segundos, após o reboot, antes de estabelecer qualquer conexão nova.

Reset de Conexões

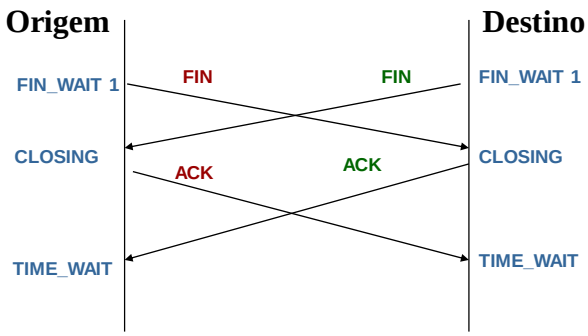
- Em geral, um Reset é gerado sempre que é recebido um segmento que não parece estar correto para a conexão identificada.
- Casos
  - Solicitações de conexões para portas inexistentes
  - Aborto de conexões
  - Solicitações de conexões falsas

Estabelecimento de Conexões Simultâneas

- É possível que 2 hosts tentem estabelecer conexão entre eles simultaneamente
  - ⇨Ambos executam um *active open*
  - ⇨Exemplo:
    - ✓Host A solicita conexão ao Host B na porta 777 e usa como porta local 888
    - ✓Host B solicita conexão ao Host A na porta 888 e usa como porta local 777
- TCP foi projetado para suportar estes casos
  - ⇨Apenas uma conexão resulta, não duas

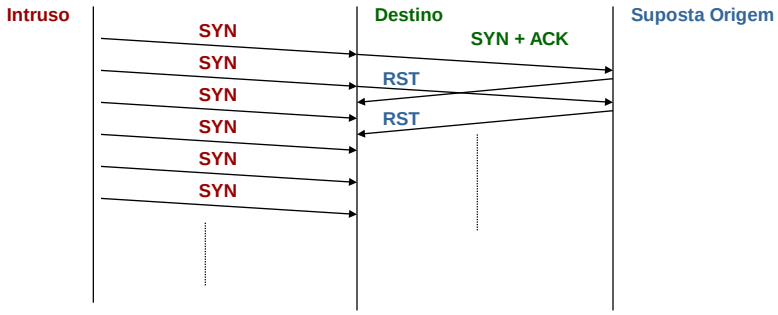
Encerramento de Conexões simultâneas

- Os hosts também podem tomar a iniciativa de encerrar a conexão simultaneamente



SYN FLOOD

- Flooding de solicitações de conexão falsas
  - ⇨Normalmente usa IP Spoofing

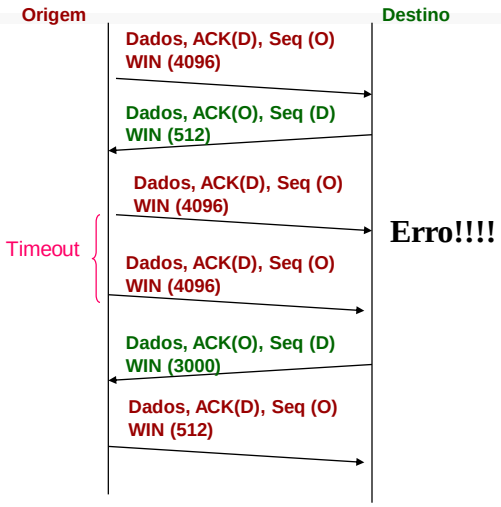


Controle de Erros

- O TCP executa controle de erro com retransmissão
  - ⇨Neste caso o checksum não é opcional
  - ⇨Se um segmento TCP é recebido com checksum igual a zero, ele é descartado
  - ⇨O destino envia mensagens de reconhecimento positivo
    - ✓Não envia NACK
    - ✓A necessidade de realizar uma retransmissão é detectada pela ausência do ACK

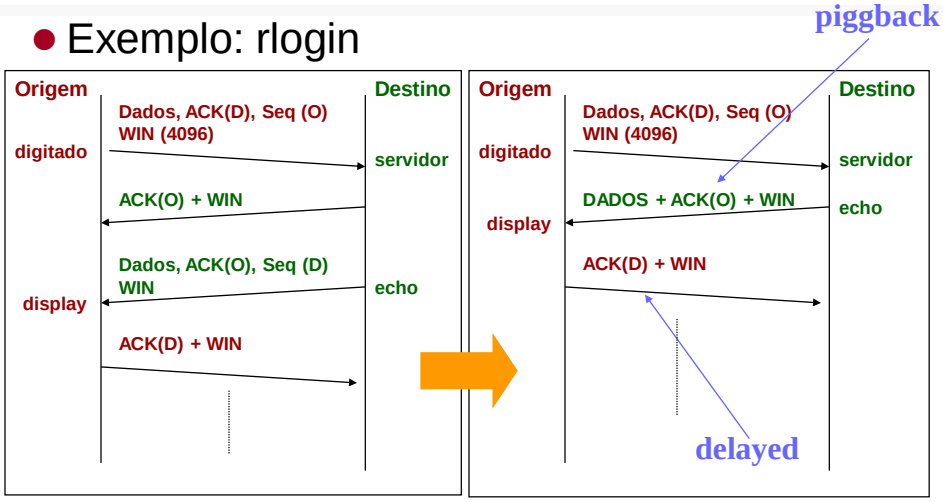
Controle de Fluxo

- O TCP executa o algoritmo de janela deslizante
  - ⇨ A cada envio de mensagens o host informa o número de bytes que podem ser recebidos



Fluxo Interativo

- Exemplo: rlogin



Algoritmo de Nagle

- Exemplo do Rlogin
  - 1 byte de dados
  - 20 bytes TCP + 20 Bytes IP
- Problema?
  - Overhead de controle
- Nagle:
  - Pequenos segmentos só podem ser enviados após o recebimento do ACK dos dados enviados anteriormente

Algoritmo de Nagle

- Assim, o TCP coleta pequenos segmentos de dados e os envia juntos
- A velocidade de envio irá variar com de acordo com a velocidade de recebimento do ACK
  - Algoritmo é *self-clocking*
- Exemplo: Ethernet
  - RTT aproximadamente de 16 ms
  - Para ser mais rápido deveria digitar mais de 60 caracteres por segundo
  - Isto significa que raramente é aplicado nestas redes
  - Utilizado em redes com RTT maior (p.ex: WAN)

Desabilitando Nagle

- Quando é necessário?
  - Trafego Interativo
  - Xwindow (movimentos do mouse....)
- A API do socket pode utilizar a opção TCP_NODELAY para desabilitar o algoritmo