Slides:

Camada de Aplicação: https://www.cin.ufpe.br/~suruagy/cursos/redes/cap2-Kurose.pdf
Camada de Transporte: https://www.cin.ufpe.br/~suruagy/cursos/redes/cap3-Kurose.pdf

Pontos destacados:

- Multiplexação e demultiplexação
- Localizar 3-way handshake no code
- Localizar na captura do wireshark (confirmação, perda de pacote, timeout, handshake)
- Identificar tcp e udp na API
- Diferenciar tcp udp
- Diferença entre controle de fluxo e de congestionamento
    - “A principal diferença entre o controle de fluxo e o controle de congestionamento é que, no controle de fluxo, os tráfegos são controlados, que são o fluxo do emissor para o receptor. Por outro lado, no controle de congestionamento, os tráfegos são controlados entrando na rede.”
- Casos de retransmissão do tcp?
- Hierarquia servidor DNS
- Registro de recurso ->

  MX → (enterprise.com, mail.enterprise.com, MX)
  A → (west4.enterprise.com, 142.81.17.206, A)
  NS → (www.enterprise.com, dns.enterprise.com, NS)
  CNAME → (www.enterprise.com, west4.enterprise.com, CNAME)

- Funcionamento do HTTP
- HTTP e DNS - pode ser wireshark ou imagem
- Dar uma lida em correio eletrônico
- Hierarquia do DNS
- Como funciona o autômato e o TCP
- Sockets
- Ver os ngc de número de sequência/ACK
- Ver a questão de DNS do kurose sobre o tempo
    - 2 RTT + tempo de transmissão do TCP…??
        - RTT do DNS + 2RTT do HTTP (1 TCP e 1 HTTP)
        - Parte do HTTP muda de acordo com o tipo de conexão tipo parallel/serial persistent/non-persistent

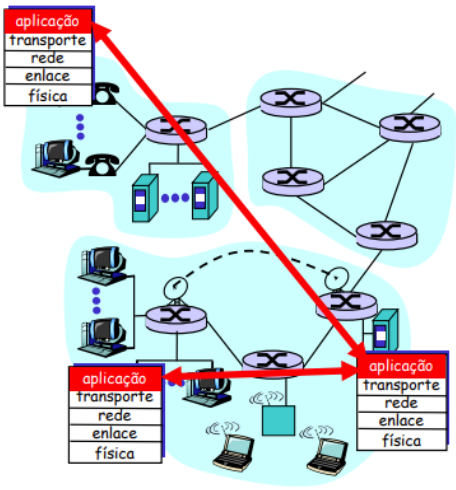
- Atividade do Wireshark pra sábado

O que NÃO vai cair:

- RDT
- 2-way handshake
- Seleção repetitiva
- Diferença entre multiplexação e encapsulamento
- Fórmulas lá complexas
- Cookies

Camada de Aplicação

<u>Criando uma aplicação de rede:</u>



Programas que

- Executam em (diferentes) sistemas finais
- Comunicam-se através da rede
    - p.ex., servidor Web se comunica com o navegador

Programas não relacionados ao núcleo da rede

- Dispositivos do núcleo da rede não executam aplicações dos usuários
- Aplicações nos sistemas finais permitem rápido desenvolvimento e disseminação

<u>Arquitetura cliente-servidor</u>

Servidor:

- Sempre ligado
- Endereço IP permanente
- Escalabilidade com data centers

Clientes:

- Comunicam-se com o servidor
- Podem estar conectados intermitentemente
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente com outros clientes

Exemplos: HTTP, IMAP, FTP

<u>Arquitetura P2P/Peer-peer:</u>

- Não há servidor sempre ligado
- Sistemas finais arbitrários se comunicam diretamente
- Pares solicitam serviços de outros pares e em troca provêem serviços para outros parceiros:
    - Autoescalabilidade – novos pares trazem nova capacidade de serviço assim como novas demandas por serviços.
- Pares estão conectados intermitentemente e mudam endereços IP
    - Gerenciamento complexo

<u>Comunicação entre Processos:</u>

Processo: programa que executa num sistema final

Processo cliente: processo que inicia a comunicação

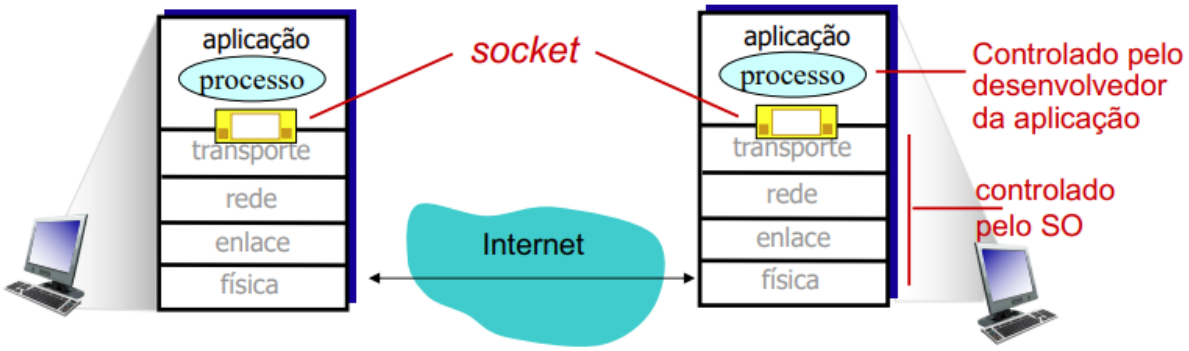
Processo servidor: processo que espera ser contatado

- Nota: aplicações com arquiteturas P2P possuem processos clientes e processos servidores

<u>Sockets:</u>

- Os processos enviam/ recebem mensagens para/dos seus sockets
- Um socket é análogo a uma porta
    - Processo transmissor envia a mensagem através da porta

- O processo transmissor assume a existência da infraestrutura de transporte no outro lado da porta que faz com que a mensagem chegue ao socket do processo receptor



Endereçamento de processos:

- Para que um processo receba mensagens, ele deve possuir um identificador
- Cada hospedeiro possui um endereço IP único de 32 bits
  - P: O endereço IP do hospedeiro no qual o processo está sendo executado é suficiente para identificar o processo?
  - R: Não, muitos processos podem estar executando no mesmo hospedeiro
- **O identificador inclui tanto o endereço IP quanto os números das portas associadas com o processo no hospedeiro .**
- Exemplo de números de portas:
  - Servidor HTTP: 80
  - Servidor de Email: 25
- Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu
  - Endereço IP: 128.119.245.12
  - Número da porta: 80

Os protocolos da camada de aplicação definem:

- Tipos de mensagens trocadas:
  - ex. mensagens de requisição e resposta
- Sintaxe das mensagens:
  - campos presentes nas mensagens e como são identificados
- Semântica das mensagens:
  - significado da informação nos campos
- Regras para quando os processos enviam e respondem às mensagens

Protocolos abertos:

- Definidos em RFCs, todos têm acesso a definição do protocolo
- Permitem a interoperação
- Ex., HTTP e SMTP

Protocolos proprietários:

- Ex., Skype, Zoom

De que serviços uma aplicação necessita?

- Integridade dos dados (sensibilidade a perdas)
  - algumas aplicações (p.ex., transf. de arquivos, transações web) requerem uma transferência 100% confiável
  - outras (p.ex. áudio) podem tolerar algumas perdas

- Temporização (sensibilidade a atrasos)
    - algumas aplicações (p.ex., telefonia Internet, jogos interativos) requerem baixo retardo para serem “viáveis”
- Vazão (throughput)
    - algumas aplicações (p.ex., multimídia) requerem quantia mínima de vazão para serem “viáveis”
- Segurança
    - criptografia, integridade dos dados, …

Serviços providos pelos protocolos de transporte da Internet:

Serviço TCP:

- transporte confiável entre processos remetente e receptor
- **controle de fluxo: remetente não vai “afogar” receptor**
- **controle de congestionamento: estrangular remetente quando a rede estiver carregada**
- não provê: garantias temporais ou de banda mínima
- orientado a conexão: apresentação requerida entre cliente e servidor

Serviço UDP:

- transferência de dados não confiável
- não provê: estabelecimento da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima

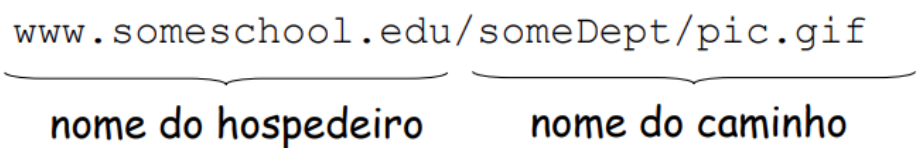
Tornando o TCP seguro:

- Sockets TCP & UDP simples
    - Sem criptografia
    - Senhas em texto aberto enviadas aos sockets atravessam a Internet em texto aberto
- SSL/Transport Layer Security (TLS)
    - Provê conexão TCP criptografada
    - Integridade dos dados
    - Autenticação do ponto terminal
        - SSL está na camada de aplicação
            - Aplicações usam bibliotecas SSL, que “falam” com o TCP
    - API do socket SSL
    - Senhas em texto aberto enviadas ao socket atravessam a rede criptografadas

A Web e o HTTP:

- Páginas Web consistem de objetos
- Páginas Web consistem de um arquivo base HTML que inclui vários objetos referenciados
- Cada objeto é endereçável por uma URL

Exemplo de URL:



Protocolo HTTP (hypertext transfer protocol):

- Protocolo da camada de aplicação da Web
- Modelo cliente/servidor:
    - Cliente: browser que pede, recebe e visualiza objetos Web
    - Servidor: servidor Web envia objetos em resposta a pedidos
- Usa serviço de transporte TCP:
    - Cliente inicia conexão TCP (cria socket) ao servidor, porta 80
    - Servidor aceita conexão TCP do cliente
    - Mensagens HTTP (mensagens do protocolo da camada de apl) trocadas entre browser (cliente HTTP) e servidor Web (servidor HTTP)
    - Encerra conexão TCP

- HTTP é “sem estado”
    - Servidor não mantém informação sobre pedidos anteriores do cliente

Dois tipos de conexões HTTP:

- **HTTP não persistente**
    - **No máximo um objeto é enviado numa conexão TCP**
    - A conexão é então encerrada
    - Baixar múltiplos objetos requer o uso de múltiplas conexões

    - Modelagem do tempo de resposta:
        - *Definição de RTT (Round Trip Time): intervalo de tempo entre a ida e a volta de um pequeno pacote entre um cliente e um servidor*
        - Tempo de resposta (por objeto):
        - 1 RTT para iniciar a conexão TCP
        - 1 RTT para o pedido HTTP e o retorno dos primeiros bytes da resposta HTTP
        - Tempo de transmissão do arquivo
            - **tempo de transmissão do arquivo total = 2RTT + tempo de transmissão do arquivo**
- **HTTP persistente**
    - **Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor**
    - O servidor deixa a conexão aberta após enviar a resposta
    - Mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão aberta
    - Pode ser necessário apenas um RTT para todos os objetos referenciados

Mensagem de requisição HTTP:

- Dois tipos de mensagem HTTP: requisição, resposta
- Mensagem de requisição HTTP:
    - ASCII (formato legível por pessoas)

- Método POST:
  - Conteúdo é enviado para o servidor no corpo da mensagem
- Método URL/GET:
  - Conteúdo é enviado para o servidor no campo URL: www.somesite.com/animalsearch?key=monkeys&bananas
- Método HEAD:
  - Requests headers (only) that would be returned if specified URL were requested with an HTTP GET method.
  - Pede para o servidor não enviar o objeto requerido junto com a resposta
- Método PUT:
  - Upload de novo arquivo (objeto) para o servidor
  - Substitui o arquivo que existe na URL especificada com o conteúdo no corpo do POST HTTP

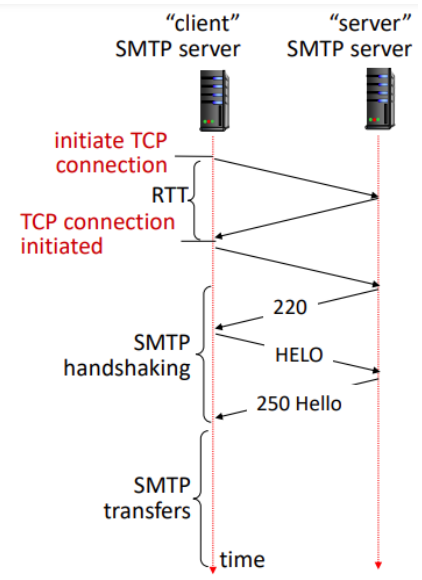
Códigos de status da resposta HTTP: Aparecem na primeira linha da mensagem de resposta servidor->cliente.

- **200 OK: sucesso, objeto pedido segue mais adiante nesta mensagem**
- **301 Moved Permanently: objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (Location: field)**
- **400 Bad Request: mensagem de pedido não entendida pelo servidor**
- **404 Not Found: documento pedido não se encontra neste servidor**
- **505 HTTP Version Not Supported: versão de http do pedido não usada por este servidor**

<u>Correio Eletrônico:</u>

Três grandes componentes:

- Agentes de usuário (UA) a.k.a. “leitor de correio”:
  - Compor, editar, ler mensagens de correio
  - ex., Outlook, Thunderbird, cliente de mail do iPhone
  - Mensagens de saída e chegando são armazenadas no servidor
- Servidores de correio
  - Caixa de correio contém mensagens de chegada (ainda não lidas) p/ usuário
  - Fila de mensagens contém mensagens de saída (a serem enviadas)



- **Simple Mail Transfer Protocol: SMTP**
    - **SMTP entre servidores de correio para transferir mensagens de correio**

- Usa TCP para a transferência confiável de msgs do correio do cliente ao servidor, porta 25
    - Transferência direta: servidor remetente ao servidor receptor
- **Três fases da transferência**
    - **Handshaking (saudação)**
    - **Transferência das mensagens**
    - **Encerramento**
- Mensagens precisam ser em ASCII de 7-bits

Cenário: Alice envia uma msg para Bob

1) Alice usa o UA para compor uma mensagem “para” bob@someschool.edu
2) O UA de Alice envia a mensagem para o seu servidor de correio; a mensagem é colocada na fila de mensagens
3) O lado cliente do SMTP abre uma conexão TCP com o servidor de correio de Bob
4) O cliente SMTP envia a mensagem de Alice através da conexão TCP
5) O servidor de correio de Bob coloca a mensagem na caixa de entrada de Bob
6) Bob chama o seu UA para ler a mensagem

SMTP: últimas palavras

- **SMTP usa conexões persistentes**
- **SMTP requer que a mensagem (cabeçalho e corpo) sejam em ASCII de 7-bits**
- Servidor SMTP usa CRLF.CRLF para reconhecer o final da mensagem
- Comparação com HTTP
    - HTTP: pull (recupera)
    - SMTP: push (envia)
    - Ambos têm interação comando/resposta, códigos de status em ASCII
    - HTTP: cada objeto é encapsulado em sua própria mensagem de resposta
    - SMTP: múltiplos objetos de mensagem enviados numa mensagem de múltiplas partes

DNS: Domain Name System

Identificadores de hospedeiros, roteadores:

- endereço IP (32 bit) - usado p/ endereçar datagramas
- “nome”, ex., www.yahoo.com - usado por gente

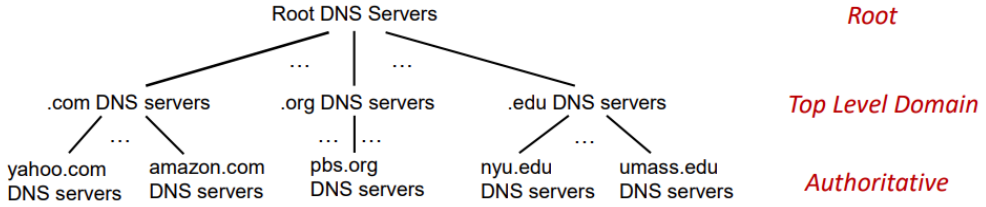
Domain Name System:

- Base de dados distribuída implementada na hierarquia de servidores de nomes
- Protocolo de camada de aplicação permite que hospedeiros, roteadores, servidores de nomes se comuniquem para resolver nomes (tradução endereço/nome)

Serviços DNS

- Tradução de nome de hospedeiro para IP
- Distribuição de carga
    - Servidores Web replicados: conjunto de endereços IP para um mesmo nome
- Por que não centralizar o DNS?
    - Ponto único de falha, Volume de tráfego, Base de dados centralizada e distante, Manutenção (da BD), Não é escalável!

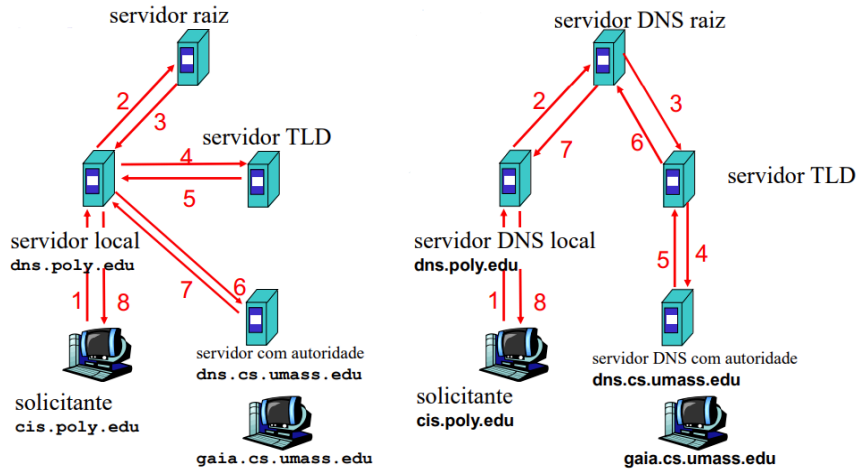
Base de Dados Hierárquica e Distribuída



- Servidores raiz
  - procurado por servidor local que não consegue resolver o nome
  - procura servidor oficial se mapeamento desconhecido
  - obtém tradução
  - devolve mapeamento ao servidor local
  - 13 servidores de nome raiz em todo o mundo
- Servidores de nomes de Domínio de Alto Nível (TLD):
  - Servidores DNS responsáveis por domínios com, org, net, edu, etc, e todos os domínios de países como br, uk, fr, ca, jp.
  - Domínios genéricos: book, globo, rio
- Servidores de nomes com autoridade:
  - Servidores DNS das organizações, provendo mapeamentos oficiais entre nomes de hospedeiros e endereços IP para os servidores da organização (e.x., Web e correio). Podem ser mantidos pelas organizações ou pelo provedor de acesso
- Servidor DNS Local
  - Não pertence necessariamente à hierarquia
  - Cada ISP (ISP residencial, companhia, universidade) possui um.
    - Também chamada do “servidor de nomes default”
  - Quanto um hospedeiro faz uma consulta DNS, a mesma é enviada para o seu servidor DNS local
    - Possui uma cache local com pares de tradução recentes
    - Atua como um intermediário, enviando consultas para a hierarquia.

Exemplo de resolução de nome pelo DNS

- Consulta interativa: servidor consultado responde com o nome de um servidor de contato. “Não conheço este nome, mas pergunte para esse servidor”
- Consulta recursiva: Transfere a responsabilidade de resolução do nome para o servidor de nomes contatado.



Uso de cache, atualização de dados

- Uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa cache local
    - Entradas na cache são sujeitas a temporização (desaparecem) depois de um certo tempo (TTL)
- Entradas na cache podem estar desatualizadas
    - Se o endereço IP de um nome de host for alterado, pode não ser conhecido em toda a Internet até que todos os TTLs expirem

Registros DNS

DNS: BD distribuído contendo registros de recursos (RR)

formato RR: (nome, valor, tipo, ttl)

- Tipo=NS: nome é domínio (p.ex. foo.com.br). valor é endereço IP de servidor oficial de nomes para este domínio
- Tipo=A: nome é nome de hospedeiro. valor é o seu endereço IPv4. Tipo=AAAA para IPv6
- Tipo=CNAME: nome é nome alternativo (alias) para algum nome “canônico” (verdadeiro). www.ibm.com é na verdade servereast.backup2.ibm.com. valor é o nome canônico
- Tipo=MX: valor é nome do servidor de correio associado ao nome

Protocolo e mensagens

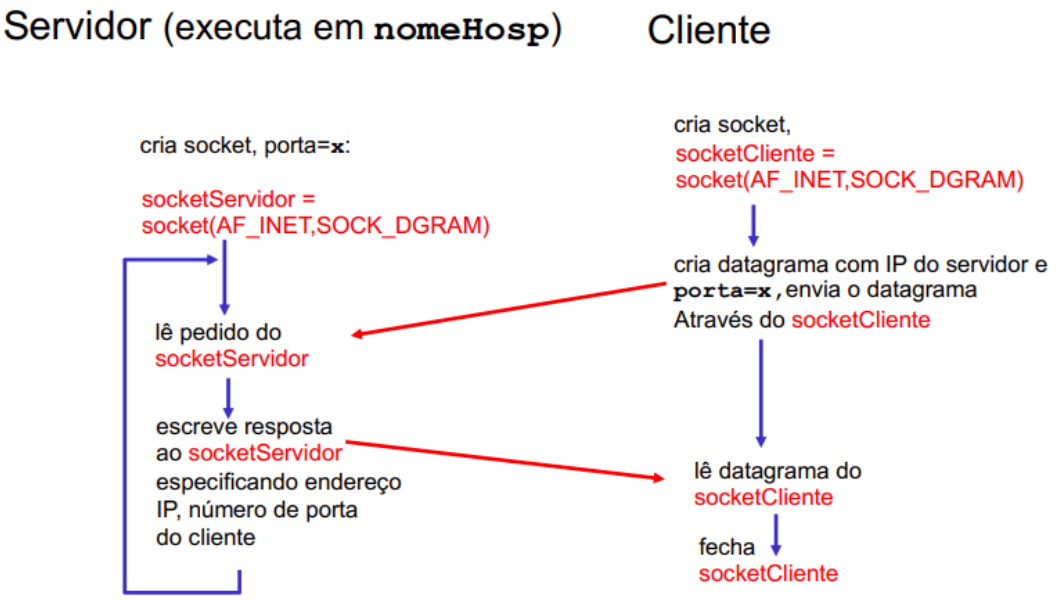
- Cabeçalho de msg:
    - identificação: ID de 16 bit para pedido, resposta ao pedido usa mesmo ID
    - flags: pedido ou resposta, recursão desejada, recursão permitida, resposta é oficial

Ataques ao DNS

- Ataques DDoS
    - Bombardeia os servidores raiz com tráfego
- Bombardeio aos servidores TLD
- Ataques de redirecionamento
    - Pessoa no meio intercepta as consultas
    - Envenenamento do DNS
        - Envia respostas falsas para o servidor DNS que as coloca em cache
- Exploração do DNS para DDoS
    - Envia consultas com endereço origem falsificado: IP alvo

Programação com sockets UDP e TCP

- Socket: porta entre o processo de aplicação e o protocolo de transporte fim-a-fim
- Aplicação Exemplo:
    - 1. o cliente lê uma linha de caracteres (dados) do seu teclado e envia os dados para o servidor
    - 2. o servidor recebe os dados e converte os caracteres para maiúsculas
    - 3. o servidor envia os dados modificados para o cliente
    - 4. o cliente recebe os dados modificados e apresenta a linha na sua tela
- Programação com UDP:
    - UDP: não tem “conexão” entre cliente e servidor
    - Remetente coloca explicitamente endereço IP e porta do destino
    - Servidor deve extrair endereço IP e número da porta do remetente do datagrama recebido
    - UDP: dados transmitidos podem ser recebidos fora de ordem, ou perdidos

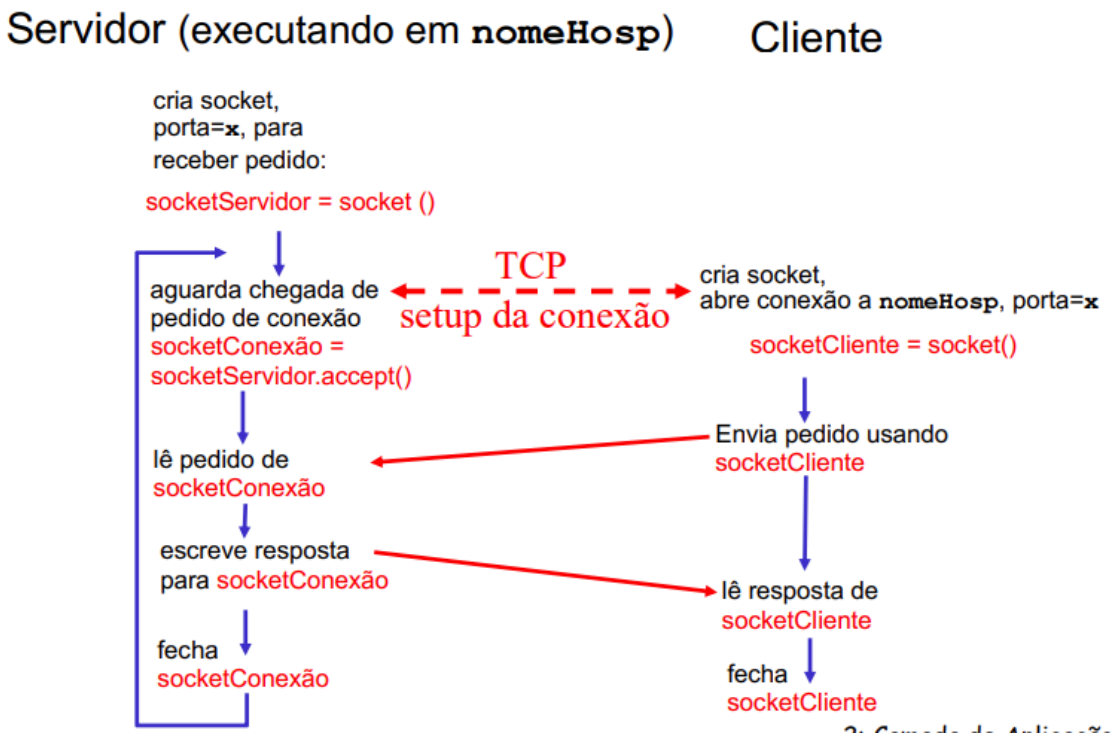



```
from socket import *                                  # inclui a biblioteca de sockets do Python
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET,                 # cria socket UDP para servidor
                      socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')      # obtém entrada do teclado do usuário
clientSocket.sendto(message,(serverName, serverPort)) # acrescenta o nome do servidor e número da porta à mensagem; envia pelo socket
modifiedMessage, serverAddress =                      # lê caracteres de resposta do socket e converte em string
                      clientSocket.recvfrom(2048)
print modifiedMessage                                 # imprime string recebido e
clientSocket.close()                                  # fecha socket
```

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)            # cria socket UDP
serverSocket.bind(('', serverPort))                   # liga socket à porta local número 12000
print "The server is ready to receive"
while 1:                                              # loop infinito
    message, clientAddress = serverSocket.recvfrom(2048)   # lê mensagem do socket UDP, obtendo endereço do cliente (IP e porta do cliente)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)    # retorna string em maiúsculas para este cliente
```

- Programação com TCP:
    - Cliente deve contactar servidor
        - Processo servidor deve antes estar em execução
        - Servidor deve antes ter criado socket (porta) que aguarda contato do cliente
    - Cliente contacta servidor para:

- Criar socket TCP local ao cliente, especificando endereço IP, número de porta do processo servidor
- Quando cliente cria socket: TCP cliente cria conexão com TCP do servidor
- Quando contatado pelo cliente, o TCP do servidor cria um novo socket para que o processo servidor possa se comunicar com o cliente
    - Permite que o servidor converse com múltiplos clientes
    - Endereço IP e porta origem são usados para distinguir os clientes




```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

inclui a biblioteca de sockets do Python

cria socket TCP socket para o servidor, porta remota 12000

não há necessidade de especificar nem o nome do servidor nem a porta

```
                        from socket import *
cria socket TCP de      serverPort = 12000
recepção ──────────►    serverSocket = socket(AF_INET,SOCK_STREAM)
                        serverSocket.bind(('',serverPort))
servidor inicia a escuta
por solicitações TCP ──► serverSocket.listen(1)
                        print 'The server is ready to receive'
loop infinito ─────────► while True:
servidor espera no accept()
por solicitações, um novo
socket é criado no retorno ──► connectionSocket, addr = serverSocket.accept()

lê bytes do socket (mas
não precisa ler endereço
como no UDP) ──────────► sentence = connectionSocket.recv(1024).decode()
                        capitalizedSentence = sentence.upper()
                        connectionSocket.send(capitalizedSentence.
                                                            encode())
fecha conexão para este
cliente (mas não o socket
de recepção) ──────────► connectionSocket.close()
```

Camada de Transporte

<u>Serviços e protocolos de transporte:</u>

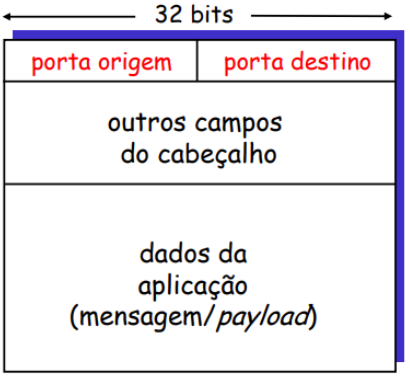
- Fornecem comunicação lógica entre processos de aplicação executando em diferentes hospedeiros
- Os protocolos de transporte são executados nos sistemas finais:
    - Lado transmissor: quebra as mensagens da aplicação em segmentos, repassa-os para a camada de rede
    - Lado receptor: remonta as mensagens a partir dos segmentos, repassa-as para a camada de aplicação
- Camada de rede: comunicação lógica entre hospedeiros (“serviço postal”)
- Camada de transporte: comunicação lógica entre os processos (“quem distribui as cartas para as crianças”)
    - Depende de e estende serviços da camada de rede

Ações da camada de transporte

- Transmissor: passa uma mensagem da camada de aplicação, determina os valores do cabeçalho do segmento, cria o segmento, passa o segmento para o IP
- Receptor: recebe o segmento do IP, checa os valores do cabeçalho, extrai a mensagem, demultiplexa a mensagem para a aplicação via socket

Principais protocolos da Internet

- TCP: Transmission Control Protocol
    - confiável, entrega ordenada, controle de congestionamento, controle de fluxo
    - connection setup
- UDP: User Datagram Protocol
    - não confiável, entrega desordenada, no-frills extension of “best-effort”



formato de segmento TCP/UDP

<u>Multiplexação e Demultiplexação:</u>

- Multiplexação no transmissor: reúne dados de muitos sockets, adiciona o cabeçalho de transporte (usado posteriormente para a demultiplexação)

- Demultiplexação no receptor: Usa info do cabeçalho para entregar os segmentos recebidos aos sockets corretos

Como funciona a demultiplexação:

- Computador recebe os datagramas IP
- Cada datagrama possui os endereços IP da origem e do destino
- Cada datagrama transporta um segmento da camada de transporte
- Cada segmento possui números das portas origem e destino
- O hospedeiro usa os endereços IP e os números das portas para direcionar o segmento ao socket apropriado

Demultiplexação não orientada a conexões (UDP):

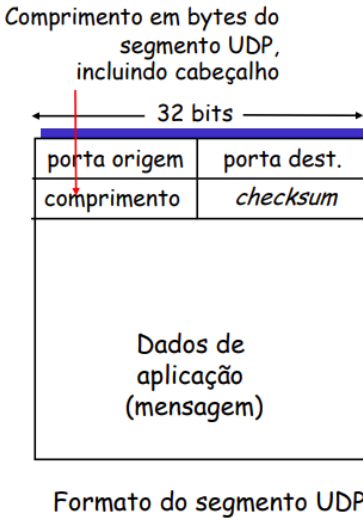
- Quando o hospedeiro recebe o segmento UDP: verifica no. da porta de destino no segmento, encaminha o segmento UDP para o socket com aquele no. de porta
- Datagramas IP com mesmo no. de porta destino, mas diferentes endereços IP origem e/ou números de porta origem podem ser encaminhados para o mesmo socket no destino

Demultiplexação orientada a conexões (TCP):

- Socket TCP identificado pela quádrupla:
  - endereço IP origem
  - número da porta origem
  - endereço IP destino
  - número da porta destino
- Demultiplexação: receptor usa todos os quatro valores para direcionar o segmento para o socket apropriado
- Servidor pode dar suporte a muitos sockets TCP simultâneos:
  - Cada socket é identificado pela sua própria quádrupla
- Servidores Web têm sockets diferentes para cada conexão de cliente
  - HTTP não persistente terá sockets diferentes para cada pedido

Basicamente:

- **Multiplexação, demultiplexação: baseado em segmento, nos valores do cabeçalho do datagrama**
- **UDP: demultiplexação usa apenas número de porta de destino**
- **TCP: demultiplexação usa quádrupla**
- **Multiplexação e demultiplexação ocorre em todas as camadas**



Formato do segmento UDP

UDP: User Datagram Protocol

- Protocolo de transporte da Internet mínimo, “sem gorduras”
- Serviço “melhor esforço”, segmentos UDP podem ser: perdidos, entregu aplicação fora de ordem
- Sem conexão: não há saudação inicial entre o remetente e o receptor U
  - tratamento independente para cada segmento UDP
- Uso do UDP: aplicações de streaming multimídia, DNS, SNMP
- Por quê existe um UDP?
  - Elimina estabelecimento de conexão (que pode causar retardo)
  - Simples: não mantém “estado” da conexão nem no remetente, nem no receptor
  - Cabeçalho de segmento reduzido
  - Não há controle de congestionamento: UDP pode transmitir tão rápido quanto desejado

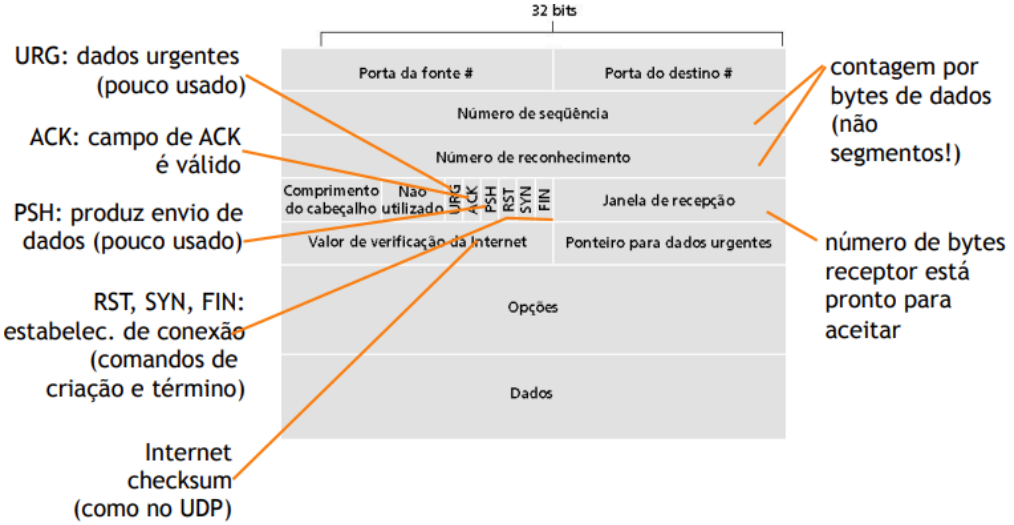
- Checksum: serve para detectar “erros” (ex.: bits trocados) no segmento transmitido
    - Transmissor:
        - trata conteúdo do segmento como sequência de inteiros de 16-bits
        - checksum: soma (adição usando complemento de 1) do conteúdo do segmento
        - transmissor coloca complemento do valor da soma no campo checksum do UDP
    - Receptor:
        - calcula checksum do segmento recebido
        - verifica se o checksum calculado bate com o valor recebido:
        - NÃO - erro detectado
        - SIM - nenhum erro detectado

Ações da camada de transporte (UDP):

- Transmissor: passa a mensagem da aplicação, determina os valores do cabeçalho do segmento, cria o segmento UDP, passa o segmento para o IP
- Receptor: recebe o segmento do IP, checa o checksum, extrai a mensagem da aplicação, demultiplexa a mensagem para a aplicação via socket

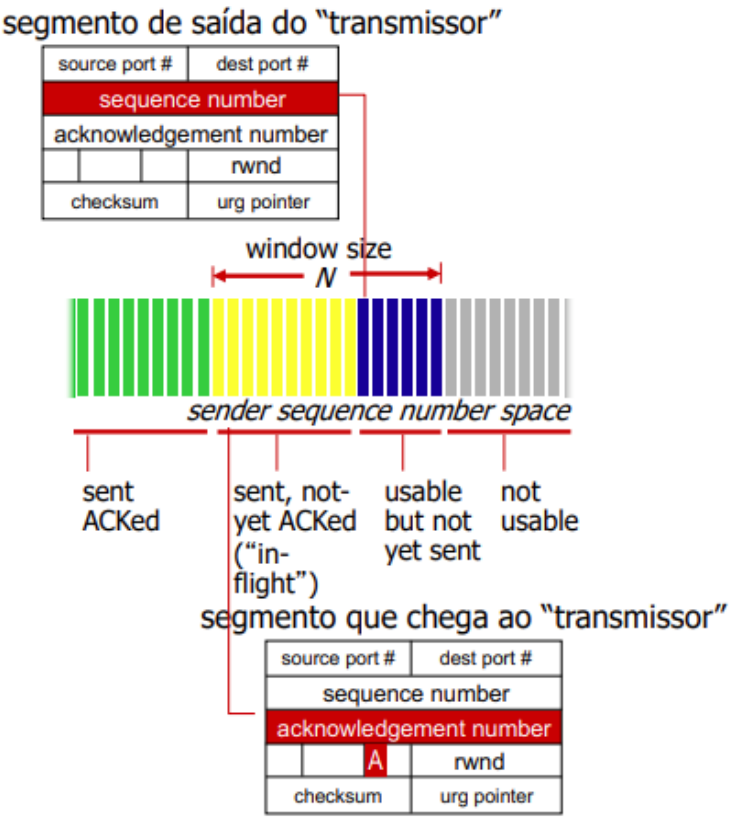
TCP: Visão geral

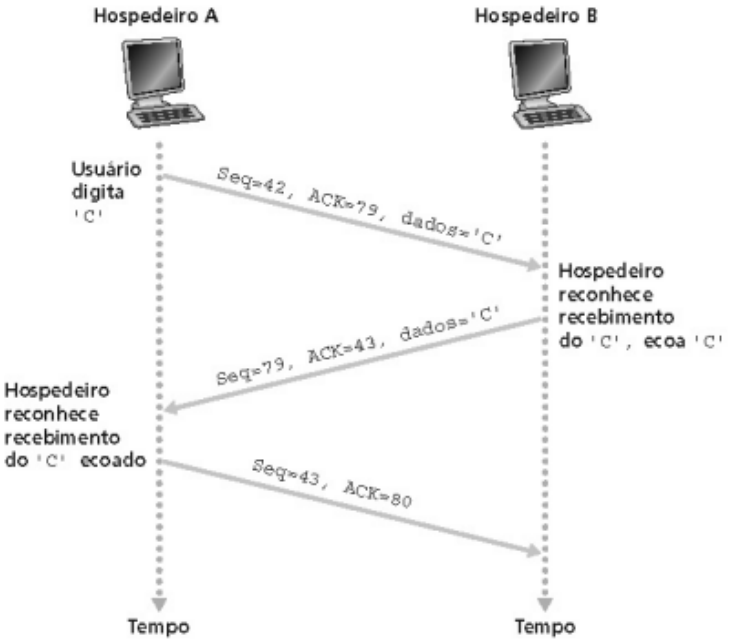
- Ponto a ponto: um transmissor, um receptor
- Fluxo de bytes, ordenados, confiável: não estruturado em msgs
- Com paralelismo (pipelined): tam. da janela ajustado por controle de fluxo e congestionamento do TCP
- Transmissão full duplex: fluxo de dados bi-direcional na mesma conexão, MSS: tamanho máximo de segmento
- Orientado a conexão: handshaking (troca de msgs de controle) inicia estado do transmissor e do receptor antes da troca de dados
- Fluxo controlado: receptor não será afogado pelo transmissor



Números de sequência e ACK:

- **Nos. de seq.: “número” dentro do fluxo de bytes do primeiro byte de dados do segmento**
- **ACKs: no. de seq do próx. byte esperado do outro lado, ACK cumulativo**





cenário telnet simples

Tempo de viagem de ida e volta (RTT – Round Trip Time) e Temporização

- P: Como escolher o valor do temporizador TCP?
  - Maior que o RTT, mas o RTT varia…
  - Muito curto: temporização prematura e retransmissões desnecessárias
  - Muito longo: reação demorada à perda de segmentos
- P: Como estimar RTT?

    - SampleRTT: tempo medido entre a transmissão do segmento e o recebimento do ACK correspondente, ignora retransmissões
    - SampleRTT varia de forma rápida, é desejável um “amortecedor” para a estimativa do RTT

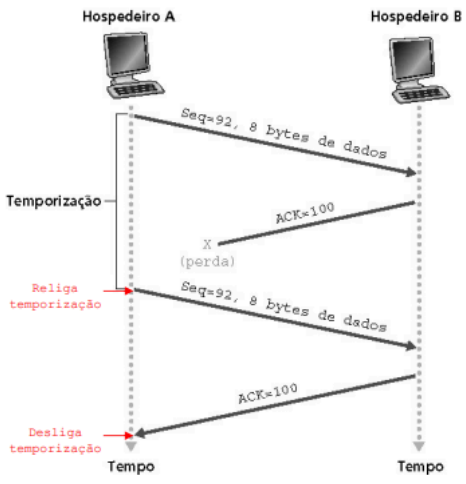
Transferência de dados confiável do TCP

- **O TCP cria um serviço RDT sobre o serviço não confiável do IP**
    - **Segmentos transmitidos em “paralelo” (pipelined)**
    - **Acks cumulativos**
    - **O TCP usa um único temporizador para retransmissões**
- **As retransmissões são disparadas por:**
    - **estouros de temporização**
    - **acks duplicados**

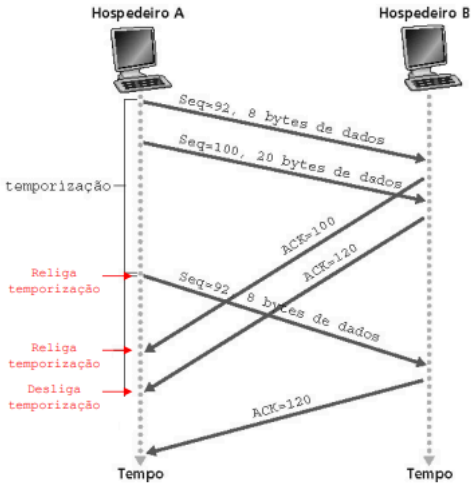
Eventos do transmissor TCP

- Dados recebidos da aplicação:
    - Cria segmento com no. de sequência (nseq)
    - nseq é o número de sequência do primeiro byte de dados do segmento
    - Liga o temporizador se já não estiver ligado (temporização do segmento mais antigo ainda não reconhecido)
- Estouro do temporizador:
    - Retransmite o segmento que causou o estouro do temporizador
    - Reinicia o temporizador
- Recepção de Ack:
    - Se reconhecer segmentos ainda não reconhecidos
        - atualizar informação sobre o que foi reconhecido
        - religa o temporizador se ainda houver segmentos pendentes (não reconhecidos)

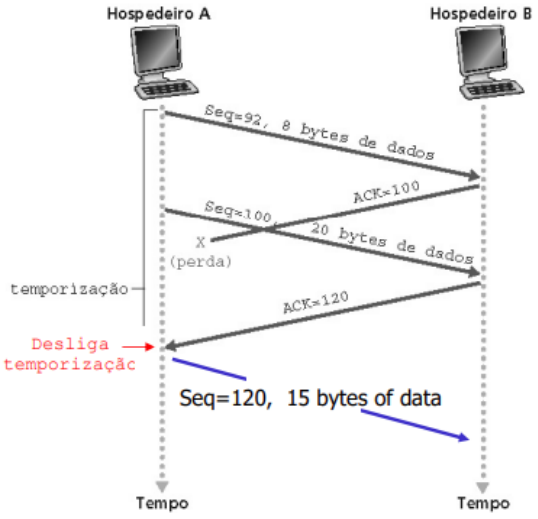
Cenários de retransmissão:



Cenário com perda do ACK



Temporização prematura, ACKs cumulativos



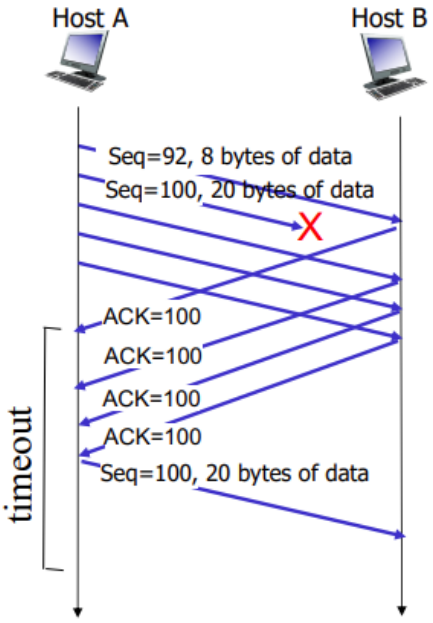
Cenário de ACK cumulativo

TCP geração de ACKs [RFCs 1122, 2581]

Evento no Receptor	Ação do Receptor TCP
chegada de segmento em ordem sem lacunas, anteriores já reconhecidos	ACK retardado. Espera até 500ms pelo próx. segmento. Se não chegar segmento, envia ACK
chegada de segmento em ordem sem lacunas, um ACK retardado pendente	envia imediatamente um único ACK cumulativo
chegada de segmento fora de ordem, com no. de seq. maior que esperado -> lacuna	envia ACK duplicado, indicando no. de seq.do próximo byte esperado
chegada de segmento que preenche a lacuna parcial ou completamente	ACK imediato se segmento começa no início da lacuna

Retransmissão rápida do TCP:

- O intervalo do temporizador é frequentemente bastante longo:
  - longo atraso antes de retransmitir um pacote perdido
- Detecta segmentos perdidos através de ACKs duplicados.
  - O transmissor normalmente envia diversos segmentos
  - Se um segmento se perder, provavelmente haverá muitos ACKs duplicados.
- Se o transmissor receber 3 ACKs para os mesmos dados (“três ACKs duplicados”), retransmite segmentos não reconhecidos com menores nos. de seq.



TCP: Controle de Fluxo do TCP

- Controle de fluxo: o receptor controla o transmissor, de modo que este não inunde o buffer do receptor transmitindo muito e rapidamente
- O receptor “anuncia” o espaço livre do buffer incluindo o valor da rwnd nos cabeçalhos TCP dos segmentos que saem do receptor para o transmissor

    - Tamanho do RcvBuffer é configurado através das opções do socket (o valor default é de 4096 bytes)
- O transmissor limita a quantidade os dados não reconhecidos ao tamanho do rwnd recebido
- Garante que o buffer do receptor não transbordará

TCP: Gerenciamento de Conexões

- Antes de trocar dados, o transmissor e o receptor TCP dialogam: concordam em estabelecer uma conexão e concordam com os parâmetros da conexão.
- 2-way handshake

TCP: Encerrando uma conexão

- Seja o cliente que o servidor fecham cada um o seu lado da conexão
    - enviam segmento TCP com bit FIN = 1
- Respondem ao FIN recebido com um ACK
    - ao receber um FIN, ACK pode ser combinado com o próprio FIN
- Lida com trocas de FIN simultâneos

<u>Princípios de Controle de Congestionamento:</u>

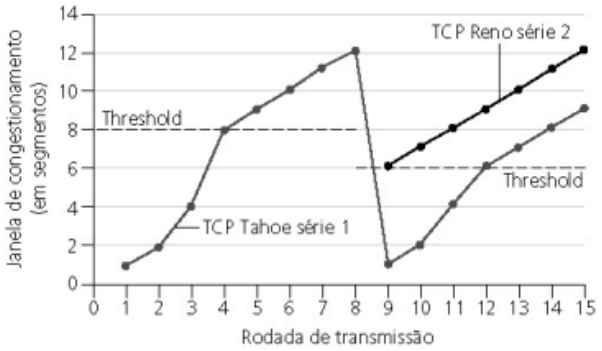
- Congestionamento: informalmente: “muitas fontes enviando dados acima da capacidade da rede de tratá-los”
- Sintomas: perda de pacotes (saturação de buffers nos roteadores), longos atrasos (enfileiramento nos buffers dos roteadores)

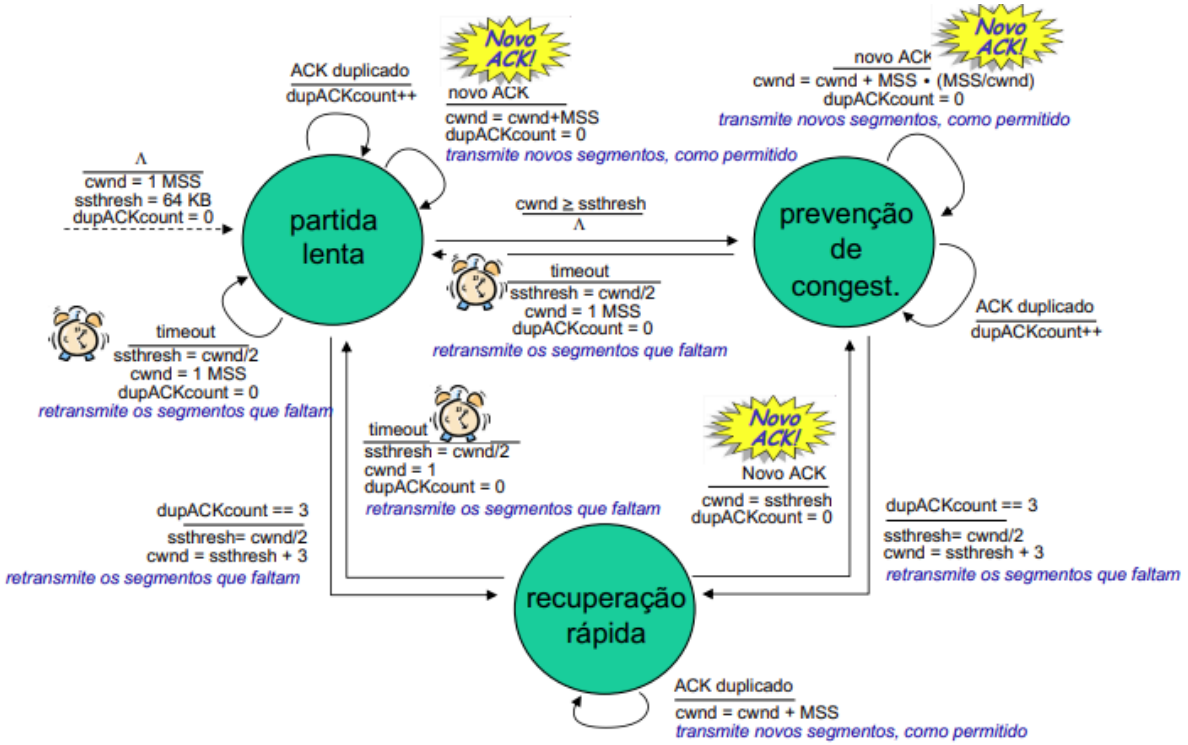
Controle de Congestionamento do TCP: aumento aditivo, diminuição multiplicativa

- Abordagem: aumentar a taxa de transmissão (tamanho da janela), testando a largura de banda utilizável, até que ocorra uma perda
    - aumento aditivo: incrementa cwnd de 1 MSS a cada RTT até detectar uma perda
    - diminuição multiplicativa: corta cwnd pela metade após evento de perda
- **TCP: Partida lenta**
    - **No início da conexão, aumenta a taxa exponencialmente até o primeiro evento de perda:**
        - inicialmente cwnd = 1 MSS
        - duplica cwnd a cada RTT
        - através do incremento da cwnd para cada ACK recebido
        - resumo: taxa inicial é baixa mas cresce rapidamente de forma exponencial
- **TCP: detectando, reagindo a perdas**
    - **perda indicada pelo estouro de temporizador:**
        - **cwnd é reduzida a 1 MSS;**
        - **janela cresce exponencialmente (como na partida lenta) até um limiar, depois cresce linearmente**
    - perda indicada por ACKs duplicados: TCP RENO
        - ACKs duplicados indicam que a rede é capaz de entregar alguns segmentos
        - corta cwnd pela metade depois cresce linearmente
    - O TCP Tahoe sempre reduz a cwnd para 1 (seja por estouro de temporizador que três ACKS duplicados)

P: Quando o crescimento exponencial deve mudar para linear?

R: Quando cwnd atingir 1/2 do seu valor antes da detecção de perda.
Limiar (Threshold) variável (ssthresh)
Com uma perda o limiar (ssthresh) é ajustado para 1/2 da cwnd imediatamente antes do evento de perda.





Pulando o resto do slide porque já deu por hoje