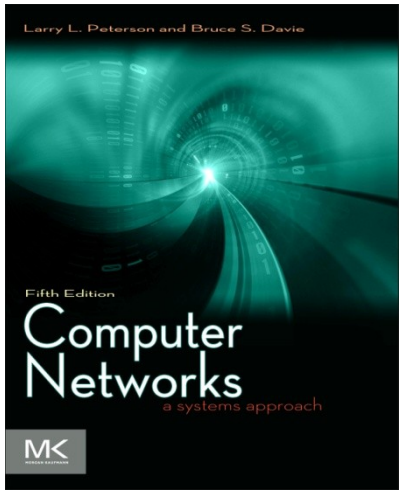


Capítulo 5

Protocolos Fim-a-Fim

Problema

- Como transformar o serviço de entrega de pacotes host-a-host em um canal de comunicação processo-a-processo?

Visão Geral do Capítulo

- Demultiplexador Simples (UDP)
- Cadeia de Bytes Confiável (*Reliable Byte Stream*) (TCP)

Objetivos do Capítulo

- Entender o serviço de demultiplexação
- Discutir um protocolo de cadeia de bytes

Protocolos Fim-a-Fim

- Propriedades comuns que podem ser esperadas de um protocolo de transporte
  - Garantir entrega de mensagens
  - Entregar mensagens na mesma ordem que foram enviadas
  - Entregar no máximo uma cópia de cada mensagem
  - Suportar mensagens arbitrariamente grandes
  - Suportar sincronização entre transmissor e receptor
  - Permitir que o receptor aplique controle de fluxo ao transmissor
  - Suportar múltiplos processos de aplicação em cada host

Protocolos Fim-a-Fim

- Limitações típicas de redes em que um protocolo de transporte tem de funcionar
  - Descarte de mensagens
  - Reordenamento de mensagens
  - Entrega de cópias duplicadas de uma dada mensagem
  - Limitação de tamanho de mensages
  - Entrega de mensagens com atrasos arbitrariamente longos

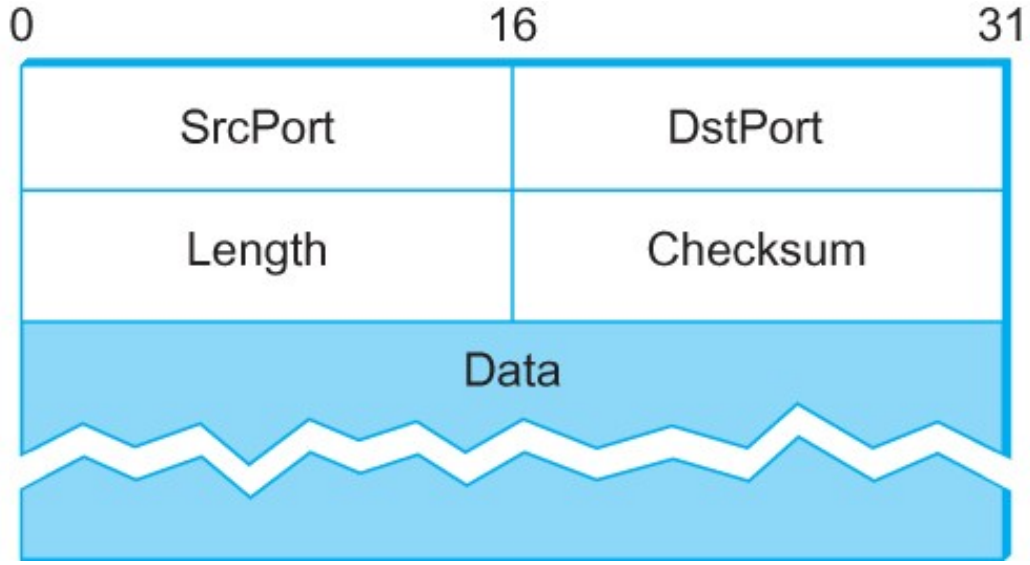
Protocolos Fim-a-Fim

- Desafios para Protocolos de Transporte
  - Desenvolver algoritmos que transformam as propriedades indesejáveis da rede subjacente em serviços de alto nível exigidos pelos programas de aplicação

Demultiplexador Simples (UDP)

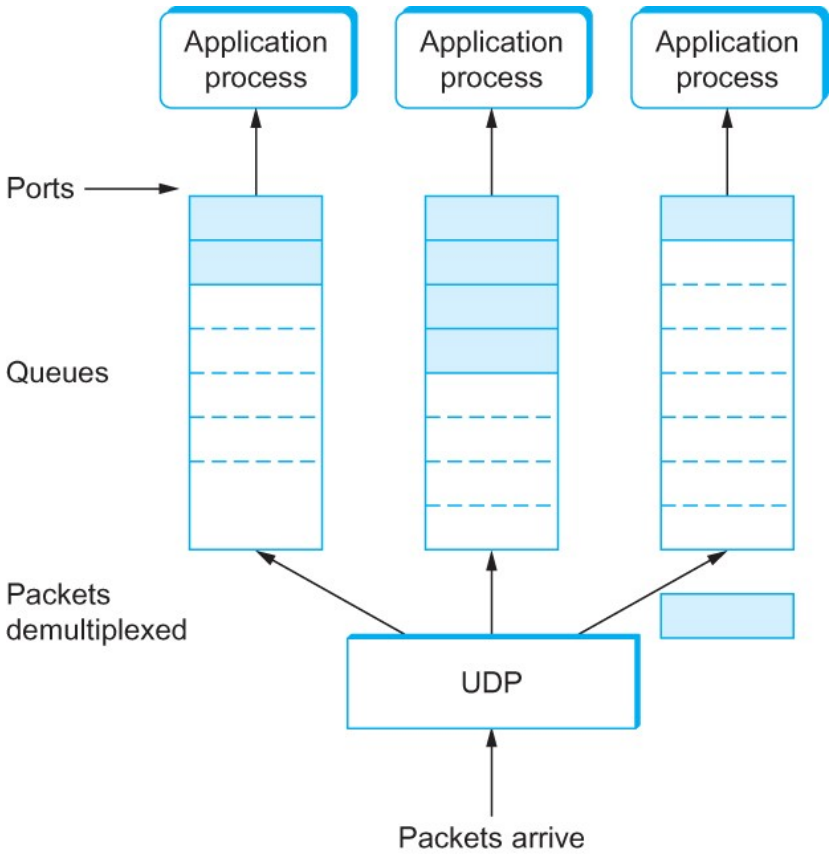
- Estende o serviço de entrega host-a-host em um serviço de comunicação processo-a-processo
- Adiciona um nível de demultiplexação que permite que múltiplos processos em cada host compartilhem a rede

Demultiplexador Simples (UDP)



Formato do cabeçalho UDP

Demultiplexador Simples (UDP)



Filas de Mensagens UDP

Cadeia de Bytes Confiável (TCP)

- Diferente de UDP, TCP (Transmission Control Protocol) oferece os seguintes serviços:
  - Confiável
  - Orientado à conexão
  - Cadeia de bytes

Controle de Fluxo VS Controle de Congestionamento

- Controle de fluxo previne que transmissores sobrecarreguem a capacidade dos receptores
- Controle de congestionamento previne que muitos dados sejam injetados na rede, fazendo com que switches e enlaces se tornem sobrecarregados (congestionados)

Questões Fim-a-Fim

- No coração de TCP está o algoritmo de janela deslizante (discutido no Capítulo 2)
- Como TCP funciona na Internet ao invés de um enlace ponto-a-ponto, os seguintes problemas precisam ser resolvidos pelo algoritmo de janela deslizante
  - TCP suporta conexões lógicas entre processos que estão rodando em dois computadores diferentes na Internet
  - Conexões TCP possuem diferentes tempos de RTT
  - Pacotes podem ser reordenados na Internet

Questões Fim-a-Fim

- TCP precisa de um mecanismo para que cada lado da conexão descubra o quanto de recursos o outro lado pode utilizar na conexão
- TCP precisa de um mecanismo para que o transmissor descubra a capacidade da rede

Segmento TCP

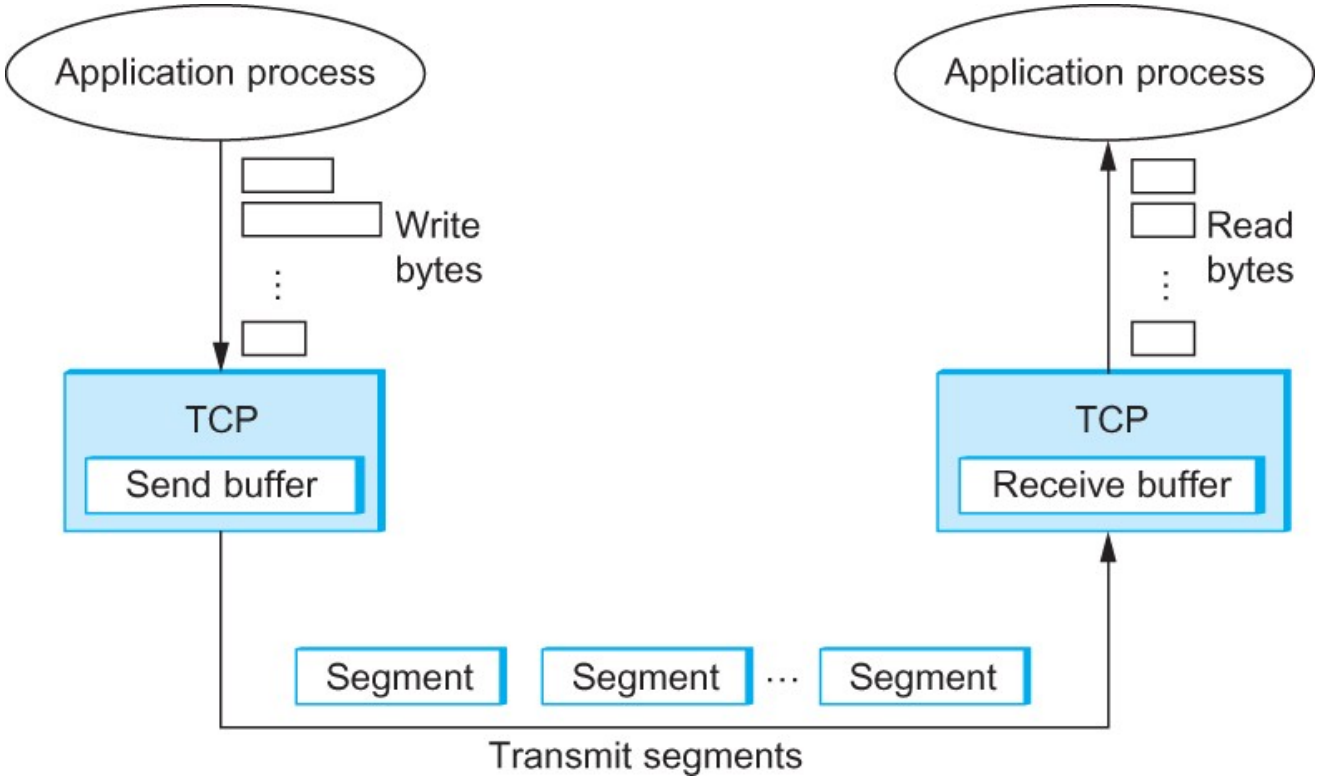
- TCP é um protocolo orientado a bytes, o que significa que o transmissor escreve bytes em uma conexão TCP e o receptor lê os bytes no outro lado da conexão TCP.

- Embora “byte stream” descreva o serviço que TCP oferece a processos de aplicação, TCP por si só não transmite os bytes individualmente na Internet.

Segmento TCP

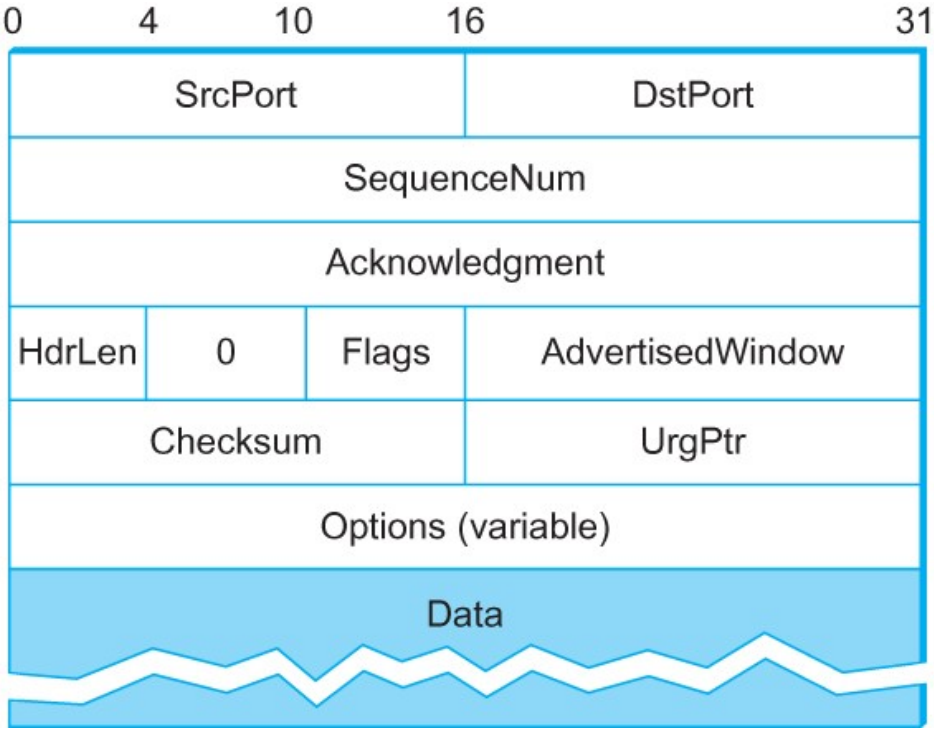
- TCP bufferiza bytes do processo transmissor para preencher um pacote de tamanho razoável e então envia o pacote para o seu par no host destino.
- TCP no host destino copia o conteúdo do pacote em um buffer e o processo receptor lê desse buffer quando puder.
- Os pacotes trocados entre pares TCP são chamados *segmentos*.

Segmento TCP



Como TCP gerencia uma cadeia de bytes.

Formato do Cabeçalho TCP



Formato do Cabeçalho TCP

Cabeçalho TCP

- Os campos SrcPort e DstPort identificam as portas de origem e destino, respectivamente.
- Os campos Acknowledgment, SequenceNum, e AdvertisedWindow são utilizados pelo algoritmo de janela deslizante de TCP.
- Como TCP é um protocolo orientado a bytes, cada byte de dados possui um número de sequência; o campo SequenceNum contém o número de sequência do primeiro byte carregado no segmento.
- Os campos Acknowledgment e AdvertisedWindow carregam informações sobre o fluxo de dados de direção oposta.

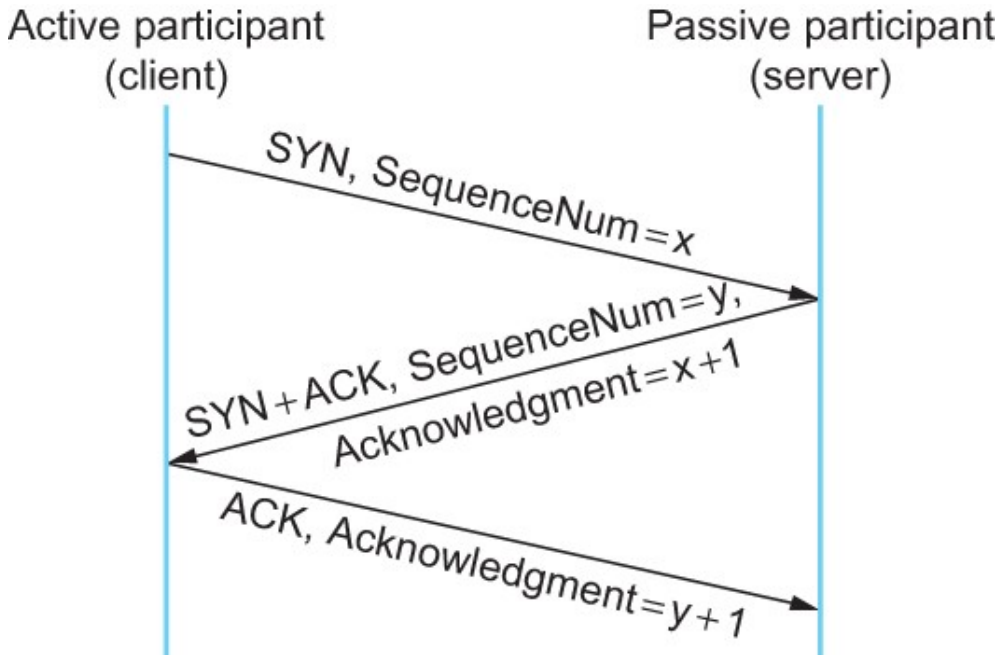
Cabeçalho TCP

- O campo de 6 bits Flags é usado para transmitir informações de controle entre os pares TCP.
- As possíveis flags incluem SYN, FIN, RESET, PUSH, URG, e ACK.
- As flags SYN e FIN são usados para estabelecer e terminar uma conexão TCP, respectivamente.
- A flag ACK é 1 sempre que o campo Acknowledgment é válido, implicando que o receptor deve processá-lo.

Cabeçalho TCP

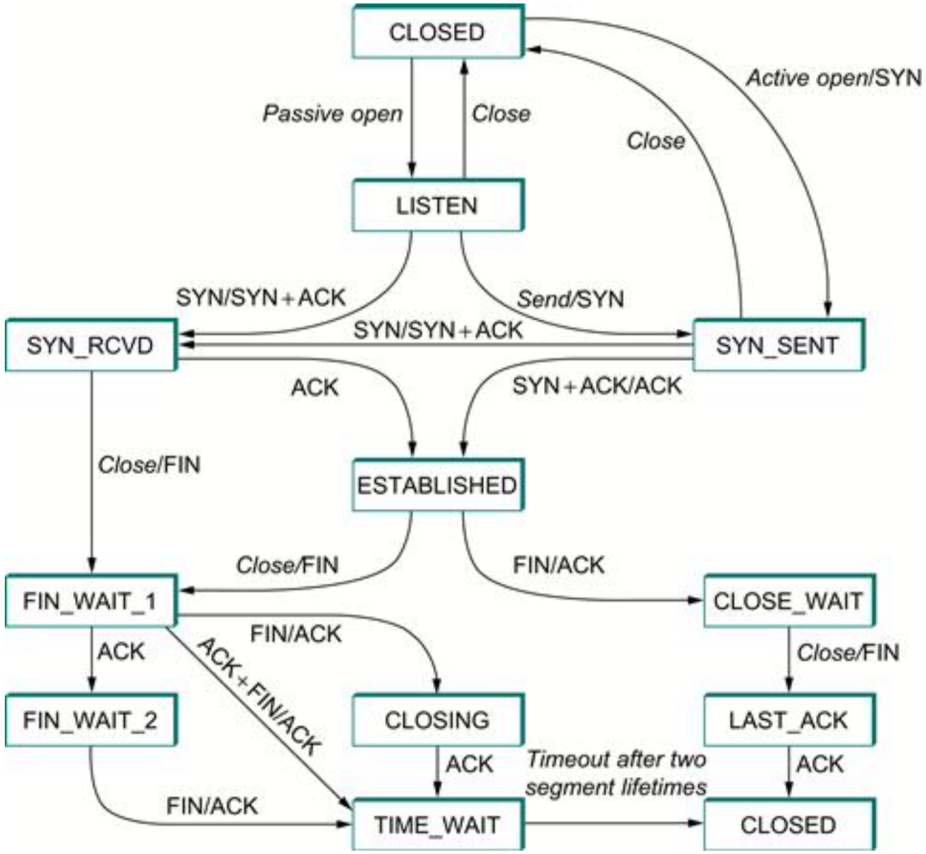
- A flag URG indica que o segmento contém dados urgentes. Quando esta flag é 1, o campo UrgPtr indica onde os dados não urgentes começam no segmento.
- Os dados urgentes estão no início do corpo do segmento e ocupam UrgPtr bytes do segmento.
- A flag PUSH indica que a camada TCP do receptor deve notificar o processo receptor que o transmissor invocou a operação push.
- A flag RESET indica que o receptor ficou confuso e quer abortar a conexão.
- O campo Checksum é calculado sobre o cabeçalho e dados TCP e o pseudocabeçalho, que é constituído dos campos do cabeçalho IP comprimento e endereços de origem e destino.

Estabelecimento/Encerramento de Conexão TCP



Linha do tempo para o algoritmo de three-way handshake

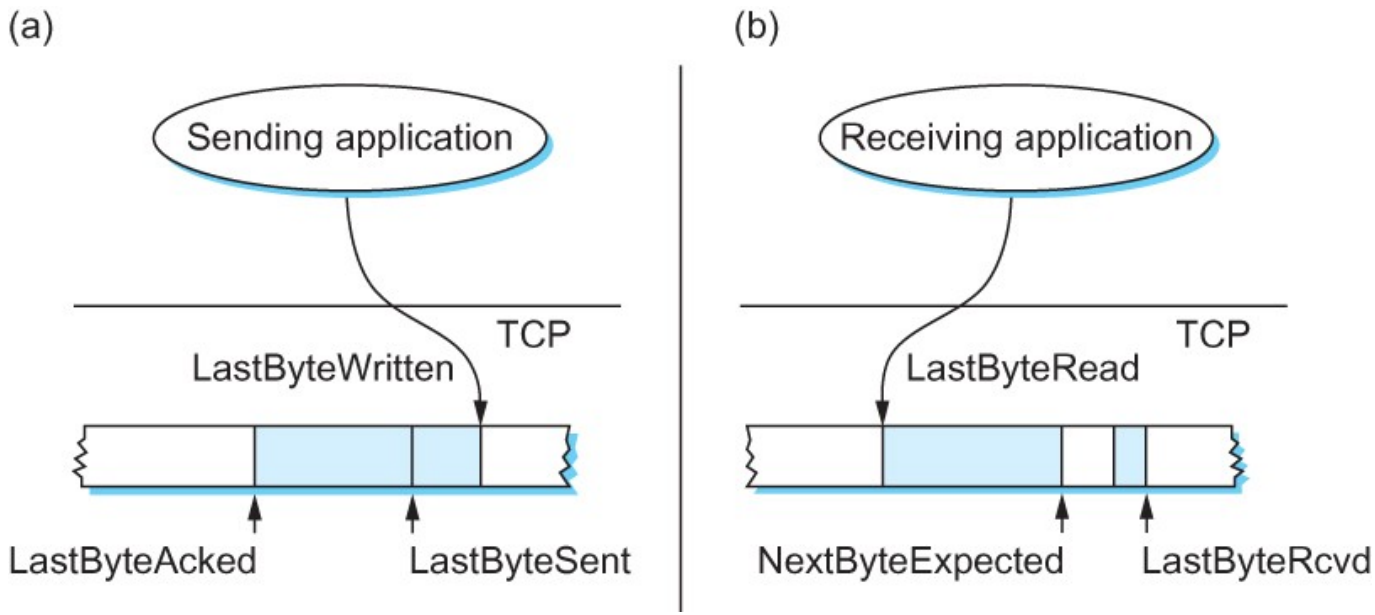
Diagrama de Estados TCP



Janela Deslizante TCP

- TCP usa uma variante do algoritmo de janela deslizante que serve para três propósitos:
  - (1) garantir entrega confiável de dados,
  - (2) garantir que dados são entregues em ordem, e
  - (3) implementar controle de fluxo entre transmissor e receptor.

Janela Deslizante TCP



Relação entre buffer do transmissor (a) e buffer do receptor (b).

Janela Deslizante TCP

- Lado do Transmissor
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
- Lado do Receptor
  - LastByteRead < NextByteExpected
  - NextByteExpected ≤ LastByteRcvd + 1

Controle de Fluxo TCP

- LastByteRcvd − LastByteRead ≤ MaxRcvBuffer
- AdvertisedWindow = MaxRcvBuffer − ((NextByteExpected − 1) − LastByteRead)
- LastByteSent − LastByteAcked ≤ AdvertisedWindow
- EffectiveWindow = AdvertisedWindow − (LastByteSent − LastByteAcked)
- LastByteWritten − LastByteAcked ≤ MaxSendBuffer
- Se o processo transmissor tentar escrever y bytes para TCP, mas

  (LastByteWritten − LastByteAcked) + y > MaxSendBuffer

  então TCP bloqueia o processo transmissor e não permite que ele gere novos dados.

Proteção Contra Reinício Cíclico

- SequenceNum: 32 bits
- AdvertisedWindow: 16 bits
  - TCP atende ao requisito do algoritmo de janela deslizante que o espaço de números de sequência seja o dobro do tamanho da janela $2^{32} >> 2 \times 2^{16}$

Proteção Contra Reinício Cíclico

- Importância do espaço de números de sequência de 32 bits
  - O número de sequência usado em dada conexão pode ser reiniciado (2^{32}-1 → 0).
  - Um byte com número de sequência x pode ser enviado em um dado momento e algum tempo depois um outro byte com o mesmo número de sequência x pode ser enviado.
  - Pacotes não podem sobreviver na Internet por mais do que **MSL** (*Maximum Segment Lifetime*).
  - **MSL** é igual a 120 segundos
  - Temos de nos certificar que o número de sequência não é reiniciado dentro de um período de tempo de 120 segundos.
  - Depende da velocidade que dados podem ser transmitidos.

Proteção Contra Reinício Cíclico

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

Tempo para que o número de sequência de 32 bits seja reiniciado.

Mantendo o “Tubo” Cheio

- O campo AdvertisedWindow de 16 bits deve ser grande o suficiente para permitir que o transmissor mantenha o “tubo” cheio.
- É claro que o receptor não precisa abriar uma janela do tamanho máximo permitido pelo campo AdvertisedWindow.
- Se o receptor possui memória suficiente para seu buffer de recepção
  - A janela precisa ter tamanho suficiente para permitir uma quantidade de dados igual ao produto atraso × largura de banda
  - Assuminto um RTT de 100 ms, temos

Mantendo o “Tubo” Cheio

Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

Tamanho de janela necessário para um RTT de 100 ms.

Iniciando a Transmissão

- Como o TCP decide transmitir um segmento?
  - TCP provê uma abstração de cadeia/fluxo de bytes.
  - Os programas de aplicação escrevem bytes nos fluxos (*streams*).
  - Cabe a TCP decidir se há bytes suficientes para enviar um segmento.

Iniciando a Transmissão

- Quais fatores influenciam essa decisão
  - Ignore flow control: window is wide open, as would be the case when the connection starts
  - TCP possui três mecanismos para disparar a transmissão de um segmento
    - 1) TCP mantém uma variável MSS e envia um segmento tão logo ele possua MSS bytes do processo transmissor
      - MSS é geralmente igual ao tamanho do maior segmento que TCP pode enviar sem causar fragmentação de um datagrama IP na rede local.
      - MSS: MTU da rede diretamente conectada – (TCP header + IP header)
    - 2) O processo transmissor pede explicitamente para TCP enviar os dados
      - TCP suporta a operação push
    - 3) Quando um temporizador expira
      - O segmento resultante contém a quantidade de bytes no buffer de transmissão

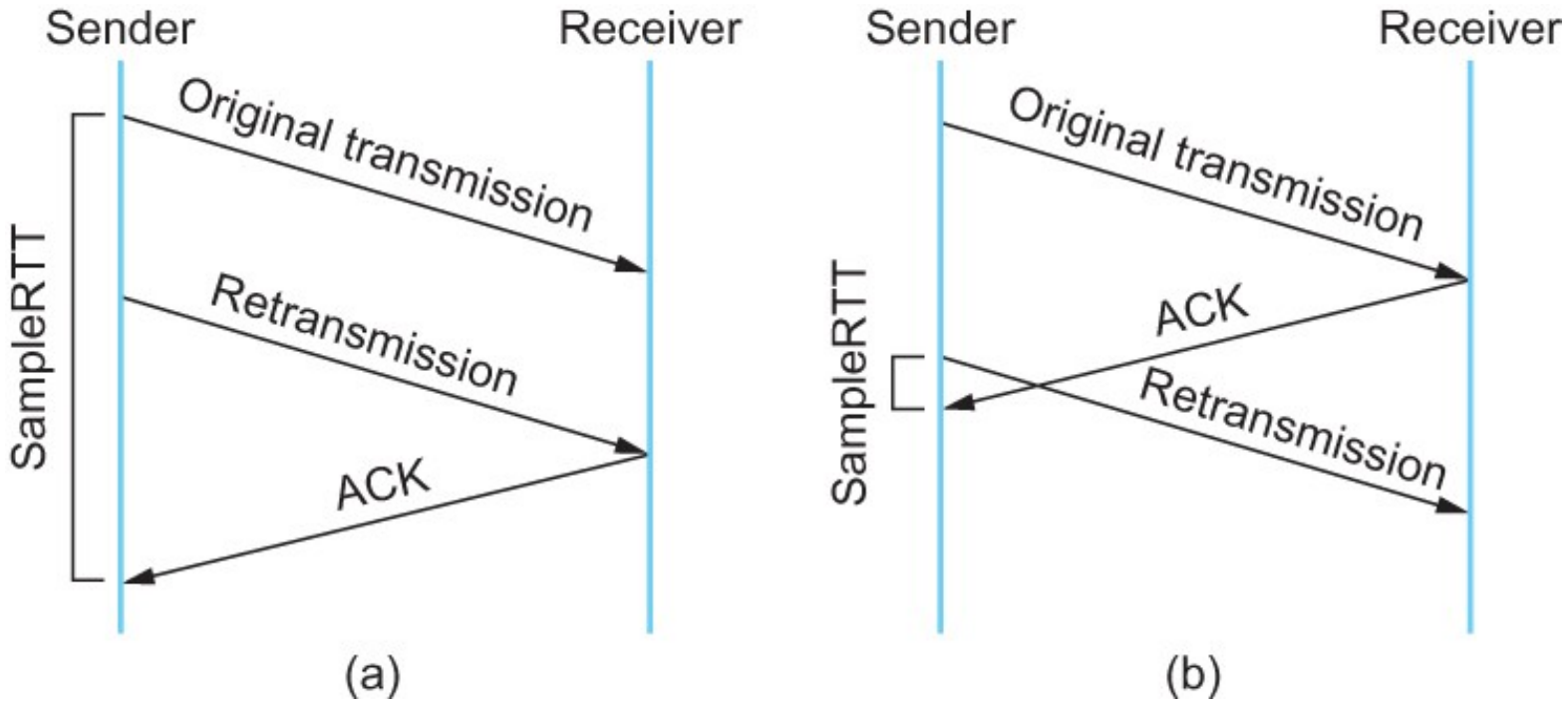
Retransmissão Adaptativa

- Algoritmo Original
  - Meça **`SampleRTT`** para cada par segmento/ACK
  - Calcule a média ponderada do RTT
    - **`EstRTT`** = α x **`EstRTT`** + (1 - α) x **`SampleRTT`**
    - α entre 0.8 e 0.9
  - Ajuste o timeout baseado em **`EstRTT`**
    - **`TimeOut = 2 x EstRTT`**

Algoritmo Original

- Problema
  - ACK não confirma uma transmissão em particular
    - Ele confirma o recebimento de dados
  - Quando um segmento é retransmitido e um ACK chega ao transmissor
    - É impossível decidir se este ACK deve estar associado a primeira ou segunda tranmissão para se calcular RTTs

Algoritmo de Karn/Partridge



Associação do ACK com (a) transmissão original versus (b) retransmissão

Algoritmo de Karn/Partridge

- Não usa amostras de RTT quando tiver retransmitido um segmento
- Dobra o timeout depois de cada retransmissão

Algoritmo de Karn/Partridge

- O algoritmo de Karn-Partridge melhorou a abordagem original, mas ele não elimina congestionamento

- Precisamos entender como timeout está relacionado a congestionamento
  - Se o timeout expira muito cedo, você pode retransmitir um segmento sem necessidade e aumentar a carga na rede

Algoritmo de Karn/Partridge

- O problema principal com o cálculo original é que ele não considera a variância das amostras.
- Se a variância entre as amostras de RTTs é pequena
  - Então pode-se confiar no valor estimado de RTT.
  - Não há necessidade de se multiplicá-lo por 2 para calcular o timeout.

Algoritmo de Karn/Partridge

- Por outro lado, uma variância grande nas amostras sugere que o valor de timeout não pode ser muito próximo do valor estimado de RTT.
- Jacobson/Karels propuseram um novo esquema de retransmissão paraTCP.

Algoritmo de Jacobson/Karels

- Difference = SampleRTT − EstimatedRTT
- EstimatedRTT = EstimatedRTT + (δ × Difference)
- Deviation = Deviation + δ × (|Difference| − Deviation)
- TimeOut = μ × EstimatedRTT + φ × Deviation

Baseado em experiência, geralmente μ é igual a 1, φ igual a 4, e δ é uma fração entre 0 e 1. Assim, quando a variância é pequena, TimeOut fica próximo de EstimatedRTT; uma variância grande faz com que o termo de desvio domine o cálculo.

Resumo

- Discutimos como converter um serviço de entrega de pacotes host-a-host em um canal de comunicação processo-a-processo.
- Discutimos UDP.
- Discutimos TCP.